



DOSSIER PROFESSIONNEL (DP)

MINISTÈRE CHARGÉ
DE L'EMPLOI

Nom de naissance

- NECIB

Nom d'usage

- NECIB

Prénom

- Mohamed El-Amine

Adresse

- 73 boulevard de roux, 13004, Marseille

Titre professionnel visé

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)

Pour compléter ce dossier, le candidat dispose d'un site web en accès libre sur le site.

 <http://travail-emploi.gouv.fr/titres-professionnels>

Sommaire

Exemples de pratique professionnelle

Développer une application sécurisée	p.	5
- Développement d'une application de mise en relation candidat/recruteur - SKILLY	p.	p.
- SafeBase – Dump and Dumper	p.	p.
- Intitulé de l'exemple n° 3	p	p.
+ Concevoir et développer une application sécurisée organisée en couches	p.	17
- Développement d'une application de mise en relation candidat/recruteur - SKILLY	p.	p.
- SafeBase – Dump and Dumper	p.	p.
- Intitulé de l'exemple n° 3	p	p.
Préparer le déploiement d'une application sécurisée	p.	24
- Développement d'une application de mise en relation candidat/recruteur - SKILLY	p.	p.
- Projet CI/CD	p.	p.
- Intitulé de l'exemple n° 3	p	p.
Titres, diplômes, CQP, attestations de formation (facultatif)	p.	
Déclaration sur l'honneur	p.	37
Documents illustrant la pratique professionnelle (facultatif)	p.	
Annexes (Si le RC le prévoit)	p.	

DOSSIER PROFESSIONNEL ^(DP)

**EXEMPLES DE PRATIQUE
PROFESSIONNELLE**

DOSSIER PROFESSIONNEL (DP)

Activité-type 1 Développer une application sécurisée

Exemple n°1 - Développement d'une application de mise en relation candidat/recruteur - SKILLY

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

SKILLY est une application mobile de matching dédiée au recrutement dans le secteur tech. Elle met en relation des candidats spécialisés (développeurs, designers, data, etc.) avec des recruteurs d'entreprises tech via une expérience fluide, moderne et engageante. L' objectif de l'application est d'offrir une expérience simple, rapide et ciblée pour trouver un emploi ou recruter un talent tech, en s'inspirant des codes du swipe et de la messagerie directe.

Seuls deux types d'utilisateurs sont possibles :

- **Candidat** : crée un profil, renseigne ses compétences et préférences, swipe des offres, postule, suit ses candidatures, échange avec les recruteurs.

- **Recruteur** : crée des offres, visualise les candidats ayant postulé à une offre spécifique, swipe les profils pertinents, entre en contact via la messagerie.

Skilly embarque différentes fonctionnalités dont les principales sont :

- Un **Swipe intelligent** des offres (pour les candidats) et des candidats (par offre pour les recruteurs)

- Une **Messagerie intégrée** pour faciliter les échanges directs

- **Suivi de candidatures** clair côté candidat

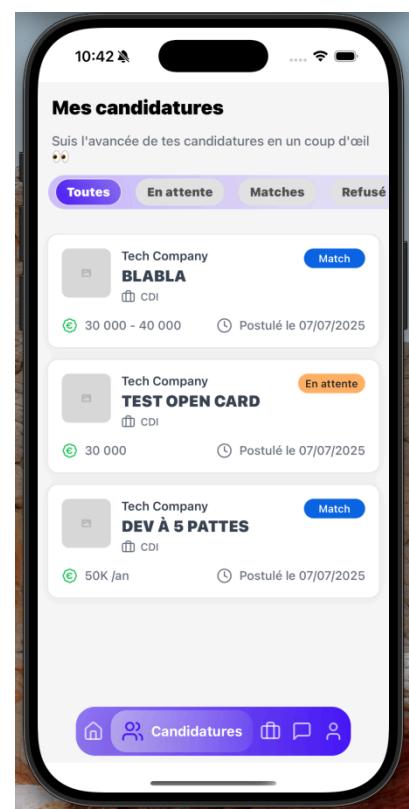
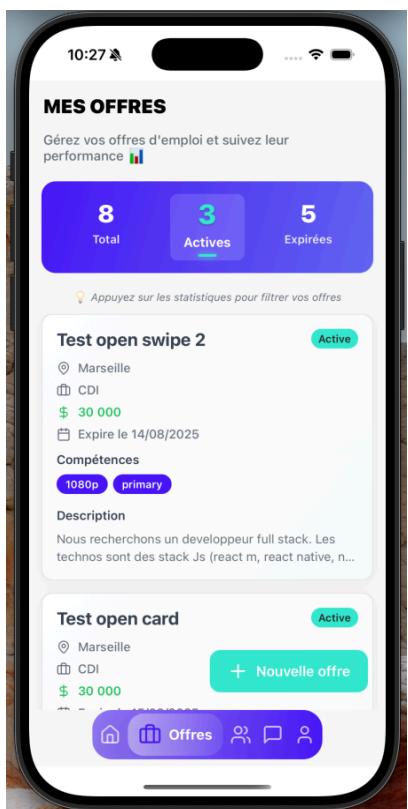
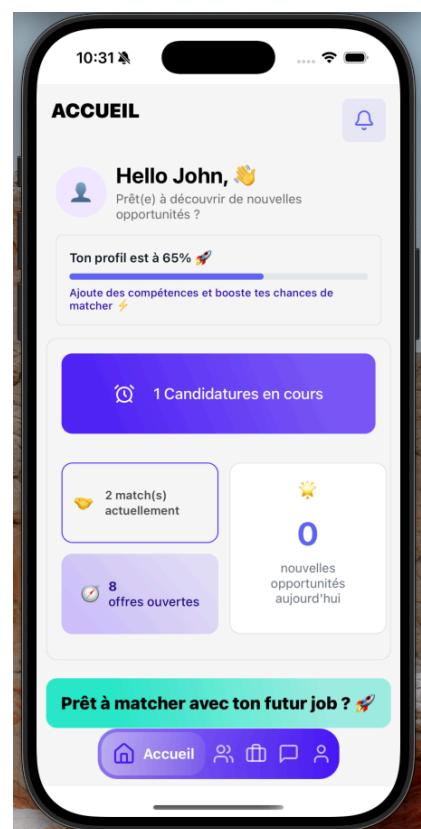
- **Création et gestion d'offres** côté recruteur

- **Onboarding adapté** selon le rôle choisi (candidat ou recruteur)

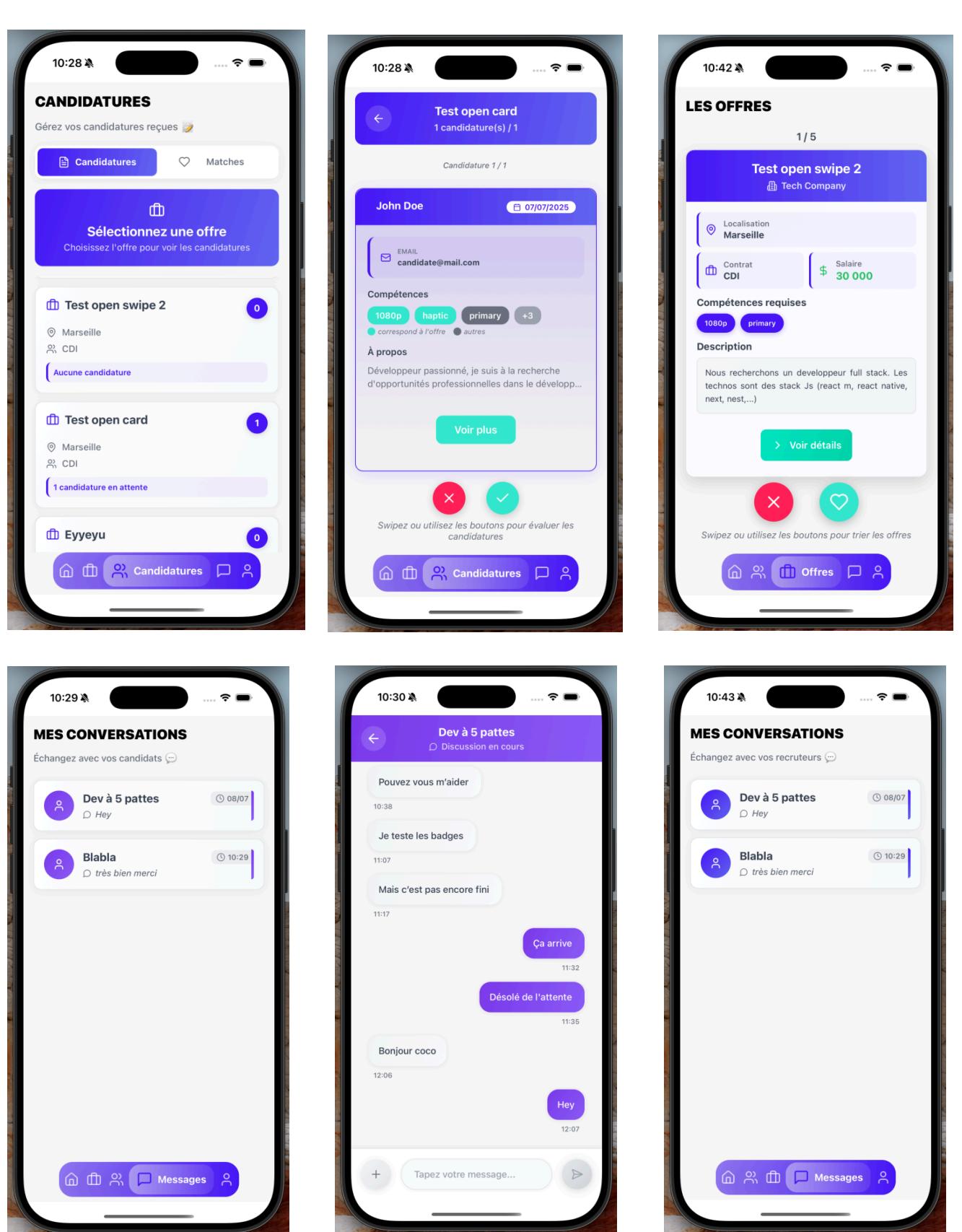
Développement de l'interface utilisateur (côté candidat et recruteur), gestion de la navigation, des formulaires, de l'authentification sécurisée. Mise en place des composants métier : gestion des offres, des candidatures, du matching entre profils et postes. Rédaction de documentation technique et réalisation des tests unitaires. Respect des recommandations ANSSI (sécurisation des formulaires, validation des entrées, RGPD...).

