Bachelor 3 CDA (Concepteur Développeur d'Applications)

Année académique : 2024/2025
Dossier professionnel

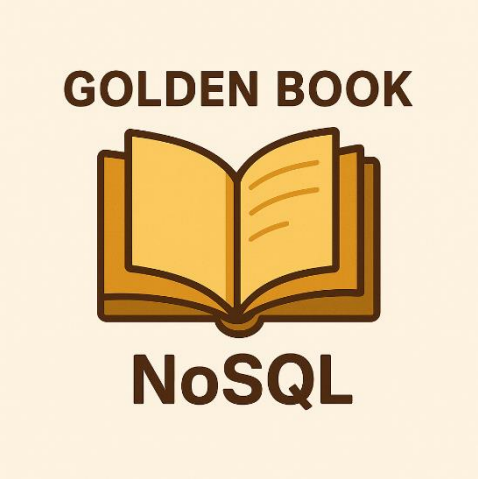
GOLDENBOOK

Laissez un mot gardez un souvenir

+

SAFEBASE

Plateforme de gestion de BDD



Réalisé par :

Adam Younsi

adam.younsi@laplateforme.io

Remerciement

Ce dossier marque l'aboutissement d'un parcours exigeant, mais profondément formateur, dans le cadre de ma formation en Bachelor 3 – Concepteur Développeur d'Applications.

Il représente non seulement la synthèse de compétences techniques acquises au fil du temps, mais aussi l'illustration concrète de ma capacité à mener un projet de bout en bout, avec rigueur, autonomie et persévérance.

Je tiens à remercier chaleureusement mes formateurs, mes camarades de promotion, ainsi que les intervenants professionnels qui ont su me guider, me conseiller et parfois me challenger avec exigence.

Leurs retours constructifs ont fortement contribué à faire évoluer ces projet et m'ont permis d'atteindre un niveau de maturité technique et organisationnelle que je n'aurais pas imaginé en début de parcours.

Ces projet a également été une aventure personnelle : ils m'ont permis de découvrir mes limites, de les repousser, mais surtout de confirmer mon appétence pour l'ingénierie logicielle, la conception d'applications et les problématiques concrètes liées au développement logiciel moderne.

Introduction

Le projet **Golden Book** est un livre d'or numérique développé dans le cadre de la formation Bachelor 3 – Développement Web. Il s'agit d'une application web simple mais fonctionnelle, permettant aux utilisateurs de laisser un message accompagné de leur prénom. Ces messages sont ensuite stockés de manière persistante dans une base de données **MongoDB**, selon une architecture **Node.js + Express + Mongoose**.

Ce projet a été pensé pour explorer en profondeur les **bases de données NoSQL**, ainsi que la structuration d'une API REST. En plus de renforcer la pratique du backend JavaScript, il met l'accent sur la **gestion des données dynamiques**, la **modularité du code**, et l'usage sécurisé des **variables d'environnement**.

Liste des compétences visées

Ce projet vise à mobiliser les compétences suivantes :

Compétence	Mise en œuvre dans Golden Book
Concevoir une API REST	Routes GET et POST sécurisées sous Express
Gérer une base NoSQL	Modélisation et requêtage avec MongoDB + Mongoose
Utiliser des variables d'environnement	Configuration sécurisée via dotenv
Structurer un projet backend	Architecture modulaire Node.js
Gérer les échanges client-serveur	Utilisation de fetch API + Express
Implémenter un serveur minimaliste	Initialisation, écoute, parsing JSON, CORS
Organiser un projet front statique	HTML + JS simple et responsive
Documenter un projet technique	README, commentaires, structure de fichiers

Technologies principales utilisées

Couche	Technologie	Rôle principal
Backend	Node.js + Express	Serveur HTTP + gestion des routes
BDD	MongoDB	Stockage des messages
ORM / ODM	Mongoose	Modélisation + requêtage Mongo
Frontend	HTML5 + JavaScript	Interface simple utilisateur
Environnement	dotenv	Sécurisation des identifiants de connexion
Sécurité	.gitignore + dotenv	Exclusion des données sensibles Git

Cahier des charges

Objectif du projet

L'objectif du projet est de créer un **livre d'or numérique** dans lequel les visiteurs peuvent **laisser un message** accompagné de leur prénom. Ces messages sont ensuite stockés dans une base MongoDB et affichés en temps réel sur la page d'accueil.

