Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE CHARGÉ
DE L'EMPLOI

DOSSIER PROFESSIONNEL (DP)

Nom de naissance - *OUATTARA*
Nom d'usage -
Prénom - *Boubacar*
Adresse - *4565 Rte du puy St Reparade puyricard*
13540 Aix-en-provence

Titre professionnel visé

Concepteur Développeur Informatique

MODALITÉ D'ACCÈS :

X Parcours de formation

☐ Validation des Acquis de l'Expérience (VAE)

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

SAFEBASE

DOSSIER PROFESSIONNEL (DP)

Documents illustrant la pratique professionnelle *(facultatif)* **p.**

Annexes **p.**

- Architecture globale de l'application
- Fichier `docker-compose.yaml` (extrait)
- Script SQL de création de la base SafeBase
- Exemple de tâche cron pour backups automatiques
- Exemple de test de connexion à une base
- Exemple de test Postman
- Structure des dossiers backend/frontend

EXEMPLES DE PRATIQUE PROFESSIONNELLE

DOSSIER PROFESSIONNEL (DP)

Activité-type 1 Présentation du projet SafeBase

1. 1 Nom du projet

SafeBase – Plateforme de gestion de sauvegardes et de restaurations de bases de données PostgreSQL et MySQL

1.2 Contexte

Dans un monde où les données sont au cœur de toutes les activités numériques, **les sauvegarder et les sécuriser est devenu essentiel**. Une simple erreur humaine, une panne ou une attaque peut entraîner la **perte de données critiques** pour une entreprise — avec des conséquences graves : financières, juridiques ou organisationnelles.

Pour répondre à ce besoin, j'ai conçu **SafeBase**, une application web complète (frontend + backend), développée dans le cadre de ma formation de **Concepteur Développeur d'Applications**.

SafeBase permet de :

- **sauvegarder** régulièrement les bases de données,
- **restaurer** rapidement en cas de problème,
- **planifier** les sauvegardes automatiquement,
- **suivre** l'état des opérations,
- et **consulter l'historique** des sauvegardes/restaurations.

L'application prend en charge les bases **PostgreSQL** et **MySQL**, qui sont très répandues dans les entreprises pour stocker des informations sensibles (clients, ventes, utilisateurs etc).

1.3 Objectifs

L'objectif principal de **SafeBase** est d'offrir aux utilisateurs une solution simple et efficace pour **gérer la sauvegarde et la restauration de bases de données PostgreSQL et MySQL**, à travers une interface moderne et accessible.

L'application permet de réaliser facilement les actions suivantes :

- 🔌 **Ajouter et configurer des connexions** à plusieurs bases de données PostgreSQL ou MySQL (locales ou distantes).
- 💾 **Lancer des sauvegardes**, qu'elles soient manuelles ou planifiées via des **jobs automatisés (cron)**.
- ♻️ **Restaurer une base de données** à partir d'un fichier de sauvegarde précédemment généré.
- 📋 **Consulter l'historique des opérations** : chaque action (sauvegarde ou restauration) affiche son **statut** (réussie, échouée, en cours) et ses **logs d'exécution**.
- ⬇️ **Télécharger les fichiers de sauvegarde** générés automatiquement par l'application.
- 🧭 **Gérer plusieurs connexions** à des bases de données via une **interface centralisée**, sans avoir besoin de passer par la ligne de commande.

En résumé

SafeBase propose une **interface claire, centralisée et automatisée** pour répondre aux besoins suivants :

- Planifier des **sauvegardes régulières**.
- Lancer des **restaurations manuelles ou automatiques**.
- **Superviser et suivre** les opérations réalisées.
- **Centraliser la gestion** de plusieurs bases, qu'elles soient locales ou distantes.
- Simplifier le travail des **administrateurs système, développeurs** ou toute personne chargée de la gestion des bases de données.

1.4 Architecture générale

SafeBase s'appuie sur une **architecture moderne en microservices**, entièrement **conteneurisée** à l'aide de **Docker** et **Docker Compose**, ce qui facilite son déploiement, sa maintenance et son évolutivité.

L'application est composée de plusieurs éléments clés :

- 🧠 **Un backend développé en Go** : il fournit une **API REST** permettant de gérer les connexions aux bases, les sauvegardes, les restaurations, et l'historique des opérations.
- 🎨 **Un frontend Angular** : il consomme l'API du backend pour offrir une **interface utilisateur claire, moderne et facile à utiliser**, accessible depuis un navigateur web.
- 🗃️ **Des bases de données PostgreSQL et MySQL simulées** : intégrées dans l'environnement Docker, elles servent à tester et démontrer les fonctionnalités de sauvegarde/restauration dans des conditions réalistes.
- ⏰ **Un système de planification automatisée** : grâce à des **cron jobs**, l'application peut effectuer des **sauvegardes régulières** sans intervention manuelle.
- 🚀 **Un environnement de production prêt à l'emploi** : grâce au fichier `docker-compose.prod.yml`, le déploiement complet de l'application en production est rapide et standardisé.

1.5 Public cible

SafeBase est conçu pour répondre aux besoins d'un large éventail de profils techniques au sein des organisations. Il s'adresse principalement à :

- 🛠️ **Administrateurs de bases de données (DBA)**

Ce sont les premiers concernés par la gestion des sauvegardes et restaurations. SafeBase leur permet de centraliser et d'automatiser ces tâches critiques, tout en réduisant les risques d'erreur humaine.

- ⚙️ **Développeurs DevOps**
  Ils cherchent souvent à **intégrer la sauvegarde dans des pipelines CI/CD** ou à automatiser la gestion des environnements de données. Grâce à son architecture Dockerisée et à ses API REST, SafeBase s'intègre facilement dans ce type de workflow.

- 🧩 **Responsables des systèmes d'information (SI)**
  SafeBase leur offre une **vision claire et traçable** des sauvegardes de données. Il permet de répondre aux exigences de sécurité, de conformité (RGPD, PRA/PCA), et de qualité de service, tout en facilitant la supervision.

- 🏢 **Petites et moyennes organisations**
  Qui ne disposent pas forcément d'outils sophistiqués ou de ressources humaines pour gérer manuellement les dumps de bases. SafeBase leur apporte une **solution simple, fiable et accessible** pour sécuriser leurs données, sans devoir développer des scripts maison.

SafeBase vise toute structure qui utilise des bases de données PostgreSQL ou MySQL et souhaite :

- éviter la perte de données critiques,
- automatiser les sauvegardes et restaurations,
- gérer plusieurs bases depuis une seule interface,
- et renforcer sa stratégie de sécurité des données.

1.6 Enjeux techniques

Le projet **SafeBase** a été conçu pour répondre à des enjeux techniques concrets, rencontrés dans de nombreux environnements professionnels. Chaque choix technologique vise à garantir **sécurité, performance, fiabilité** et **simplicité d'intégration** :

1.6.1. Sécurité des données

La perte de données critiques peut avoir des conséquences lourdes.
SafeBase assure leur protection grâce à :

- des **dumps horodatés** pour identifier chaque sauvegarde dans le temps,
- des **fichiers compressés** pour limiter l'espace disque utilisé,
- un **stockage centralisé** et organisé, facilitant la gestion et la restauration rapide en cas d'incident.

1.6.2. Robustesse et performance du backend

Le backend est développé en **Go (Golang)**, un langage moderne reconnu pour :

- sa **rapidité d'exécution**,
- sa **gestion efficace de la concurrence** (goroutines),
- sa **fiabilité** dans les environnements de production.
  Ce choix technique garantit des traitements stables, même pour de gros volumes de données ou des bases multiples.

1.6.3. Automatisation des sauvegardes

Grâce à l'intégration de **cron jobs**, SafeBase permet :

- la **planification automatique des sauvegardes** (quotidiennes, hebdomadaires, etc.),
- une **réduction de la charge opérationnelle**,
- une **meilleure régularité et fiabilité** du processus de backup.

1.6.4. Interfaçage et extensibilité via API

SafeBase expose une **API REST claire et documentée**, facilitant :

- l'intégration avec d'autres outils ou tableaux de bord,
- le développement de fonctionnalités externes (scripts, automatisations CI/CD),
- l'accès à ses services par des clients web, CLI ou tiers.

1.6.5. Déploiement rapide et standardisé

Le projet est entièrement **conteneurisé avec Docker**, ce qui permet :

- un **lancement rapide** de l'application sur n'importe quel environnement,
- une **gestion cohérente des dépendances** (bases, backend, frontend),
- une **publication automatique** via GitHub Container Registry pour faciliter la mise à jour et le déploiement continu en production.

SafeBase ne se contente pas d'apporter une solution fonctionnelle. Il repose sur des fondations techniques solides qui garantissent :

- la **sécurité** des données,
- la **fiabilité** de l'exécution,
- l'**automatisation des tâches récurrentes**,
- une **intégration facile** dans des infrastructures existantes,
- et une **mise en œuvre rapide**.

1.7 Mon rôle dans le projet

Dans le cadre du développement de **SafeBase**, j'ai assuré **pas mal de cycle du projet**, de la réflexion initiale à la mise en production. Ce travail m'a permis de mobiliser et de consolider mes compétences en conception, développement, déploiement et documentation.