Développer l'interface utilisateur selon le rôle (candidat ou recruteur),

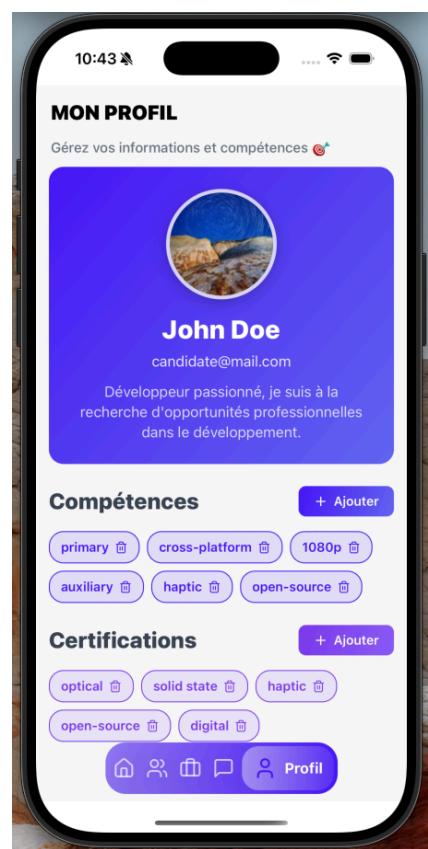
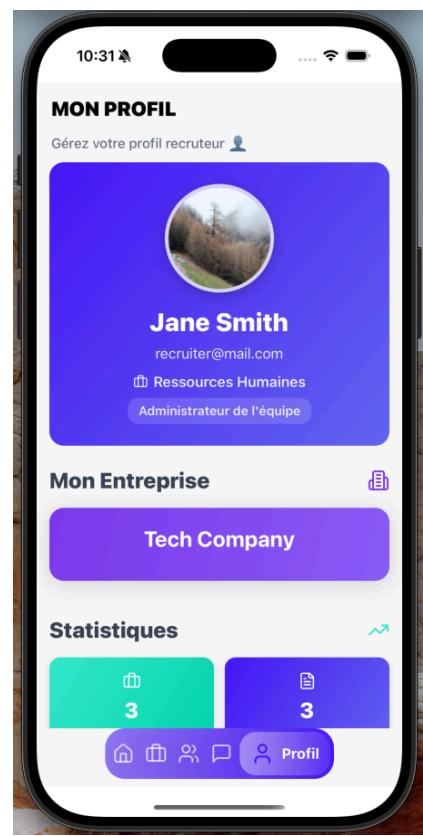
DOSSIER PROFESSIONNEL (DP)



DOSSIER PROFESSIONNEL (DP)



DOSSIER PROFESSIONNEL (DP)



DOSSIER PROFESSIONNEL (DP)

Gérer l'authentification sécurisée avec JWT,



```
1 // Create the token
2     token := jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.MapClaims{
3         "email":      savedUser.Email,
4         "role":       savedUser.Role,
5         "id":        savedUser.ID,
6         "firstName":  savedUser.FirstName,
7         "lastName":   savedUser.LastName,
8         "companyID":  recruiterProfile.CompanyID,
9         "companyRole": recruiterProfile.Role,
10        "recruiterID": recruiterProfile.ID,
11    })
12
13 // Sign the token with a secret key
14 tokenString, err := token.SignedString([]byte(os.Getenv("secret")))
15 if err != nil {
16     c.JSON(500, gin.H{"error": err.Error()})
17 }
18
19
20 c.JSON(200, gin.H{
21     "user":  savedUser,
22     "token": tokenString,
23 })
```

Créer des composants métier : publication d'offres, candidature, matching, messagerie,

Assurer la sécurité des données utilisateur (chiffrement, validation côté front et back),

Réaliser les tests unitaires (back end en Go),

Participer à la planification (GitHub Projects), rédaction de tickets et gestion des branches.

2. Précisez les moyens utilisés :

DOSSIER PROFESSIONNEL (DP)

- **IDE** : Cursor et VSCode
- **Front-end** : React Native + NativeWind + stylesheet classique (selon compatibilité)
- **Back-end** : Go (Gin + GORM + Gorilla)
- **BDD** : PostgreSQL (DBeaver), MongoDB (MongoDB Compass)
- **CI** : GitHub Projects (planification, tickets, branches)
- **Outils** : Postman, Docker, GitHub, Excalidraw, Notion

3. Avec qui avez-vous travaillé ?

Projet mené en trinôme avec deux autres développeurs. Recours ponctuel à un mentor/formateur pour valider certaines orientations techniques et de sécurité.

4. Contexte

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

Chantier, atelier, service ➤ *Dans le cadre de la formation Concepteur Développeur d'Applications*

Période d'exercice ➤ Du : *Janvier 2025* au : *Juin 2025*

5. Informations complémentaires (facultatif)

DOSSIER PROFESSIONNEL (DP)

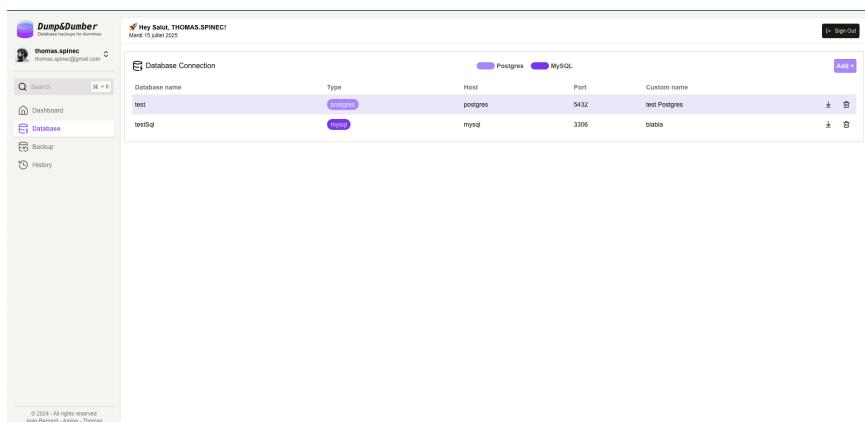
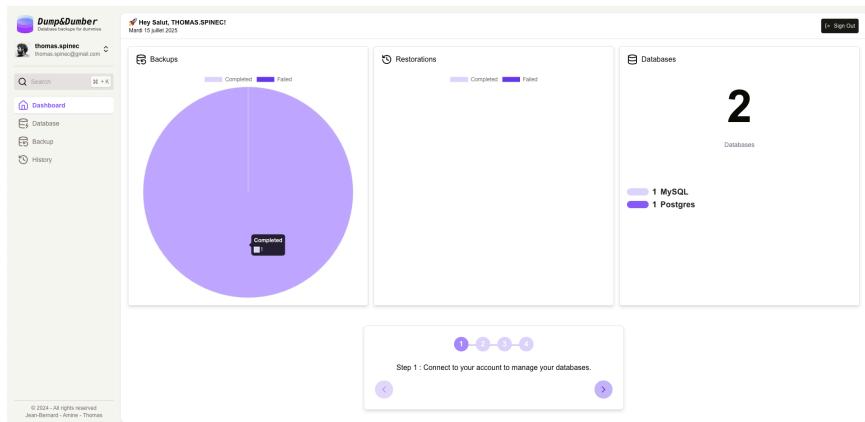
Activité-type 1 Développer une application sécurisée

Exemple n°2 - SafeBase – Dump and Dumper

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

Dans le cadre du projet SafeBase, j'ai développé une **API REST** permettant la **sauvegarde automatisée** et la **restauration de bases de données MySQL et PostgreSQL**. Le front a quant à lui été réalisé grâce à NextJs, le visuel de l'application n'étant pas demandé dans le projet, nous sommes partis sur un design simple et moderne.