Ce projet met en avant :

- la mise en œuvre d'un **backend minimaliste**,
- la gestion de **données dynamiques avec MongoDB**,
- l'intégration simple d'un frontend statique (HTML + JavaScript).

Utilisateur cible

L'application est pensée pour être utilisée dans des **événements publics ou privés**, comme :

- des salons professionnels
- des cérémonies d'entreprise
- des anniversaires, mariages ou expositions

L'utilisateur est **anonyme** : aucune inscription, ni mot de passe n'est nécessaire.

Fonctionnalités principales

- **Ajout de message** : saisie du prénom + message texte
- **Envoi via POST** : formulaire JSON envoyé à /messages
- **Stockage automatique** : insertion dans MongoDB avec createdAt
- **Affichage des messages** : tri du plus récent au plus ancien
- **Design simple et responsive**

Contraintes techniques et fonctionnelles

Type	Détail
Technique	Doit fonctionner avec un serveur Node.js + MongoDB local ou distant
Sécurité	Pas de mot de passe → éviter les injections ou XSS
Données	Stockage en NoSQL, format BSON/JSON
Déploiement	Compatible avec Docker ou hébergement simple (Render, Railway...)
Accès front	Interface statique accessible sur le port 3000
Évolutivité	Base conçue pour accueillir plusieurs centaines de messages

Méthodologie adoptée

Le projet a été réalisé selon une approche **itérative et incrémentale** :

- Initialisation de l'environnement Node.js
- Mise en place de la base MongoDB locale
- Définition du modèle avec Mongoose
- Développement des routes API (GET/POST)
- Intégration du frontend HTML/JS
- Tests manuels via le navigateur
- Sécurisation via dotenv et .gitignore
- Documentation et finalisation

Architecture logicielle

L'architecture logicielle du projet **Golden Book** repose sur une organisation simple, modulaire et facilement maintenable. Elle suit un modèle de type **client-serveur** avec une API REST et une base de données NoSQL MongoDB.

Le code est divisé en deux grandes parties :

- Un **frontend** statique : interface HTML/JS permettant la saisie et l'affichage des messages
- Un **backend Node.js/Express** : serveur REST traitant les requêtes et interagissant avec MongoDB

Cette organisation suit les principes suivants :

- **Responsabilité unique** : chaque fichier a un rôle clair
- **Séparation des couches** : présentation, logique métier, persistance
- **Communication JSON** : cohérent avec les APIs modernes