Voici les différentes responsabilités que j'ai prises en charge :

1.7.1 Analyse des besoins et conception

- **Écoute du besoin utilisateur** : identification des enjeux liés à la sauvegarde et la restauration des bases de données.
- **Rédaction des spécifications fonctionnelles et techniques** : définition des fonctionnalités clés, de l'architecture globale et des flux de données.

1.7.2. Développement backend (Go)

- Création d'une **API REST robuste** en Go (Golang) pour gérer :
  - les connexions aux bases,
  - les sauvegardes/restaurations,
  - l'historique des opérations.
- **Gestion des dumps** avec les outils `pg_dump`, `mysqldump`, `psql`, `mysql`.

1.7.3. Développement frontend (Angular)

- Création de l'interface utilisateur avec **Angular**, intégrant :
  - **PrimeNG** pour les composants graphiques,
  - des formulaires réactifs pour la saisie des connexions,
  - un tableau d'historique interactif.
- Connexion du frontend à l'API via le service HTTP Angular.

1.7.4 Conteneurisation et orchestration

- Mise en place d'un environnement **Dockerisé complet** :
  - Backend Go,
  - Frontend Angular,

- Services de bases de données (PostgreSQL et MySQL).

- Utilisation de **Docker Compose** pour gérer l'ensemble des services (multi-conteneurs).
  - Préparation d'un fichier `docker-compose.prod.yml` pour le déploiement en production.

1.7.5 Modélisation des bases de données

- Conception des **modèles relationnels** nécessaires au stockage des logs et des connexions.
- **Rédaction des scripts SQL** pour l'initialisation des bases PostgreSQL et MySQL intégrées dans l'environnement de test.

1.7.6. Documentation technique

- Rédaction d'un **README clair et structuré** (installation, usage, structure du projet).
- Création d'un fichier `.env.example` et documentation des variables d'environnement.
- Explication des scripts SQL et de leur exécution automatique.

J'ai travaillé en binôme sur toutes les étapes du projet, ce qui m'a permis de :

- Travailler en autonomie,
- Apprendre à résoudre des problèmes concrets en environnement réel,
- Mener un projet complet de bout en bout avec des technologies professionnelles.

1.8 Technologies utilisées

🖥️ **Backend (serveur API)**

- **Langage :** Go (Golang)
- **Framework :** net/http + outils internes
- **Sauvegardes :** `pg_dump`, `mysqldump`
- **Restauration :** `psql`, `mysql`
- **Authentification :** JWT (si nécessaire)
- **Stockage des fichiers :** système de fichiers local (volume Docker)

💻 **Frontend (interface web)**

- **Technologie :** Angular avec Vite
- **Bibliothèques UI :** PrimeNG, Lucide Icons
- **Connexion API :** HTTPClient (Angular)

🐳 **Environnement de développement**

- **Conteneurisation :** Docker & Docker Compose
- **Bases supportées :**
  - PostgreSQL (version 16)
  - MySQL (via images officielles)

Activité-type 2 Réalisation des activités professionnelles

2.1 Bloc 1 : Développement Frontend (Angular)

2.1 Bloc 1 : Développement Frontend (Angular)

🎯 Objectif du bloc

Ce bloc de compétences vise à **concevoir et développer une interface utilisateur** :

- Responsive (adaptée à tous les écrans),
- Accessible (ergonomie, navigation clavier, feedback visuel),
- Moderne (framework, composants UI),
- Dynamique (affichage des données via des appels API REST),
- Maintenable (code modulaire et lisible).

🧪 Mise en œuvre dans le projet SafeBase

Pour ce projet, j'ai développé l'ensemble du **frontend** avec **Angular 18**, en m'appuyant sur :

- **PrimeNG** pour les composants visuels (tableaux, toasts, dialogues…),
- **PrimeFlex** pour la gestion du layout responsive,

- **Lucide-Angular** pour les icônes SVG modernes.

L'objectif était d'offrir une interface simple et fluide pour gérer des sauvegardes/restaurations de bases de données via API.

📁 Conception et organisation du code

Le projet est structuré de manière modulaire autour du dossier `src/` :

- `components/` → composants réutilisables (ex. : `BackupCard`, `DatabaseTable`, `ToastNotification`)
- `services/` → communication avec l'API REST (`DatabaseService`, `BackupService`, `RestoreService`)
- `screens/` → vues principales (`HomeScreen`, `DatabaseScreen`, `HistoryScreen`)
- `context/` → gestion d'état global (`ThemeContext`, `UserContext`)
- `constants/` → constantes de configuration (URL de l'API, types de base, etc.)

Cette architecture respecte les bonnes pratiques Angular et facilite l'évolutivité du projet.

🚀 Fonctionnalités principales développées

- 🔍 **Liste des bases configurées** (PostgreSQL/MySQL) sous forme de tableau.
- ➕ **Ajout d'une base** via formulaire avec validation dynamique.
- 🔄 **Test de connexion en temps réel**, avec message de succès ou d'erreur.
- 💾 **Lancement manuel de sauvegardes ou restaurations**, avec retour visuel immédiat.
- 📄 **Affichage des backups** : taille, date, statut, lien de téléchargement.
- 🔔 **Système de notifications** (toasts) pour les retours d'action utilisateur.
- 🌙 **Thème sombre/clair** dynamique via classes CSS conditionnelles et stockage local.

🛠️ Technologies et outils utilisés

Outil / Lib	Rôle
Angular 18	Framework principal du SPA
PrimeNG	Composants UI (tables, boutons, dialogues, toasts, etc.)
PrimeFlex	Layout responsive
Lucide-Angular	Icônes SVG modernes
Angular Forms	Formulaires réactifs avec validation intégrée
HttpClientModule	Appels API REST (GET, POST, etc.)
CSS personnalisé	Thème, animations, gestion du mode sombre
Prettier	Formatage automatique du code

📱 Accessibilité et responsivité

- Mise en page **100% responsive** avec PrimeFlex et media queries CSS.
- Navigation **entièrement au clavier** possible.
- Couleurs contrastées, **messages ARIA** pour les notifications, et textes alternatifs intégrés pour l'accessibilité.

🔗 Interaction avec le backend (API Go)

Les **services Angular** communiquent avec l'API REST exposée par le backend :

- `DatabaseService` → pour créer, tester et gérer les connexions aux bases.
- `BackupService` → pour lancer une sauvegarde et afficher les fichiers disponibles.

- `RestoreService` → pour lancer une restauration depuis une sauvegarde.

Les services utilisent **HttpClient** avec `catchError()` pour gérer les erreurs côté utilisateur (toast + console) de façon élégante et non bloquante.

🧠 **Compétences mobilisées**

- Analyse des besoins utilisateur et **maquettage d'interface**.
- Création d'une **SPA responsive et modulaire** avec Angular.
- **Consommation d'API REST sécurisée** (via services Angular).
- Création de **formulaires complexes** avec validation côté client.
- Mise en œuvre de **CSS dynamique** pour gérer des thèmes personnalisés.
- Utilisation de `npm`, `Angular CLI`, et gestion du formatage avec **Prettier**.
    - Respect des **bonnes pratiques de lisibilité et de structure du code**.