Voici le rendu:



DOSSIER PROFESSIONNEL (DP)

The screenshot shows the 'Database Connection' section of the application. A modal window titled 'Add New Database' is displayed, asking for the database name (test), type (MySQL), host, port, user, and password. There are also fields for custom name and port, and buttons for 'Test Connection' and 'Cancel'.

The screenshot shows the 'Backups' section. It displays a table with columns: Name, Schedule, Database name, Creation date, and Action. One backup entry is listed: 'Test' with a schedule of 'Every day at 6am', created on '16 June at 15:54'.

The screenshot shows the 'History' section. It displays a table with columns: Name, Status, Creation date, Source database, and Action. One history entry is shown: 'testSQL_2025-06-16' with a status of 'Success' and a creation date of '16 June at 16:13'.

Mon rôle a été de concevoir les **routes principales** de l'API et d'implémenter :

- Des endpoints sécurisés pour ajouter une base, lancer une sauvegarde ou effectuer une restauration,

DOSSIER PROFESSIONNEL (DP)

- Des scripts systèmes exécutés depuis le backend en Go (`exec.Command`) pour déclencher `pg_dump`, `mysqldump`, `psql`, `mysql`,



```
1  time := time.Now().Local().Format(
2      "2006-01-02T15-04-05")
3
4  cmd := exec.Command("mysqldump", "-h", c.Host,
5      "--port", c.Port, "--user", c.User, "--password="+c.
6      Password, c.Db_name)
7  outfile, err := os.Create("./backups/mysql/" +
8      fileName)
9  defer outfile.Close()
10
11 var stderr bytes.Buffer
12 cmd.Stdout = outfile
13 cmd.Stderr = &stderr
14
15 err = cmd.Run()
```

- Une logique de **versioning des sauvegardes** avec génération de fichiers horodatés,
- La création de **tâches planifiées** (cron job) pour une exécution automatique des sauvegardes

DOSSIER PROFESSIONNEL (DP)



```
1  var Cr gocron.Scheduler
2  var CronList = make(map[int]uuid.UUID)
3
4  func InitCron() {
5      dumpModel := DumpModel{}
6      dumps, err := dumpModel.GetAll()
7      if err != nil {
8          fmt.Println(err)
9      }
10
11     Cr, _ = gocron.NewScheduler()
12     for _, d := range dumps {
13         if !d.Active {
14             continue
15         }
16         job, _ := Cr.NewJob(
17             gocron.CronJob(d.Cron_job, false),
18             gocron.NewTask(func() {
19                 connection := connection.ConnectionModel{}
20                 dbConn, error := connection.GetById(d.Connection_id)
21                 if error != nil {
22                     fmt.Println(error)
23                 }
24                 var result string
25                 if dbConn.Db_type == "postgres" {
26                     result = PostgresDump(&dbConn)
27                 } else if dbConn.Db_type == "mysql" {
28                     result = MysqlDump(&dbConn)
29                 } else {
30                     fmt.Println("Invalid database type")
31                 }
32                 fmt.Println(result)
33             }),
34         )
35
36         CronList[d.Id] = job.ID()
37     }
38     fmt.Println(CronList)
39     Cr.Start()
40 }
```

DOSSIER PROFESSIONNEL (DP)

- Des contrôles de **validation des données** reçues dans les requêtes (connexion, credentials...),
- Une gestion des erreurs et des retours d'exécution avec messages clairs,
- Des **tests manuels** et début de tests fonctionnels automatisés,
- Une logique d'alerte en cas d'échec (log + retour d'erreur détaillé).

Le projet a été mené en équipe, avec planification des tâches via **GitHub Projects**, gestion des branches et relectures croisées.

2. Précisez les moyens utilisés :

Pour développer le projet SafeBase, nous avons utilisé une stack complète et un environnement technique polyvalent :

- **Go (Golang)** avec le framework **Fiber** : pour développer l'API REST performante et légère,
- **Next.js (React)** et **Tailwind CSS** : pour créer une interface utilisateur simple permettant d'interagir avec l'API,
- **npm** : pour la gestion des dépendances côté front-end,
- **Air (Live Reload)** : pour accélérer le développement backend avec recharge automatique des modifications,
- **PostgreSQL et MySQL** : comme bases de données cibles à sauvegarder et restaurer,
- **Commandes système** : exécution de **pg_dump**, **mysqldump**, **psql**, **mysql** via **os/exec** en Go pour déclencher les processus,
- **Docker** : pour conteneuriser l'ensemble de l'application (API, bases de données, volumes de sauvegarde),
- **Postman** : pour tester les routes de l'API manuellement tout au long du développement,

DOSSIER PROFESSIONNEL (DP)

- **GitHub + GitHub Projects** : pour le versionning du code, la planification des tâches et le suivi de l'avancement,
- **Environnements de développement différents** : nous avons travaillé à **trois développeurs** sur des machines aux systèmes d'exploitation différents (**macOS, Linux et Windows**), ce qui nous a amenés à prendre en compte les contraintes de compatibilité multi-plateformes dès le début du projet.

3. Avec qui avez-vous travaillé ?

Projet mené en trinôme avec deux autres développeurs. Recours ponctuel à un mentor/formateur pour valider certaines orientations techniques et de sécurité.

4. Contexte

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

Chantier, atelier, service ➤ *Dans le cadre de la formation Concepteur Développeur d'Applications*

Période d'exercice ➤ Du : *Septembre 2024* au : *Octobre 2024*

5. Informations complémentaires (facultatif)

DOSSIER PROFESSIONNEL (DP)

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

Exemple n° 1 - Développement d'une application de mise en relation candidat/recruteur - SKILLY

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

Dans la phase de conception du projet SKILLY, nous avons réalisé une **étude approfondie des besoins** à l'aide de **deux formulaires distincts** :

- Un adressé aux **recruteurs** (RH, direction, responsables d'équipe),
- Un autre pour les **candidats potentiels** (étudiants, formateurs, professionnels du numérique).

Les retours obtenus nous ont permis de **formaliser des besoins concrets** en termes de fonctionnalités, de parcours utilisateur, et de priorités (matching, messagerie, gestion des candidatures, tableau de bord...).

À partir de cette étude, nous avons :

- Déduit les **cas d'usage** principaux selon les rôles,
- Construit un **Userflow** complet pour représenter les parcours utilisateurs,
- Conçu une **arborescence de navigation** mobile,
- Élaboré un **MCD**, puis un **MLD** et un **MPD** pour la base de données PostgreSQL (cf annexe),
- Défini une **architecture multicouche** (contrôleur/API REST, services métier, accès aux données),
- Structuré le **back-end Go** avec un **service dédié par entité** (utilisateur, offre, candidature, message...), permettant une meilleure séparation des responsabilités et une maintenance facilitée,
- Mis en place une logique de **gestion des rôles et des droits**,
- Documenter les routes principales de l'API.

Nous avons également fait une recherche de l'existant, Le secteur du recrutement en ligne est dominé par de grandes plateformes telles que Indeed, Monster, Jora, Linkedin ... Ces outils répondent aux besoins de mise en relation entre entreprises et candidats, mais il reste souvent basé sur des modèles

DOSSIER PROFESSIONNEL (DP)

traditionnels: formulaires de recherche, candidatures par email ou portail web, échange par mail, sans personnalisation avancée ni expérience utilisateur fluide.

Le tout a été développé dans une démarche collaborative, avec des points réguliers et des itérations sur les modèles.