Arborescence du projet

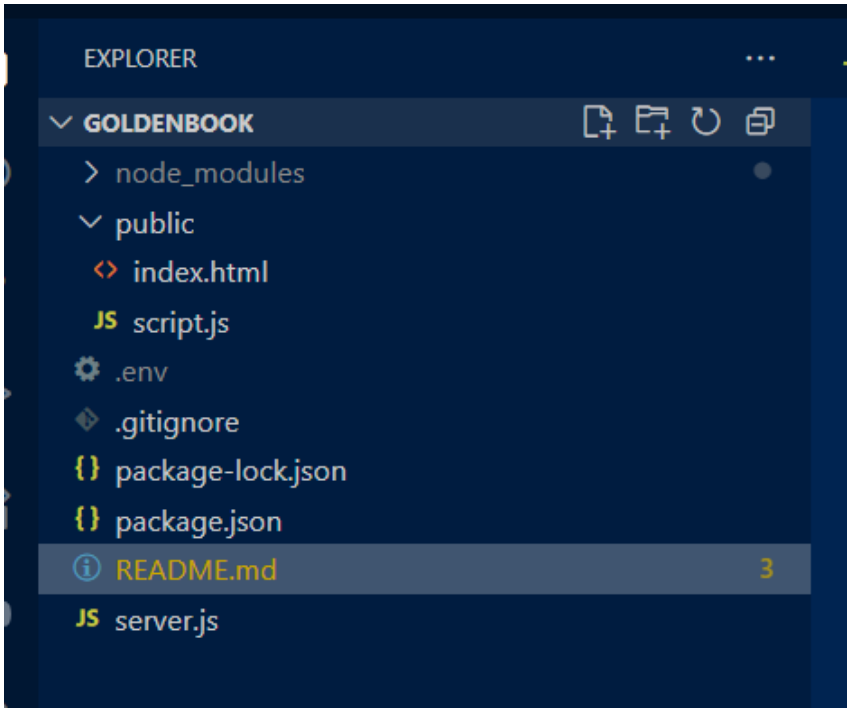


Figure 1 : Architecture GoldenBook

Développement backend

Le développement backend repose sur **Node.js** et le framework **Express.js**, qui forment un écosystème léger, performant et très adapté aux API REST.

Technologie principale

Qu'est-ce que Node.js ?

Node.js est un environnement d'exécution JavaScript côté serveur. Il permet d'écrire des applications réactives, non bloquantes, et rapides. Il repose sur le moteur V8 de Google Chrome.

Caractéristiques principales :

- Exécution asynchrone (via callbacks, promesses, async/await)
- Monothread mais très performant en I/O
- Utilisé massivement dans les applications web modernes

Dans Golden Book, Node.js gère :

- L'initialisation du serveur
- Les routes GET et POST
- La communication avec la base MongoDB

Qu'est-ce qu'Express.js ?

Express est un framework minimaliste pour Node.js. Il permet de créer facilement des routes HTTP et de gérer les requêtes entrantes/sortantes.

Fonctionnalités apportées :

- Routing GET/POST
- Middleware JSON (body-parser intégré)
- Gestion du CORS
- Modularité et syntaxe très concise

Express a été choisi car il est idéal pour les **projets de petite et moyenne taille** qui n'ont pas besoin d'un framework lourd.

Pourquoi ce choix technologique ?

1. **Légèreté** : le serveur tient dans un seul fichier
2. **Rapide à mettre en place** : Express se configure en quelques lignes
3. **Très populaire** : vaste communauté, documentation abondante
4. **Compatible avec MongoDB** : via la librairie Mongoose
5. **Aligné avec la stack JS** : même langage côté client et serveur

Exemple d'implémentation backend

Figure 2 : Implémentation Backend

Bilan

Le backend de Golden Book incarne la simplicité et l'efficacité de la stack JavaScript moderne. Node.js et Express forment un duo minimaliste mais puissant, parfait pour créer une API REST connectée à une base MongoDB. Leur adoption garantit une très bonne compatibilité avec les outils modernes, une facilité de prise en main, et une grande flexibilité pour faire évoluer le projet.

Développement frontend

L'interface utilisateur de l'application Golden Book repose sur une architecture **statique et légère** composée de deux fichiers principaux : un fichier HTML (index.html) et un script JavaScript (script.js).

Ce choix volontairement minimaliste vise à se concentrer sur l'interaction directe avec l'API REST sans passer par un framework lourd (React, Vue ou Angular). Ce type d'approche est parfaitement adapté pour des applications simples ou des prototypes fonctionnels.

Technologie et structure

HTML5 : structure de la page

Le fichier index.html constitue la base de la structure visuelle. Il contient :

- Un **formulaire** HTML classique avec un champ texte (nom), une zone de texte (message), et un bouton de validation
- Une section #messages qui sera remplie dynamiquement avec les messages récupérés depuis la base de données
- Un lien vers script.js en bas de page

JavaScript : interaction avec l'API

Le fichier script.js gère toute la logique d'interaction frontend/backend :

- Récupération des données du formulaire
- Appel fetch en POST vers /messages
- Appel fetch en GET vers /messages pour afficher les messages
- Manipulation du DOM pour afficher les messages

CSS : style basique

Le style est intégré directement dans index.html via des balises <style>. Il pourrait être externalisé dans un fichier CSS séparé pour un projet plus complexe.