Captures d'écran / extraits de code à insérer

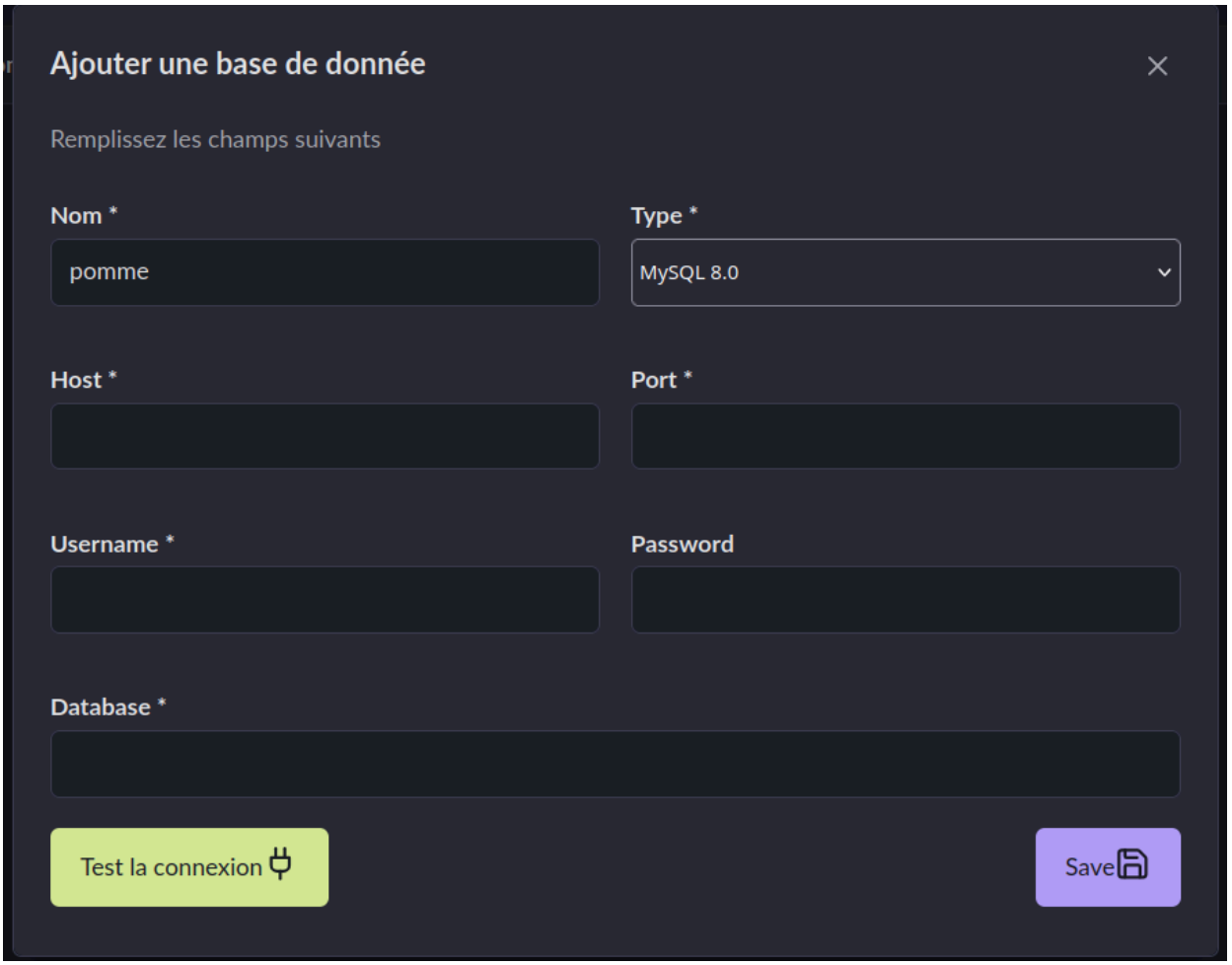
Extrait du formulaire réactif (FormGroup, Validators)

```
  this.databaseForm = new FormGroup({
    name: new FormControl('', Validators.required),
    type: new FormControl('', Validators.required),
    host: new FormControl('', Validators.required),
    port: new FormControl('', Validators.required),
    username: new FormControl('', Validators.required),
    password: new FormControl(''),
    database_name: new FormControl('', Validators.required),
  });
}
```

Ce bloc montre :

- La création d'un **formulaire réactif (`FormGroup`)** avec **`FormControl`**

- L'utilisation de **Validators** intégrés **(`required`)**

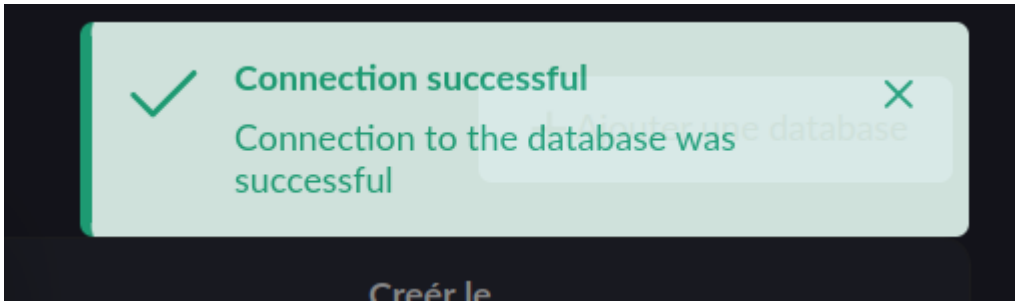


Méthode `testConnection()`

```
100   testConnection() {
101     if (this.databaseForm.valid) {
102       this.databaseService.testConnection(this.databaseForm.value).subscribe({
103         next: (data) => {
104           this.messageService.add({
105             severity: 'success',
106             summary: 'Connection successful',
107             detail: 'Connection to the database was successful',
108           });
109         },
110         error: (error) => {
111           this.messageService.add({
112             severity: 'error',
113             summary: 'Connection failed',
114             detail: error.error.error,
115           });
116         },
117       });
118     }
```

Ce bloc illustre :

- Appel à un **service Angular (API REST)** via `HttpClient`
- Gestion **réactive** des erreurs/succès avec `MessageService`
- **Vérification de la validité du formulaire**



Soumission du formulaire (`onSubmit()`)

```
65    onSubmit() {
66      if (this.databaseForm.valid) {
67        this.databaseService.addDatabase(this.databaseForm.value).subscribe({
68          next: (data) => {
69            this.databaseAdded.emit(); // Émet l'événement
70            this.messageService.add({
71              severity: 'success',
72              summary: 'Database added',
73              detail: 'Database added successfully',
74            });
75            this.visible = false; // Fermez le dialog après soumission
76          },
77          error: (error) => {
78            console.log('Database add failed', error);
79            console.log(error.error);
80            this.messageService.add({
81              severity: 'error',
82              summary: 'Database add failed',
83              detail: error.error.error,
84            });
85          },
86        });
87      } else {
88        // Marquez tous les champs comme touchés pour afficher les erreurs
89        Object.values(this.databaseForm.controls).forEach((control) => {
90          control.markAsTouched();
91        });
92        this.messageService.add({
93          severity: 'error',
94          summary: 'Form validation failed',
95          detail: 'Please fill in all required fields',
96        });
97      }
```

Ce bloc prouve :

- Que tu sais **interagir avec une API REST**
- Gérer les **retours utilisateurs (succès, échec, validation)**

Compétences mobilisées

- **Analyser les besoins et maquetter une application.**
- **Développer des interfaces utilisateur web dynamiques.**
- **Réaliser des formulaires avec validation.**
- **Communiquer avec des API REST.**
- **Mettre en œuvre une interface responsive.**
- **Mettre en œuvre un thème CSS dynamique.**

- **Utiliser un gestionnaire de paquets (npm).**
- **Assurer la lisibilité et la qualité du code frontend (Prettier, structure modulaire).**

2.2 Bloc 2 : Conception et gestion des bases de données (SQL)

🎯 **Objectif du bloc**

Ce bloc de compétences consiste à :

- Concevoir un **modèle de données** pertinent (MCD, MLD, MPD).
- Créer des bases relationnelles respectant les contraintes d'intégrité (clés primaires, étrangères).
- Écrire et exécuter des **requêtes SQL** (création, insertion, sélection, modification, suppression).
- Travailler avec plusieurs **SGBD** (PostgreSQL, MySQL).
- Automatiser l'initialisation des bases à l'aide de **scripts SQL** intégrés au déploiement Docker.

Mon Modele MCD/ MLD

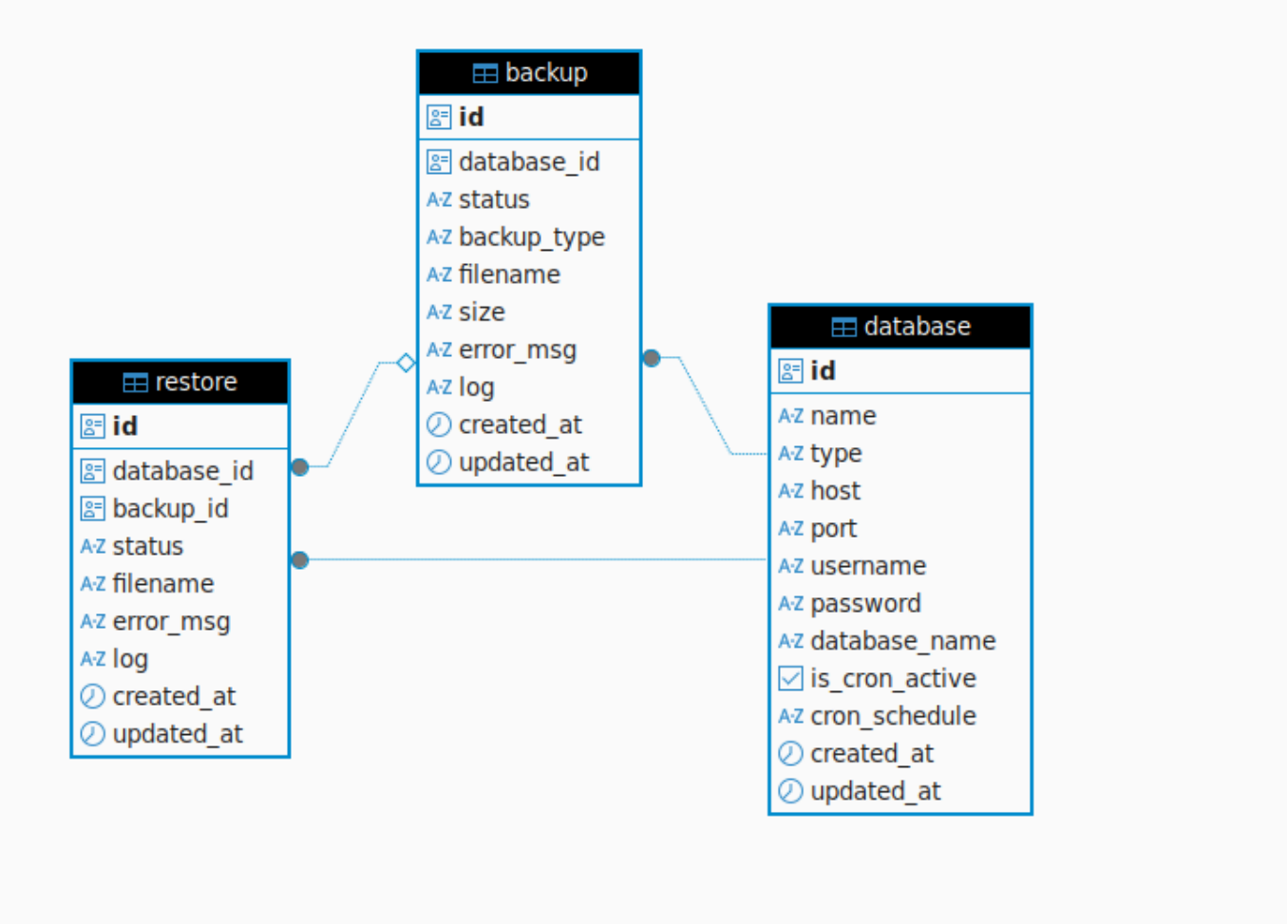


TABLE database (
id UUID PRIMARY KEY,
name TEXT,
type TEXT,
host TEXT,
port TEXT,
username TEXT,
password TEXT,
database_name TEXT,