2. Précisez les moyens utilisés :

Pour concevoir l'architecture fonctionnelle et technique de l'application SKILLY, nous avons utilisé les outils suivants :

- **Lucidchart** : pour modéliser le **Userflow** (parcours utilisateur) ainsi que les schémas **MCD**, **MLD** et **MPD** de la base de données relationnelle et non relationnelle,
- **Figma** : pour créer l'**arborescence de navigation** mobile de l'application en fonction des rôles (candidat, recruteur),
- **PostgreSQL** : base de données utilisée en production, respectant le schéma relationnel conçu,
- **MongoDB** : base de données utilisée pour le système de messagerie
- **DBeaver** : pour interagir avec la base de données en phase de conception et de test,
- **MongoDB Compass** : pour interagir avec la base de données mongoDB
- **Go (Golang)** avec le framework **Gin** : pour la création d'une API REST robuste et sécurisée,
- **GORM** : pour l'abstraction des accès aux données et la gestion de la couche de persistance,
- **Gorilla** : pour la gestion des websockets nécessaire au fonctionnement du système de messagerie
- **Postman** : pour tester les endpoints de l'API tout au long du développement,
- **Architecture modulaire** : développement d'un **service dédié par entité métier** (user, offer, application, message...) selon une structure en couches:

. La couche présentation (front-end)

C'est l'interface utilisateur. Elle gère ce que voit et fait l'utilisateur (clics, formulaires, navigation...). React Native pour l'application mobile. Elle interagit avec l'API avec des appels HTTP.

. La couche métier (business logic)

Elle contient la **logique de l'application** : les règles, traitements, conditions. Dans notre application les services GO gèrent la logique du recrutement (exemple: "si un candidat a postulé, alors ..."). C'est aussi ici que l'on protège les accès et que l'on applique les règles de matching, d'autorisation, etc...

DOSSIER PROFESSIONNEL (DP)

. La couche d'accès aux données (data access)

Elle s'occupe de **parler à la base de données** : lecture, écriture, suppression, mise à jour. Dans SKILLY c'est GORM (notre ORM GO) qui gère cette couche. Elle interagit avec PostgreSQL et fournit les données à la couche métier. Les opérations sur la base de données Mongo sont réalisées avec le package mongo-driver

- **Swagger/OpenAPI** : pour la documentation de l'API (endpoints, méthodes, statuts, authentification),

The screenshot shows the Skilly API documentation generated by Swagger. At the top, there's a navigation bar with the Skilly logo, a 'doc.json' button, and an 'Explore' button. Below the bar, the title 'Skilly API 1.0' is displayed, along with the URL '[Base URL: localhost:8088 /]'. A note says 'To exit full screen, press and hold Esc'. The main content area is titled 'applications' and contains several API endpoints:

- GET /application/jobpost/{id}** Récupérer les candidatures d'une offre
- GET /application/me** Récupérer mes candidatures
- POST /application/{id}** Créer une candidature
- PUT /application/{id}/state** Mettre à jour le statut d'une candidature

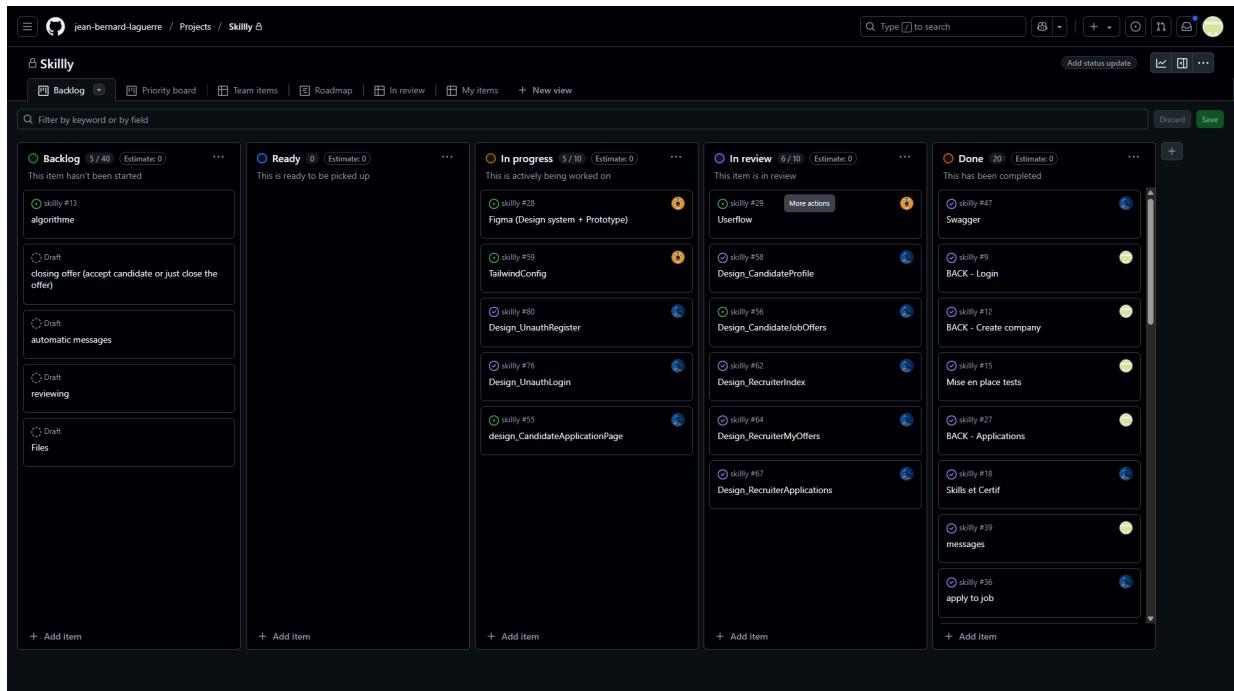
Below the applications section is the 'auth' section, which includes:

- POST /auth/login** Connexion utilisateur
- GET /auth/me** Profil utilisateur actuel
- POST /auth/signup/candidate** Incription candidat
- POST /auth/signup/recruiter** Incription recruteur

At the bottom, there's a 'certifications' section.

DOSSIER PROFESSIONNEL (DP)

- **GitHub Projects** : gestion du projet (tickets, planification, affectation des tâches, création des branches).



3. Avec qui avez-vous travaillé ?

Projet mené en trinôme avec deux autres développeurs. Recours ponctuel à un mentor/formateur pour valider certaines orientations techniques et de sécurité.

4. Contexte

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

Chantier, atelier, service ➔ *Dans le cadre de la formation Concepteur Développeur d'Applications*

Période d'exercice ➔ Du : *Janvier 2025* i au : *Juin 2025*

5. Informations complémentaires (facultatif)

DOSSIER PROFESSIONNEL (DP)

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

Exemple n° 2 ▶ SafeBase – Dump and Dumper

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

Le projet SafeBase visait à créer une **application de gestion de sauvegardes/restaurations de bases de données PostgreSQL et MySQL**. Ce projet s'inspire directement du fonctionnement de **pgbackweb**, un outil open-source développé par **Luis Eduardo (eduardolat)**.

Notre professeur nous a présenté pgbackweb en démonstration en début de projet. À partir de cette base, nous avons :

- Étudié l'**architecture générale** de pgbackweb (fonctionnalités, logique, commandes utilisées),
- Identifié les **composants essentiels** à réimplémenter dans notre propre version,
- Redéfini entièrement le **back-end** avec notre propre stack technique : **Go (Fiber), Docker, commandes système**, etc.

Côté architecture, nous avons mis en place une **structure en couches**, avec :

- Une **couche API REST** (routes Fiber),
- Une **couche métier** : gestion des bases connectées, exécution des sauvegardes/restaurations, versioning,
- Une **couche technique** : appels aux commandes systèmes (**pg_dump, mysqldump**, etc.), gestion des chemins et des fichiers de dump.

Chaque domaine fonctionnel (connexion de base, sauvegarde, restauration) a été encapsulé dans un **service dédié**, garantissant la clarté et l'organisation du code.

Concernant le front-end, nous avons conçu l'interface **en nous basant sur notre propre analyse des besoins utilisateurs**, avec un souci de **simplicité et d'accessibilité**, en proposant des vues de type **dashboard** et des actions claires (ajout de base, déclenchement d'une sauvegarde, etc.).

Enfin, l'ensemble a été **conteneurisé avec Docker**, afin d'assurer la portabilité et la reproductibilité de l'environnement.

DOSSIER PROFESSIONNEL (DP)

2. Précisez les moyens utilisés :

Pour concevoir et développer l'application SafeBase, nous avons mobilisé les outils et technologies suivants :

- **Go (Golang)** avec le framework **Fiber** : utilisé pour structurer l'API REST, séparer les responsabilités via des services dédiés et gérer les middlewares,
- **Commandes système** : utilisation de **pg_dump**, **psql**, **mysqldump**, **mysql** pour exécuter les opérations de sauvegarde et de restauration depuis l'API,
- **Docker** : pour conteneuriser l'environnement complet (API + bases de données MySQL et PostgreSQL), avec volumes dédiés pour stocker les fichiers de dump,
- **Next.js** (React) avec **Tailwind CSS** : pour construire une interface simple sous forme de **dashboard** permettant de visualiser et déclencher les opérations,
- **npm** : pour la gestion des dépendances du front-end,
- **Air (Go live reload)** : pour fluidifier le développement côté backend en rechargeant automatiquement le serveur lors des modifications,
- **Postman** : pour tester manuellement les routes de l'API à chaque étape de développement,
- **Git + GitHub** : pour le versioning du code et le travail collaboratif,
- **GitHub Projects** : pour organiser les tâches, suivre l'avancement du projet et gérer les branches,
- **Environnement multi-plateforme** : le développement a été réalisé en équipe de trois personnes sur des systèmes différents (**macOS**, **Linux**, **Windows**), ce qui nous a poussés à faire attention à la compatibilité dès le départ.

3. Avec qui avez-vous travaillé ?

Projet mené en trinôme avec deux autres développeurs. Recours ponctuel à un mentor/formateur pour valider certaines orientations techniques et de sécurité.

DOSSIER PROFESSIONNEL (DP)

4. Contexte

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

Chantier, atelier, service ▶ *Dans le cadre de la formation Concepteur Développeur d'Applications*

Période d'exercice ▶ Du : *Septembre 2024* au : *Octobre 2024*

5. Informations complémentaires (*facultatif*)

DOSSIER PROFESSIONNEL (DP)

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

Exemple n° 1 - Développement d'une application de mise en relation candidat/recruteur - SKILLY

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

Dans le cadre du projet SKILLY, nous avons mis en œuvre une démarche d'intégration continue (CI) afin de garantir la fiabilité et la qualité du code au fil du développement.

Cette démarche avait pour objectif de faciliter la collaboration, de détecter rapidement les erreurs, et de s'assurer que le code restait fonctionnel après chaque modification.

Mes principales tâches ont été les suivantes :

- Configuration d'un pipeline GitHub Actions, déclenché à chaque push ou pull request sur la branche principale,



```
1 name: Pipeline SKILLY CI/CD
2 on:
3   pull_request:
4     branches:
5       - master
6       - main
7   push:
8     branches:
9       - main
```

- Automatisation des tests backend en Go via des scripts `go test`, couvrant les fonctionnalités principales de l'API,

DOSSIER PROFESSIONNEL (DP)



```
1 package test
2
3 import (
4     "os"
5     "testing"
6
7     "skillly/test/setup"
8     testUtils "skillly/test/utils"
9 )
10
11 func TestMain(m *testing.M) {
12
13     // Setup the tests databases
14     setup.SetupTestPostgres()
15     setup.SetupTestMongo()
16
17     testUtils.InitTestRepositories()
18
19     // Run the tests
20     code := m.Run()
21
22     // Clean up
23     setup.CleanupTestPostgres()
24     setup.CleanupTestMongo()
25
26     // Exit with the test code
27     os.Exit(code)
28 }
29
```

DOSSIER PROFESSIONNEL (DP)



```
1  func TestDB(t *testing.T) {
2      t.Run("PostgresDatabaseConnection", db_test.PostgresDatabaseConnection)
3      t.Run("MongoDatabaseConnection", db_test.MongoDatabaseConnection)
4      t.Run("PostgresTableCheck", db_test.PostgresTableCheck)
5      t.Run("MongoCollectionCheck", db_test.MongoCollectionCheck)
6
7
8 }
9
10 func TestAuth(t *testing.T) {
11     t.Run("RegisterCandidate", auth_test.RegisterCandidate)
12     t.Run("RegisterRecruiter", auth_test.RegisterRecruiter)
13     t.Run("Login", auth_test.Login)
14 }
15
16 func TestUser(t *testing.T) {
17     t.Run("CreateUser", user_test.CreateUser)
18     t.Run(" GetUserByEmail", user_test.GetUserByEmail)
19     t.Run("UpdateUser", user_test.UpdateUser)
20     t.Run("GetAllUsers", user_test.GetAllUsers)
21     t.Run(" GetUserById", user_test.GetUserById)
22 }
```

- Mise en place d'un fichier YAML de workflow précisant les étapes de build, de test, et de vérification du code,

DOSSIER PROFESSIONNEL (DP)



```
1 func CreateUser(t *testing.T) {
2
3     newUser := userDto.CreateUserDTO{
4         Email:      "test@test.com",
5         Password:   "password123",
6         FirstName:  "Test",
7         LastName:   "User",
8         Role:       models.RoleRecruiter,
9     }
10    user, err := testUtils.UserRepo.CreateUser(newUser, config.DB)
11    require.NoError(t, err, "Failed to create user")
12
13    assert.NotNil(t, user, "Expected user to be created")
14
15    assert.Equal(t, newUser.Email, user.Email, "Expected user email to match")
16    assert.Equal(t, newUser.FirstName, user.FirstName, "Expected user first name to match")
17    assert.Equal(t, newUser.LastName, user.LastName, "Expected user last name to match")
18    assert.NotEqual(t, newUser.Password, user.Password, "Expected user password to be hashed")
19 }
```

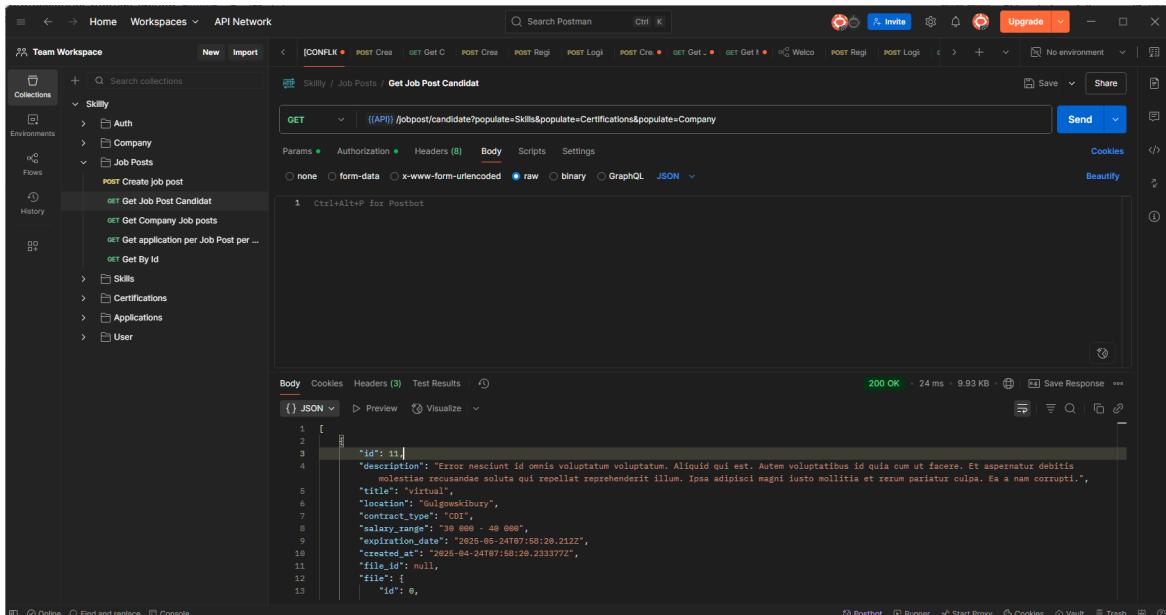
- Utilisation de Docker pour conteneuriser l'environnement backend et s'assurer de sa reproductibilité (Go + PostgreSQL),

DOSSIER PROFESSIONNEL (DP)

```
1 services:
2   back:
3     container_name: Server
4     build:
5       context: .
6       dockerfile: Dockerfile.dev
7     ports:
8       - "8080:8080"
9     volumes:
10      - ./app
11     environment:
12       - NODE_ENV=development
13     depends_on:
14       - postgres
15       - mongodb
16
17   postgres:
18     image: postgres:16.4
19     container_name: skillly_db
20     environment:
21       POSTGRES_USER: ${DB_USER}
22       POSTGRES_PASSWORD: ${DB_PASSWORD}
23       POSTGRES_DB: ${DB_NAME}
24     ports:
25       - "3308:5432"
26     volumes:
27       - db_data:/var/lib/postgresql/data
28
29   mongodb:
30     image: mongo:latest
31     container_name: skillly_mongo
32     environment:
33       MONGO_INITDB_ROOT_USERNAME: ${DB_USER}
34       MONGO_INITDB_ROOT_PASSWORD: ${DB_PASSWORD}
35       MONGO_INITDB_DATABASE: ${DB_NAME}
36     ports:
37       - "27018:27017"
38     volumes:
39       - ./docker-entrypoint-initdb.d/:/docker-entrypoint-initdb.d/:ro
40       - db_data:/mongo-data/db
41     volumes:
42       db_data:
```

DOSSIER PROFESSIONNEL (DP)

- Création d'un environnement de test local pour simuler le fonctionnement de l'application en conditions proches de la production,
- Tests manuels des routes API avec Postman, pour s'assurer du bon fonctionnement des endpoints après chaque modification.