is_cron_active BOOLEAN,
cron_schedule TEXT,
created_at TIMESTAMP,
updated_at TIMESTAMP
);

TABLE backup (
id UUID PRIMARY KEY,
database_id UUID REFERENCES database(id),
status BACKUP_STATUS,
backup_type BACKUP_TYPE,
filename TEXT,
size TEXT,
error_msg TEXT,
log TEXT,
created_at TIMESTAMP,
updated_at TIMESTAMP
);

TABLE restore (
id UUID PRIMARY KEY,
database_id UUID REFERENCES database(id),
backup_id UUID REFERENCES backup(id),
status RESTORE_STATUS,
filename TEXT,
error_msg TEXT,
log TEXT,
created_at TIMESTAMP,
updated_at TIMESTAMP
);

MCD (Modèle Conceptuel de Données)

Le MCD représente les entités et les relations sans se soucier des détails techniques de la base de données.

✳ **Entités :**

- **Database**
  - Identifiée par `id`
  - Contient : nom, type (MySQL/PostgreSQL), hôte, port, nom d'utilisateur, mot de passe, nom de la BDD distante, statut du cron, etc.
- **Backup**
  - Identifiée par `id`
  - Liée à une `Database`
  - Contient : type de backup, statut, nom de fichier, taille, logs, erreur, etc.
- **Restore**
  - Identifiée par `id`
  - Liée à une `Database` et à un `Backup`
  - Contient : statut, nom de fichier restauré, log, message d'erreur, etc.

🔗 **Relations :**

- **Une Database peut avoir plusieurs Backups (1,N)**
- **Une Database peut avoir plusieurs Restores (1,N)**
- **Un Restore est associé à un seul Backup (0,1)**

🛠️ Mise en œuvre dans le projet SafeBase

Le projet SafeBase repose sur une architecture multi-bases :

Base	Type	Rôle
safebase_db	PostgreSQL	Base principale de l'application (stocke les utilisateurs, bases, backups…)
postgres_db	PostgreSQL	Base de test à sauvegarder/restaurer
mysql_db	MySQL	Deuxième base externe simulant un autre environnement de sauvegarde

🧩 Conception du modèle relationnel

La base `safebase_db` repose sur les entités suivantes :

Entité	Description
User	Gestion des utilisateurs (email, mot de passe hashé, rôle `ENUM`)
Database	Représente une base connectée, avec type, port, hôte, identifiants
Backup	Historique des sauvegardes (type, taille, statut, logs, erreurs, chemin fichier)
Restore	Trace les restaurations effectuées, liée à un `Backup`

Chaque table possède des champs :

- `created_at` et `updated_at` pour la traçabilité.
- Contraintes de clés étrangères entre `Database` → `User`, `Backup` → `Database`, etc.
- Types `ENUM` (`backup_status`, `restore_status`) assurant la validité des statuts.

📜 Création et alimentation des bases

- Tous les scripts sont organisés dans le répertoire `sql/`, avec un fichier par base.

Script	Contenu principal
`safebase.sql` (PostgreSQL)	Tables utilisateurs, connexions, backups, restores, ENUM, exemples
`postgres_db/init_db.sql`	Tables `albums`, `artistes`, `genres`
`mysql_db/online_library.sql`	Tables `livres`, `auteurs`, `catégories`

```
44      volumes:
45        - safebase_db_data:/var/lib/postgresql/data
46        - ./sql/safebase_db/safebase.sql:/docker-entrypoint-initdb.d/safebase.sql
```

Requêtes et manipulations

Des requêtes SQL sont exécutées pour :

- Ajouter une base MySQL/PostgreSQL après soumission d'un formulaire (INSERT).
- Lancer un backup et enregistrer le résultat dans la table `Backup`.
- Rechercher les sauvegardes par base ou par statut.
- Associer un `Restore` à un `Backup`.

Compétences mobilisées

- 🔧 Modélisation de base de données (MCD, MLD, MPD).

- 🧱 Création de tables relationnelles avec contraintes.
- 📋 Écriture de requêtes SQL (SELECT, INSERT, DELETE…).
- 📊 Utilisation de types ENUM PostgreSQL.
- 💾 Utilisation de DBeaver pour les contrôles manuels.
- 🐘 Travail avec PostgreSQL et MySQL en parallèle.
- 🐳 Déploiement automatique avec Docker (scripts init).

2.3 Bloc 3 : Backend API en Go

🎯 Objectif du bloc

Ce bloc de compétences vise à développer une **API REST performante et maintenable**, capable de gérer des processus métiers complexes (sauvegarde, restauration, historisation) dans un environnement sécurisé, modulaire et conteneurisé. Go a été choisi pour ses **hautes performances**, sa gestion **native de la concurrence** et sa **simplicité syntaxique**, parfaitement adaptée aux outils système comme `pg_dump` ou `mysqldump`.

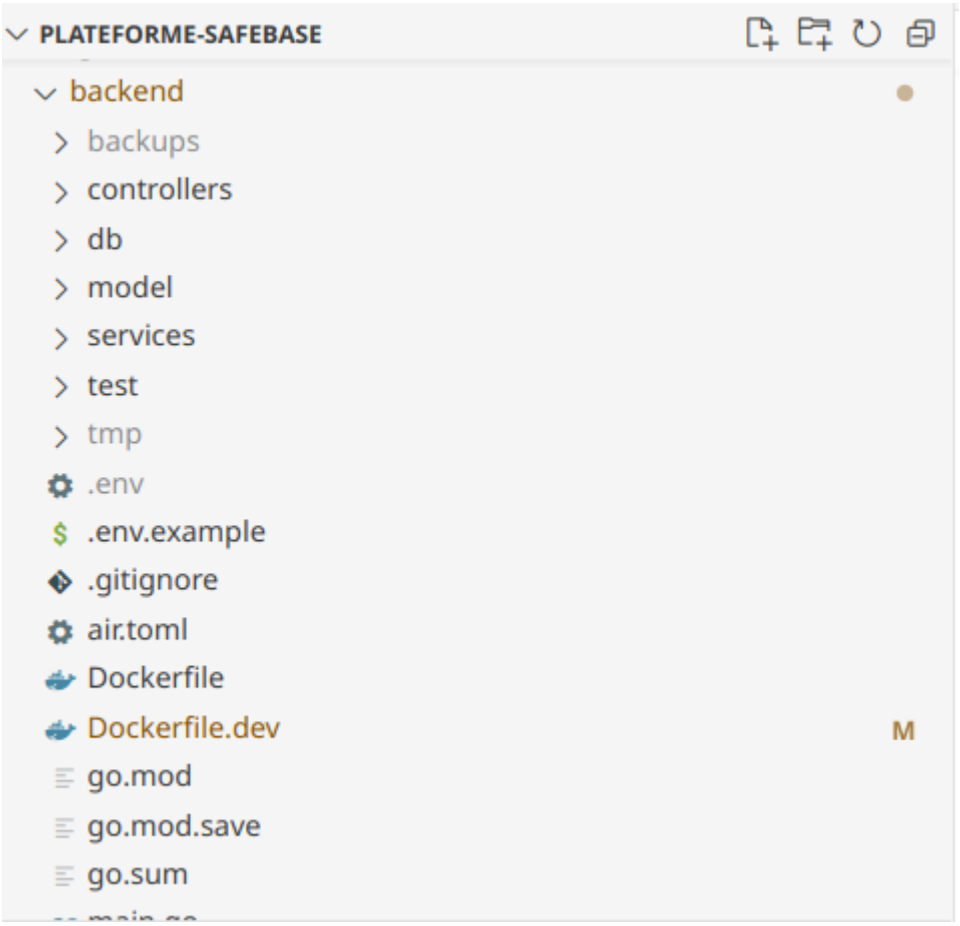
🧱 Architecture logicielle

L'API backend repose sur une architecture **modulaire** et **découpée par responsabilité** :

- **Handlers (Contrôleurs)** : réceptionnent les requêtes HTTP (ex. `POST /backup`, `GET /databases`).
- **Services** : contiennent la logique métier (déclenchement des sauvegardes/restaurations, test de connexion, etc.).