2. Précisez les moyens utilisés :

Pour mettre en place l'intégration continue de l'application SKILLY, j'ai utilisé les outils suivants :

- **GitHub Actions** : pour créer un pipeline CI automatisé, déclenché à chaque modification du code (push/pull request),
- **Go test** : pour exécuter les tests unitaires sur les routes et services du backend écrit en Go,
- **Fichier YAML** : configuration du workflow CI décrivant les étapes d'installation, de build, et d'exécution des tests,
- **Docker** : pour conteneuriser le backend et les base de données PostgreSQL et MongoDB, assurant la reproductibilité de l'environnement de test,
- **Postman** : pour effectuer des tests manuels sur les endpoints de l'API REST (authentification, création d'offres, candidatures...),
- **GitHub** : gestion du versioning, des branches, des pull requests, et suivi des erreurs éventuelles

DOSSIER PROFESSIONNEL (DP)

via les logs CI.

3. Avec qui avez-vous travaillé ?

Projet mené en trinôme avec deux autres développeurs. Recours ponctuel à un mentor/formateur pour valider certaines orientations techniques et de sécurité.

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 sécurisée

Exemple n° 2 - CI/CD – Dump and Dumper

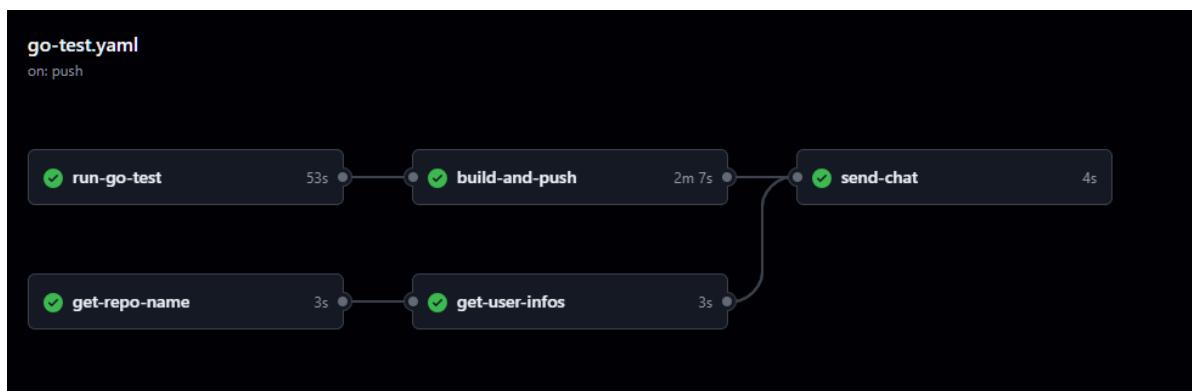
1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Dans le prolongement du projet SafeBase, j'ai mis en place une **pipeline d'intégration continue (CI)** avec **GitHub Actions**, dans le but d'automatiser les vérifications de qualité et de faciliter le travail collaboratif.

Mon objectif était de garantir un fonctionnement fiable et reproduitible du projet à chaque modification, tout en posant les bases d'une future mise en production (CD).

Les tâches réalisées comprennent :

- La **configuration d'un pipeline CI** exécuté automatiquement à chaque **push** sur la branche **main**,



- L'écriture d'un **workflow GitHub Actions** au format YAML incluant les étapes de build, test et packaging,

```
on:
  pull_request:
    branches:
      - master
      - main
```

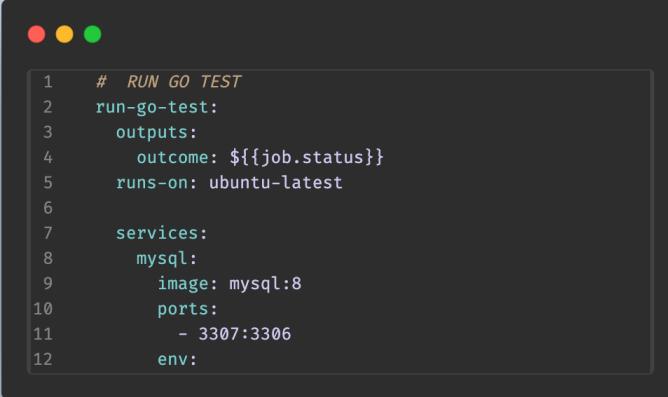
DOSSIER PROFESSIONNEL (DP)

```
● ● ●  
1 jobs:  
2   # GET REPO NAME  
3   get-repo-name:  
4     runs-on: ubuntu-latest  
5     outputs:  
6       repo-name: ${{ github.repository }}  
7     steps:  
8       - name: Checkout code  
9         uses: actions/checkout@v2  
10  
11      - name: Get repo name  
12        run:  
13          echo "Repo name is ${{ github.repository }}"  
14  
14 id: get-repo-name
```

```
● ● ●  
1   # GET USER INFOS  
2   get-user-infos:  
3  
4     # récupérer les infos de l'utilisateur venant du repo  
5     needs: get-repo-name  
6     runs-on: ubuntu-latest  
7     outputs:  
8       user: ${{ github.actor }}  
9     steps:  
10       - name: Checkout code  
11         uses: actions/checkout@v2  
12  
13      - name: Get user infos  
14        run: echo "User infos are ${{ github.actor }}"  
14 id: get-user-infos
```

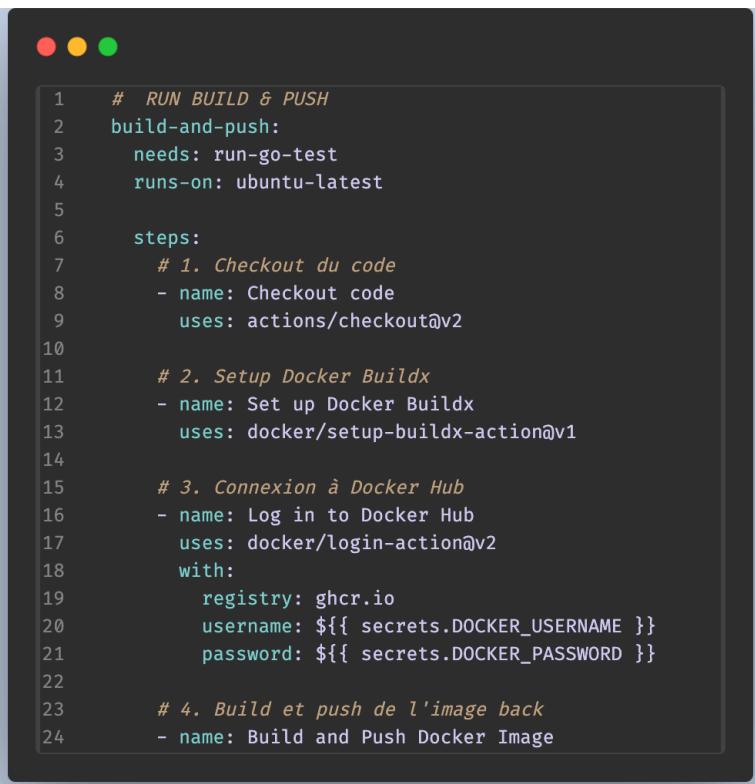
- L'exécution automatique des tests (back-end Go) avec arrêt immédiat de la pipeline en cas d'échec,

DOSSIER PROFESSIONNEL (DP)



```
1 # RUN GO TEST
2 run-go-test:
3   outputs:
4     outcome: ${{job.status}}
5   runs-on: ubuntu-latest
6
7 services:
8   mysql:
9     image: mysql:8
10    ports:
11      - 3307:3306
12
13 env:
```