- **Repositories** : exécutent les requêtes SQL vers PostgreSQL (connexion en SQL natif).
- **Cron Jobs** : planifient des sauvegardes automatiques grâce à une librairie Go de scheduling (ex. `robfig/cron`).

arborescence du dossier backend



🛠️ Fonctionnalités développées

🔍 Gestion des bases de données

- Création, édition, suppression.
- Test de connexion à PostgreSQL ou MySQL.

- Validation et gestion d'erreurs avec logs.

💾 Sauvegarde des bases

- Lancement manuel d'un backup via une requête REST.
- Génération d'un dump via `pg_dump` (PostgreSQL) ou `mysqldump` (MySQL).
- Stockage dans le dossier `/backups` avec logs horodatés et statut enregistré en base.

voici extrait de `database_handler.go` montrant la gestion de POST `/database`

```go
func (s *BackupService) CreateBackup(
    databaseID string,
    status model.BackupStatus,
    backupType model.BackupType,
    filename string,
    size string,
    errorMsg string,
    log string,
) (*model.Backup, error) {

    dbID, err := uuid.Parse(databaseID)
    if err != nil {
        return nil, err
    }

    backup := &model.Backup{
        DatabaseID: dbID,
        Status:     status,
        BackupType: backupType,
        Filename:   filename,
        Size:       size,
        ErrorMsg:   errorMsg,
        Log:        log,
    }

    result := s.DB.Create(backup)
    if result.Error != nil {
        return nil, result.Error
    }

    return backup, nil
}
```

♻️ Restauration

- Restauration d'un dump existant sur une base sélectionnée.
- Gestion des erreurs de restauration.
- Journalisation de l'opération avec timestamp et statut.

📜 Historique & suivi

- Chaque opération de sauvegarde ou de restauration est enregistrée avec :
    - son type,

- son statut (`pending`, `success`, `failed`, etc.),
- son horodatage,
- et son log éventuel.

- Affichage via l'interface frontend pour un suivi clair.

🔐 Sécurité et configuration

- Variables d'environnement dans `.env` pour les accès à la base, ports, mots de passe (jamais codés en dur).
- Gestion des erreurs API avec messages détaillés (400, 500, etc.).

2.4 — Bloc 4 : Conteneurisation et Déploiement (Docker + CI/CD)

Le projet SafeBase a été conçu pour être portable, reproductible et déployable facilement dans tout environnement, grâce à la conteneurisation avec Docker et une stratégie de déploiement continu basée sur Docker Compose et GitHub Container Registry (GHCR).

🎯 Objectifs

- Portabilité : lancer l'application sur n'importe quelle machine compatible Docker.
- Reproductibilité : garantir le même environnement en développement et en production.
- Automatisation : simplifier le déploiement via CI/CD

🏗️ Structure des conteneurs

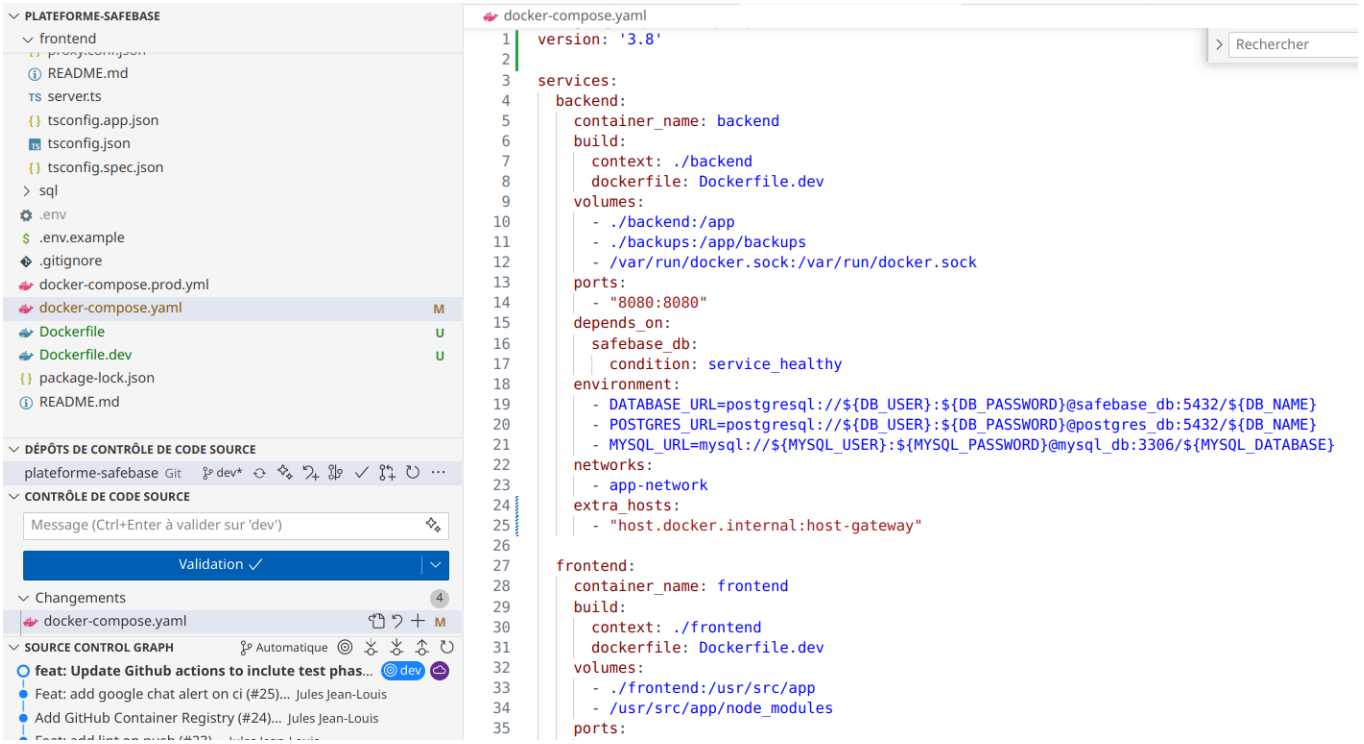
L'environnement se base sur plusieurs services définis dans deux fichiers :

- `docker-compose.yml` pour le développement

- `docker-compose.prod.yml` pour la production

Service	Description
`backend`	API REST en Go pour la gestion des sauvegardes/restaurations
`frontend`	Interface Angular servie avec Nginx
`safebase_db`	Base PostgreSQL contenant les configurations SafeBase
`postgres_db`	Base PostgreSQL externe à sauvegarder/restaurer
`mysql_db`	Base MySQL externe à sauvegarder/restaurer

Exemple de docker-compose.yaml



🐳 **Dockerfile (multi-étapes)**

🔧 **Dockerfile (production backend)**

```dockerfile
FROM golang:1.23.1 AS build

# On configure le répertoire de travail
WORKDIR /app

# On copie les fichiers de dépendances go mod et go sum pour profiter du cache docker
COPY go.mod go.sum ./

# Installation des dépendances
RUN go mod download

# On copie le reste du projet
COPY . .

# On construit l'exécutable Go avec des paramètres pour un binaire statique et optimisé pour
RUN CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -o /safebase-backend

# Étape finale : utilisation d'une image légère
FROM alpine:latest

# On ajoute le certificat CA pour éviter les erreurs SSL lors des appels réseau
RUN apk --no-cache add ca-certificates

# On définit l'utilisateur à non-root pour des raisons de sécurité (optionnel)
USER 1000:1000

# On copie l'exécutable construit dans l'étape précédente
COPY --from=build /safebase-backend /safebase-backend

# On expose le port 8080
EXPOSE 8080

# Commande de lancement de l'application
ENTRYPOINT ["/safebase-backend"]
```

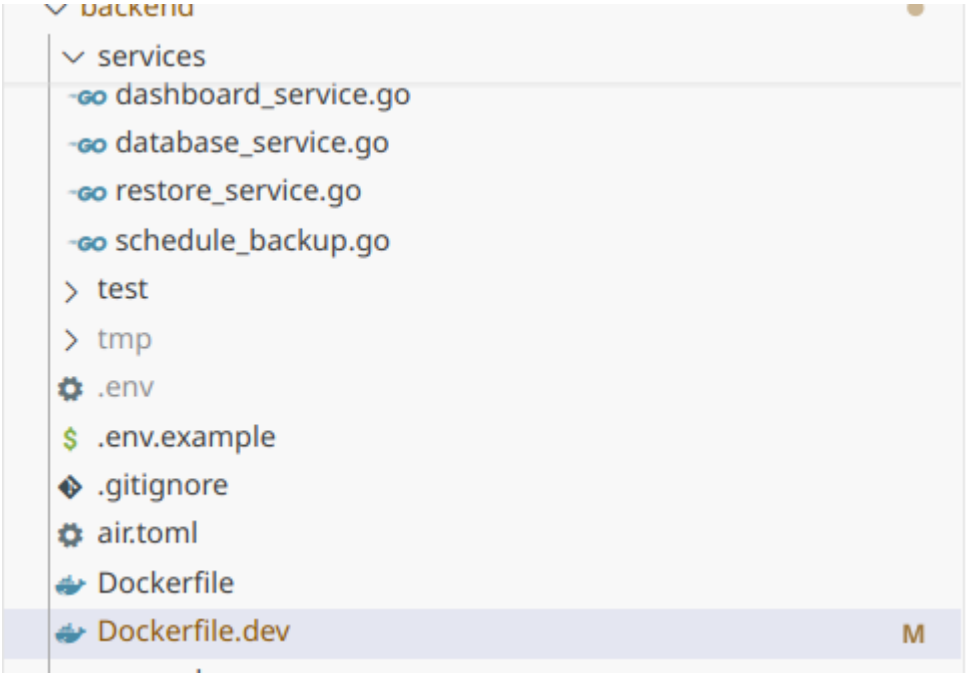
🔧 **Dockerfile.dev (développement backend)**

```
You, il y a 3 jours | 2 authors (Jules Jean-Louis and one other)
# Use Go image      Jules Jean-Louis, il y a 10 mois • feat: wip github action
FROM golang:1.24.0

# Install necessary packages including MySQL client and specific PostgreSQL client version
RUN apt-get update && \
    apt-get install -y wget gnupg default-mysql-client && \
    echo "deb http://apt.postgresql.org/pub/repos/apt bookworm-pgdg main" > /etc/apt/sources.list.d/pgdg.list
    wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add - && \
    apt-get update && \
    apt-get install -y postgresql-client-16 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Setup application
WORKDIR /app
RUN go install github.com/air-verse/air@latest
RUN go install github.com/golangci/golangci-lint/cmd/golangci-lint@latest
COPY go.mod go.sum ./
RUN go mod download
COPY . .
RUN go build -o main .

# Expose port and define entrypoint
EXPOSE 8080
CMD ["air"]
```



🧪 Initialisation locale

bash Copier Modifier

```
docker compose build
docker compose up
```

💡 En une commande, tous les services sont opérationnels.

🚀 Déploiement production avec `docker-compose.prod.yml`

bash Copier Modifier

```
docker-compose -f docker-compose.prod.yml up -d
```

✔️ Lancement des conteneurs en tâche de fond, avec configuration optimisée (Nginx, build Angular, production Go).

🔁 CI/CD avec GitHub Actions

```yaml
jobs:
  backend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 1
      - name: Setup Go
        uses: actions/setup-go@v5
        with:
          go-version: '1.23'
      - name: Install golangci-lint
        run: go install github.com/golangci/golangci-lint/cmd/golangci-lint@latest
      - name: Run golangci-lint
        run: golangci-lint run ./...
        working-directory: ./backend

```

2.4 – Bloc 4 : Conteneurisation et Déploiement (Docker + CI/CD)

Objectifs

- Rendre l'application **portable** et **facile à déployer**.
- Permettre une exécution par une **seule commande** `docker-compose`.
- Intégrer une stratégie DevOps via GitHub Actions et GHCR

Dans le cadre du projet **SafeBase**, l'approche DevOps a été mise en œuvre pour automatiser le processus de **build**, de **tests**, de **vérifications de code** et de **déploiement continu**. Cela repose principalement sur deux outils clés :

GitHub Actions : pour l'automatisation des workflows CI/CD

GitHub Container Registry (GHCR) : pour héberger les images Docker versionnées

🧩 Structure des conteneurs

Les services sont définis dans deux fichiers :

- `docker-compose.yaml` (développement)
- `docker-compose.prod.yml` (production)

capture docker-compose.yaml

docker-compose.yaml

You, il y a 3 jours | 3 authors (Jules Jean-Louis and others)

```yaml
version: '3.8'

services:
  backend:
    container_name: backend
    build:
      context: ./backend
      dockerfile: Dockerfile.dev
    volumes:
      - ./backend:/app
      - ./backups:/app/backups
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - "8080:8080"
    depends_on:
      safebase_db:
        condition: service_healthy
    environment:
      - DATABASE_URL=postgresql://${DB_USER}:${DB_PASSWORD}@safebase_db:5432/${DB_NAME}
      - POSTGRES_URL=postgresql://${DB_USER}:${DB_PASSWORD}@postgres_db:5432/${DB_NAME}
      - MYSQL_URL=mysql://${MYSQL_USER}:${MYSQL_PASSWORD}@mysql_db:3306/${MYSQL_DATABASE}
    networks:
      - app-network
    extra_hosts:
      - "host.docker.internal:host-gateway"
```

vue d'ensemble des conteneurs en cours d'exécution **(`docker ps`).**

```
boubacar@boubacar-Latitude-5480:~/Bureau/plateforme-safebase/backend$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS
   PORTS                                                NAMES
7b8d088abec4   plateforme-safebase-backend    "air"                    46 seconds ago   Up 32 seconds
   0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp          backend
1225d0a7b42c   plateforme-safebase-frontend   "docker-entrypoint.s…"   24 hours ago     Up 45 seconds
   0.0.0.0:4200->4200/tcp, [::]:4200->4200/tcp          frontend
33ffa74e8e7d   postgres:16                    "docker-entrypoint.s…"   3 days ago       Up 45 seconds (healthy)
   0.0.0.0:5434->5432/tcp, [::]:5434->5432/tcp          plateforme-safebase-safebase_db-1
0be546ec1e94   postgres:16                    "docker-entrypoint.s…"   3 days ago       Up 45 seconds
   0.0.0.0:5435->5432/tcp, [::]:5435->5432/tcp          plateforme-safebase-postgres_db-1
eaf0b1b8ac1e   mysql                          "docker-entrypoint.s…"   3 days ago       Up 45 seconds
   33060/tcp, 0.0.0.0:3307->3306/tcp, [::]:3307->3306/tcp   plateforme-safebase-mysql_db-1
boubacar@boubacar-Latitude-5480:~/Bureau/plateforme-safebase/backend$
```

Dockerfile personnalisé

🔧 **Développement : `Dockerfile.dev`**

Utilisé avec hot-reload (`air`) :

Dockerfile.dev

```
# Dockerfile.dev

# Utiliser une version récente de Go
FROM golang:1.24

# Définir le dossier de travail
WORKDIR /app

# Copier les fichiers go.mod et go.sum d'abord pour optimiser le cache
COPY go.mod ./
COPY go.sum ./
RUN go mod download

# Copier tout le code source
COPY . .

# Installer Air (reloader Go)
RUN go install github.com/air-verse/air@latest

# Commande par défaut
CMD ["air"]
```

```
backend  |
backend  |   __    _   ___
backend  |  / /\  | | | |_)
backend  | /_/--\ |_| |_| \_ v1.62.0, built with Go go1.24.0
backend  |
backend  | watching .
backend  | watching backups
```

Frontend Angular + Nginx

la commande
ng build --configuration production

Déploiement en production

docker-compose -f docker-compose.prod.yml up -d

CI/CD avec GitHub Container Registry

git add .

git commit -m "CI/CD test"

git push origin dev

tests automatisés Backend en Go

Afin d'assurer la fiabilité des fonctionnalités critiques de l'API SafeBase, des tests automatisés ont été mis en place en Go à l'aide du package standard `testing`. Ces tests couvrent notamment les opérations de création, lecture et suppression de bases de données, en s'appuyant sur les services métiers du backend.

🎯 Objectif du test

Vérifier le fonctionnement complet du service de gestion de base de données :

- ✅ Connexion à PostgreSQL
- ✅ Insertion d'une base de données dans la table `database`
- ✅ Récupération de l'entrée par nom
- ✅ Comparaison des données
- ✅ Suppression de l'entrée pour nettoyage

Maquettage de l'interface utilisateur

Pour garantir une expérience utilisateur fluide et intuitive, le projet *Finance Flow* a été précédé d'une phase de maquettage. Ces maquettes permettent de visualiser les principales interfaces de l'application avant le développement et assurent une cohérence graphique tout au long du parcours utilisateur.