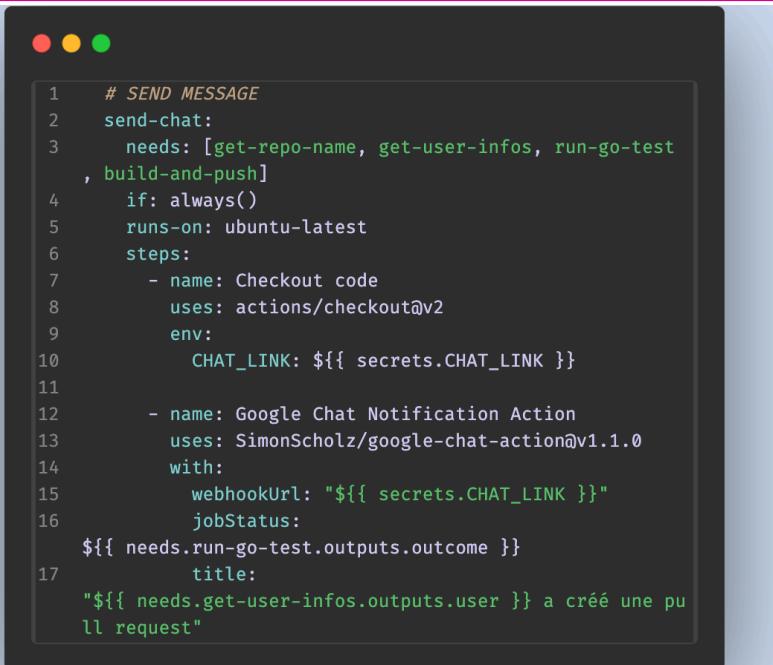
- La construction d'une image Docker du projet (API SafeBase) adaptée à l'environnement de développement,
- Le push automatique de l'image Docker vers GitHub Container Registry (GHCR),



```
1 # RUN BUILD & PUSH
2 build-and-push:
3   needs: run-go-test
4   runs-on: ubuntu-latest
5
6   steps:
7     # 1. Checkout du code
8     - name: Checkout code
9       uses: actions/checkout@v2
10
11    # 2. Setup Docker Buildx
12    - name: Set up Docker Buildx
13      uses: docker/setup-buildx-action@v1
14
15    # 3. Connexion à Docker Hub
16    - name: Log in to Docker Hub
17      uses: docker/login-action@v2
18      with:
19        registry: ghcr.io
20        username: ${{ secrets.DOCKER_USERNAME }}
21        password: ${{ secrets.DOCKER_PASSWORD }}
22
23    # 4. Build et push de l'image back
24    - name: Build and Push Docker Image
```

- L'intégration de notifications Google Chat pour indiquer le succès ou l'échec de la pipeline, en précisant le commit, le statut, et les logs d'erreur si besoin,

DOSSIER PROFESSIONNEL (DP)



```
1 # SEND MESSAGE
2 send-chat:
3   needs: [get-repo-name, get-user-infos, run-go-test
4 , build-and-push]
5   if: always()
6   runs-on: ubuntu-latest
7   steps:
8     - name: Checkout code
9       uses: actions/checkout@v2
10    env:
11      CHAT_LINK: ${{ secrets.CHAT_LINK }}
12
13    - name: Google Chat Notification Action
14      uses: Simonscholz/google-chat-action@v1.1.0
15      with:
16        webhookUrl: "${{ secrets.CHAT_LINK }}"
17        jobStatus:
18          ${{ needs.run-go-test.outputs.outcome }}
19          title:
20            "${{ needs.get-user-infos.outputs.user }} a créé une pu
21 ll request"
```

- La documentation de la configuration dans un dépôt public GitHub, incluant les fichiers **Dockerfile**, **.github/workflows**, et les scripts associés.

L'automatisation mise en place permet aujourd'hui de garantir une qualité de code constante, une exécution reproductible du projet, et une meilleure réactivité de l'équipe lors des développements.

2. Précisez les moyens utilisés :

Pour automatiser l'intégration continue du projet SafeBase, j'ai utilisé les outils et technologies suivants :

- **GitHub Actions** : pour créer un pipeline CI automatisé déclenché à chaque **push** sur la branche **main**,
- **Go test** : pour exécuter les tests unitaires du back-end Go dans la pipeline,
- **Fichiers YAML** : pour définir le workflow CI (**.github/workflows/ci.yml**) incluant les étapes de build, test, et push d'image Docker,

DOSSIER PROFESSIONNEL (DP)

- **Docker** : pour construire une image de l'API SafeBase à partir du **Dockerfile** défini dans le projet,
- **GitHub Container Registry (GHCR)** : pour héberger les images Docker construites à chaque exécution de la pipeline,
- **Google Chat API** : pour envoyer automatiquement des notifications avec le statut de la pipeline, le commit concerné, et les messages d'erreur éventuels,
- **GitHub** : pour le versioning, la gestion du dépôt public, et le déclenchement automatique de la pipeline CI,
- **Postman** (en parallèle) : pour vérifier manuellement les endpoints après les déploiements locaux à partir des images.

3. Avec qui avez-vous travaillé ?

Projet mené en trinôme avec deux autres développeurs. Recours ponctuel à un mentor/formateur pour valider certaines orientations techniques et de sécurité.

4. Contexte

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

Chantier, atelier, service ▶ *Dans le cadre de la formation Concepteur Développeur d'Applications*

Période d'exercice ▶ Du : *Septembre 2024* au : *Octobre 2024*

5. Informations complémentaires (facultatif)

DOSSIER PROFESSIONNEL (DP)

Titres, diplômes, CQP, attestations de formation

(facultatif)

DOSSIER PROFESSIONNEL (DP)

Déclaration sur l'honneur

Je soussigné(e) [prénom et nom] Mohamed El-Amine NECIB , 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 28/07/2025

pour faire valoir ce que de droit.

Signature :



DOSSIER PROFESSIONNEL ^(DP)

Documents illustrant la pratique professionnelle

(facultatif)

Intitulé

Cliquez ici pour taper du texte.

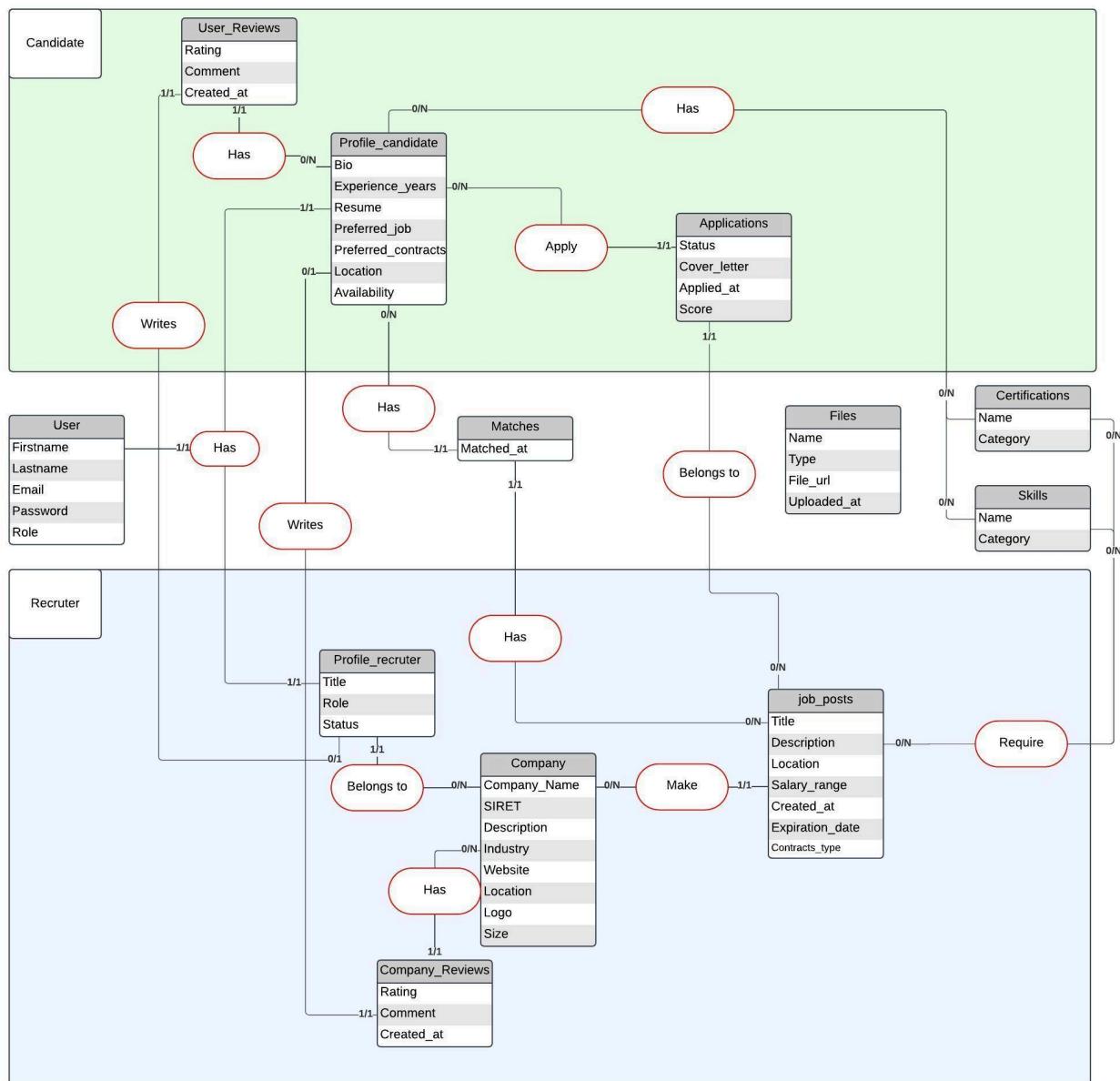
DOSSIER PROFESSIONNEL (DP)