Objectifs du maquettage

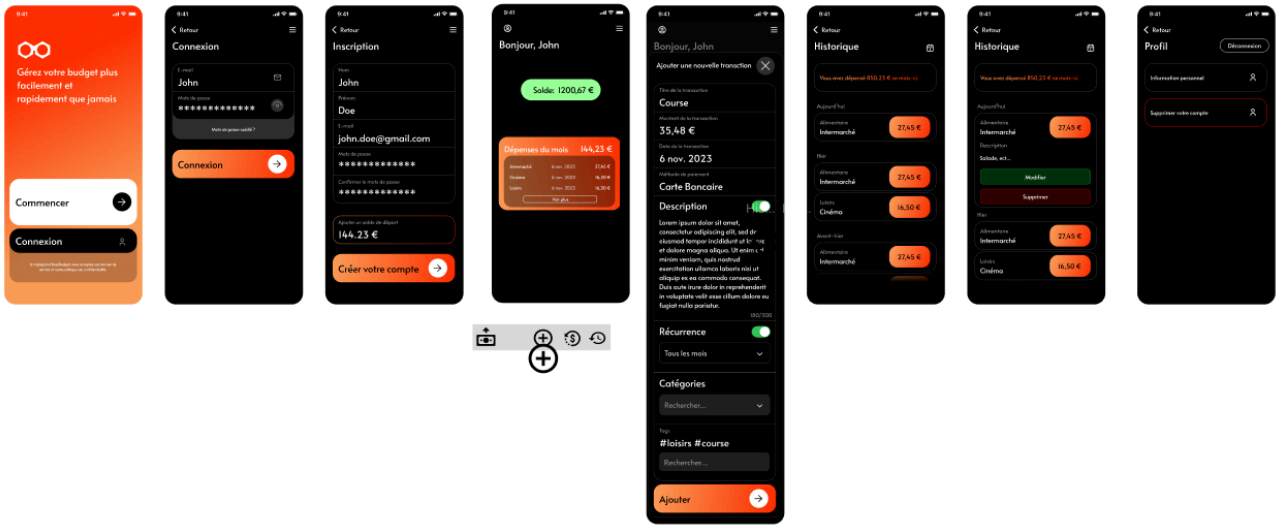
Les maquettes ont pour objectif de :

- Représenter visuellement les principales fonctionnalités de l'application ;
- Valider l'ergonomie et la navigation entre les écrans ;

- Servir de guide pour l'intégration front-end (React) ;
- Assurer l'harmonie visuelle et la clarté de l'information.

Outils utilisés

Le maquettage a été réalisé à l'aide de Figma, un outil collaboratif en ligne permettant de créer des prototypes interactifs et adaptables aux différents formats (desktop et mobile).



2.5 Compétences mobilisées

Le projet **SafeBase** m'a permis de mobiliser un large éventail de compétences techniques et professionnelles, réparties sur les 4 blocs de compétences du référentiel RNCP24449. Voici un tableau synthétique des compétences mises en œuvre.

🧩 ception et développement d'une solution applicative

Code	Compétence mise en œuvre
	Concevoir une solution technique applicative à partir de spécifications

fonctionnelles

Maquetter une application

Développer des interfaces utilisateur (Angular)

Développer des composants métiers (API Go)

Construire une base de données relationnelle (PostgreSQL/MySQL)

Préparer et exécuter les plans de tests (unitaires, fonctionnels)

Conception et gestion de bases de données

Compétence mise en œuvre

Modéliser les données (MCD, MLD)

Concevoir et implémenter des bases de données SQL

Implémenter des procédures de sauvegarde/restauration

Optimiser les requêtes d'accès aux données

Sécuriser l'accès aux données (gestion des utilisateurs, rôles, hashage des mots de passe)

🛠️ **Développement d'une API sécurisée**

Compétence mise en œuvre

Concevoir et développer une API REST sécurisée

Implémenter des services de gestion des erreurs et des statuts

Gérer des fichiers (upload, téléchargement, sauvegardes .sql)

Documenter les endpoints et les flux d'utilisation

Intégrer des tâches planifiées (cron jobs)

🚀 **Déploiement, conteneurisation et DevOps**

Compétence mise en œuvre

Écrire des Dockerfile pour backend et frontend

Configurer un fichier docker-compose pour multi-services

Déployer des conteneurs sur un environnement local ou distant

Versionner des images sur GitHub Container Registry

Créer une stratégie de déploiement continue avec GitHub Actions (prévu)

test Unitaires

Maquettage

Activité-type 3 Conclusion et perspectives d'évolution

Conclusion

Le projet **SafeBase** a constitué une expérience complète et formatrice, mobilisant des compétences techniques variées dans un cadre proche d'un environnement professionnel réel. À travers la mise en œuvre d'un **backend en Go**, d'un **frontend en Angular 18**, de **bases de données PostgreSQL et MySQL**, et d'un **système de conteneurisation via Docker**, j'ai pu réaliser une application cohérente, stable et évolutive, répondant à un besoin concret : la gestion sécurisée des sauvegardes et restaurations de bases de données.

Ce projet m'a permis de :

- Concevoir et développer une **architecture logicielle modulaire et robuste**.
- Travailler avec des outils professionnels comme **pg_dump**, **mysqldump**, **Docker**, **PrimeNG**, **GitHub Actions**, etc.
- Développer une **API REST sécurisée**, documentée, avec une bonne gestion des erreurs.
- Gérer l'intégration de **tâches planifiées (cron)** pour automatiser les sauvegardes.
- Maîtriser la **chaîne de déploiement en conteneur**, aussi bien pour le développement que la production.

Perspectives d'évolution

Plusieurs axes d'amélioration sont envisageables pour faire évoluer SafeBase vers un produit encore plus complet et professionnel :

1. **Gestion des utilisateurs avancée** : ajouter une interface d'administration, des rôles personnalisés et un système d'authentification JWT pour sécuriser l'accès.

2. **Système de notification** : notifier les utilisateurs en cas de succès/échec d'un backup ou d'une restauration (email, interface web, logs enrichis).

3. **Sauvegarde sur le cloud** : permettre de stocker les fichiers `.sql` sur des services comme AWS S3, Google Cloud Storage ou Nextcloud.

4. **Interface mobile** : proposer une version mobile ou responsive pour gérer les sauvegardes/restaurations depuis un smartphone.

5. **Monitoring et alertes** : intégrer des outils comme Grafana ou Prometheus pour suivre l'état des bases de données et des opérations.

6. **Amélioration de la sécurité** : chiffrement des fichiers de backup, audit de sécurité de l'API, protection contre les injections SQL.

Titres, diplômes, CQP, attestations de formation

(facultatif)

Intitulé	Autorité ou organisme	Date
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.

Déclaration sur l'honneur

Je soussigné(e) [prénom et nom] BOUBACAR OUATTARA ,

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 29 Juillet 2025

pour faire valoir ce que de droit.

Signature :

Ouattara Boubacar

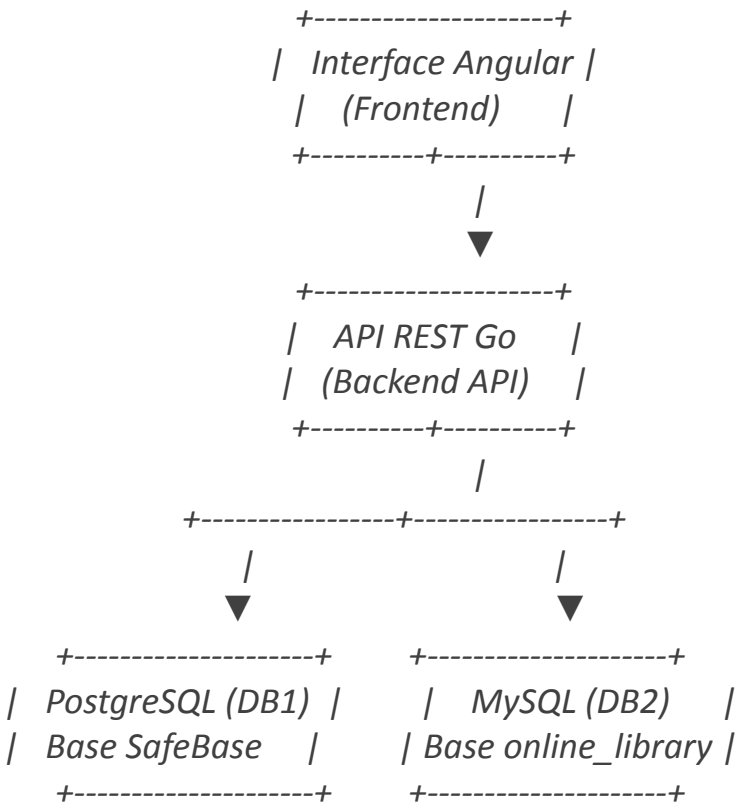
Documents illustrant la pratique professionnelle

(facultatif)

Intitulé
Cliquez ici pour taper du texte.

ANNEXES

🌐 *Architecture globale de l'application*



Infrastructure déployée avec Docker Compose (multi-conteneurs)

⚙️ ***Fichier `docker-compose.yaml` (extrait)***

docker-compose.yaml

You, il y a 4 jours | 3 authors (Jules Jean-Louis and others)

```yaml
version: '3.8'

services:
  backend:
    container_name: backend
    build:
      context: ./backend
      dockerfile: Dockerfile.dev
    volumes:
      - ./backend:/app
      - ./backups:/app/backups
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - "8080:8080"
    depends_on:
      safebase_db:
        condition: service_healthy
    environment:
      - DATABASE_URL=postgresql://${DB_USER}:${DB_PASSWORD}@safebase_db:5432/${DB_NAME}
      - POSTGRES_URL=postgresql://${DB_USER}:${DB_PASSWORD}@postgres_db:5432/${DB_NAME}
      - MYSQL_URL=mysql://${MYSQL_USER}:${MYSQL_PASSWORD}@mysql_db:3306/${MYSQL_DATABASE}
    networks:
      - app-network
    extra_hosts:
      - "host.docker.internal:host-gateway"
```

🛢️ Script SQL de création de la base SafeBase

```sql
CREATE TYPE user_role AS ENUM ('admin', 'user');
CREATE TABLE "user" (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  email VARCHAR NOT NULL UNIQUE,
  password VARCHAR NOT NULL,
  role user_role NOT NULL DEFAULT 'user'
);

CREATE TABLE "database" (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name VARCHAR NOT NULL,
  type VARCHAR NOT NULL,
  host VARCHAR NOT NULL
);
```