ANNEXES

(Si le RC le prévoit)

MCD:

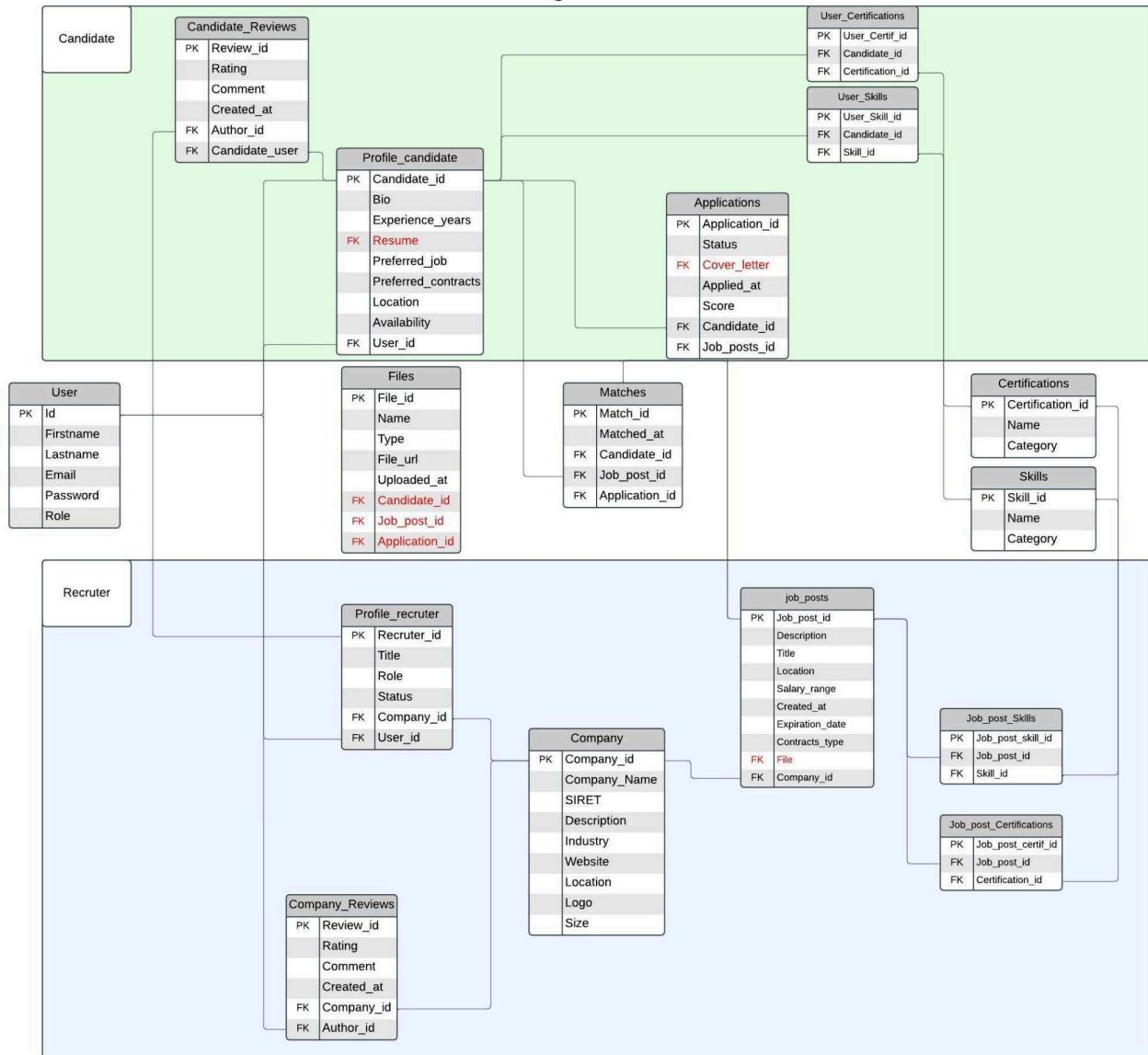
SQL



DOSSIER PROFESSIONNEL (DP)

MLD

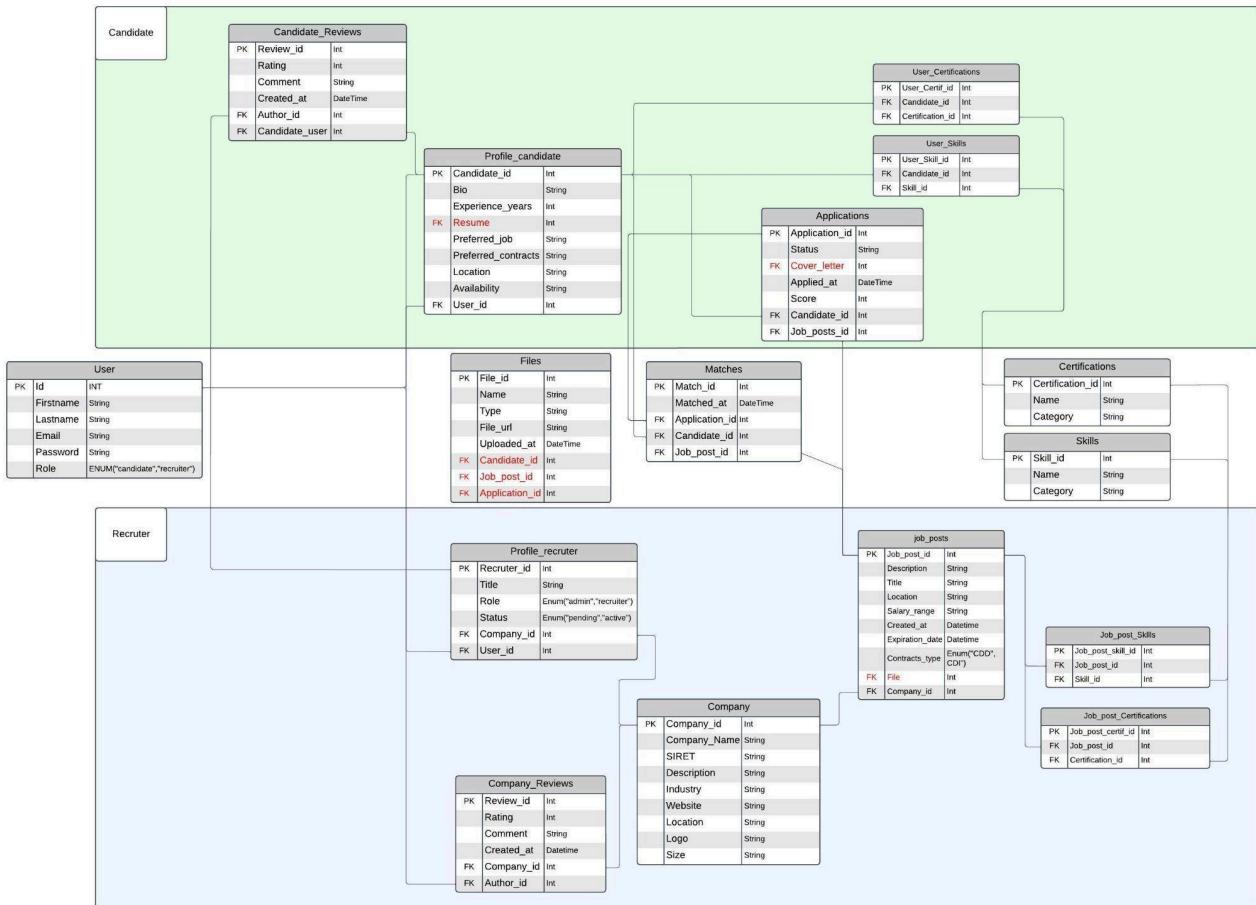
SQL



DOSSIER PROFESSIONNEL (DP)

MPD

SQL



DOSSIER PROFESSIONNEL (DP)

Partie NoSQL