🔁 Exemple de tâche cron pour backups automatiques

```go
cron := cron.New()
cron.AddFunc("@every 24h", func() {
  service.ExecuteBackup()
})
cron.Start()
```

🔒 Exemple de test de connexion à une base

GET
/test-connection?host=postgres_db&port=5432&username=postgres&password=xxx&dbName=safebase

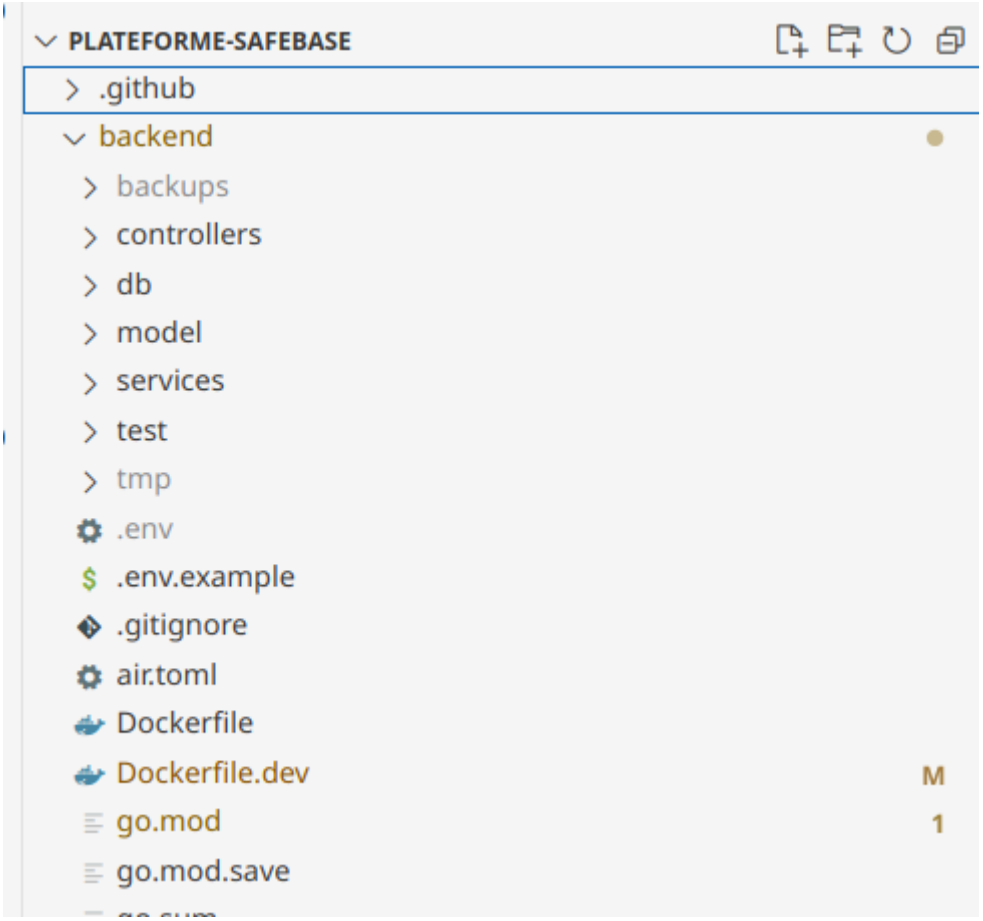
🧪 Exemple de test Postman

POST `/database`

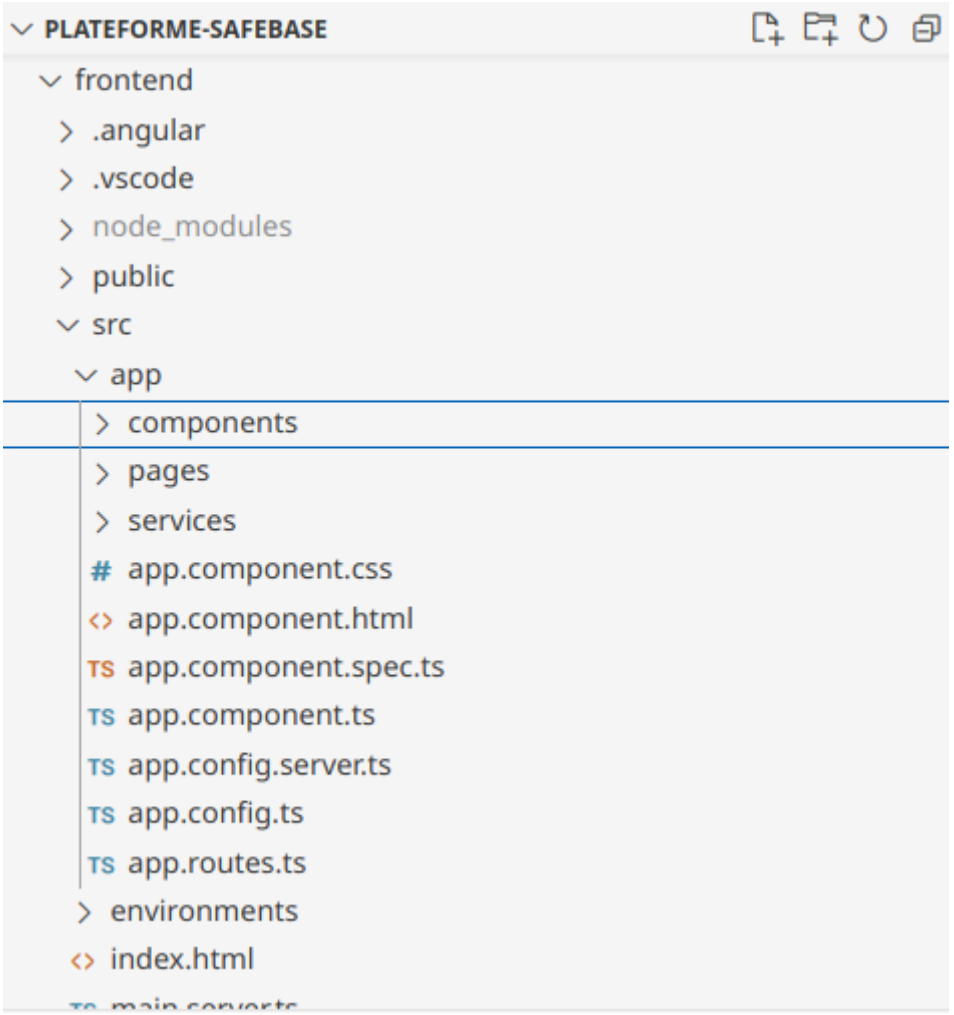
```json
{
  "name": "online_library",
  "type": "mysql",
  "host": "mysql_db",
  "port": "3306",
  "username": "root",
  "password": "password",
  "database_name": "online_library"
}
```

📦 Structure des dossiers backend/frontend

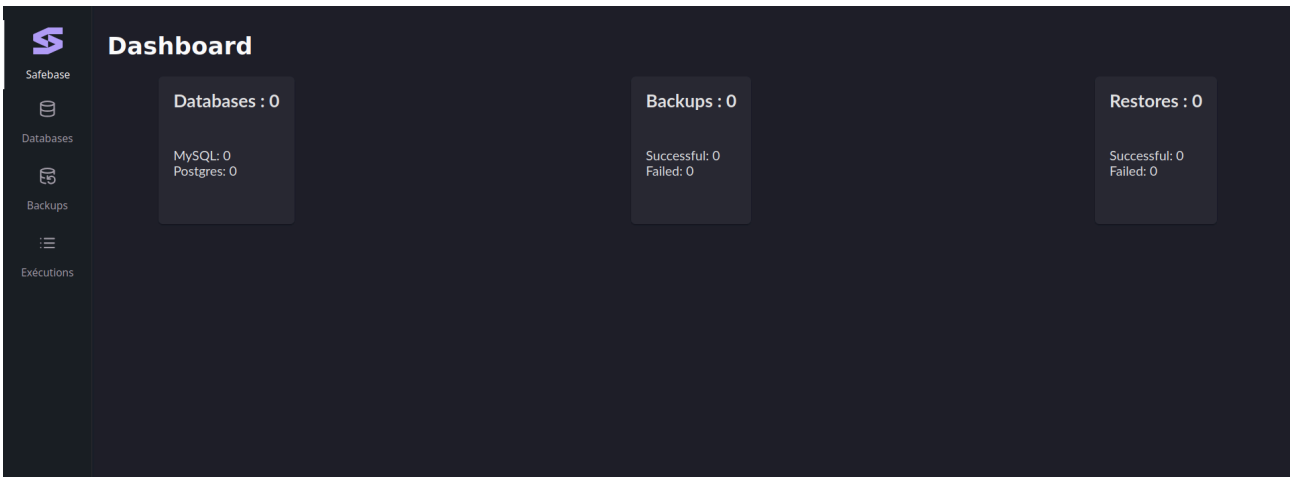
backend



front-end



Interface d'ajout de base de données.



DOSSIER PROFESSIONNEL (DP)