Figure : Interface utilisateur

Structure des composants

Bien que le projet ne repose pas sur un framework composants, nous pouvons considérer les éléments suivants comme des "composants logiques" :

Formulaire d'envoi (HTML)

```
7   <h1>Laissez un message</h1>
8   <input type="text" id="name" placeholder="Votre prénom" />
9   <br />
10  <textarea id="message" placeholder="Votre message..."></textarea>
11  <br />
12  <button onclick="sendMessage()">Envoyer</button>
13  <hr />
14  <h2>Messages</h2>
15  <div id="messages"></div>
```

Figure 3 : Formulaire HTML

Zone d'affichage des messages

```
10    <textarea id="message" placeholder="Votre message..."></textarea>
```

Le JavaScript va remplir dynamiquement cette zone avec des div contenant les données de la base.

Script principal (JS)

public > JS script.js > ...

```
async function sendMessage() {
  const name = document.getElementById("name").value;
  const message = document.getElementById("message").value;

  await fetch("/messages", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ name, message }),
  });

  loadMessages();
}

async function loadMessages() {
  const res = await fetch("/messages");
  const messages = await res.json();

  const container = document.getElementById("messages");
  container.innerHTML = messages
    .map((m) => `<p><strong>${m.name}</strong>: ${m.message}</p>`)
    .join("");
}

loadMessages();
```

Figure 4 : Script.js

Appel API

Le frontend communique avec le backend grâce à l'objet **fetch()** de l'API Web JavaScript. Il s'agit d'un outil moderne permettant de réaliser des requêtes HTTP de manière asynchrone.

Requête POST (ajouter un message)

Déjà illustré dans le formulaire ci-dessus. L'utilisateur remplit les champs, et l'objet JSON est envoyé au serveur Express, qui le sauvegarde via Mongoose.

Bilan frontend

- Structure très simple, mais efficace
- Facilement adaptable à un framework JS si évolution du projet
- Appels API parfaitement isolés via fetch()
- Affichage dynamique en DOM pur, sans bibliothèque externe

Ce type de frontend est adapté aux projets légers, démos internes, ou interfaces événementiell

Base de données

La base de données est un élément central du projet **Golden Book**, car elle permet le **stockage persistant** des messages laissés par les utilisateurs. Contrairement aux bases traditionnelles basées sur SQL, ce projet utilise une base **NoSQL**, et plus précisément **MongoDB**.

Qu'est-ce qu'une base NoSQL ?

Le terme **NoSQL** signifie "Not Only SQL". Il désigne une famille de bases de données qui ne reposent **pas sur un modèle relationnel** (avec des tables, lignes et colonnes), mais sur des **modèles flexibles** comme :

- les **documents** (ex. MongoDB)
- les **clés-valeurs** (Redis)
- les **graphiques** (Neo4j)
- les **colonnes** (Cassandra)

Les bases NoSQL ont été conçues pour répondre aux besoins du web moderne :

- Grands volumes de données non structurées
- Hautes performances en lecture/écriture
- Évolution flexible du schéma

Différences entre SQL et NoSQL

Critère	SQL (relationnel)	NoSQL (documentaire - MongoDB)
Structure	Tables avec schéma fixe	Documents JSON flexibles
Langage de requête	SQL (SELECT, JOIN, etc.)	API JS-like (find, insert, update)
Schéma	Défini à l'avance	Dynamique et adaptatif
Performance	Excellente pour les jointures complexes	Optimisée pour les opérations simples
Cas d'usage	ERP, compta, gestion détaillée	Web apps, IoT, Big Data

MongoDB : base NoSQL documentaire

MongoDB est la base de données NoSQL la plus populaire. Elle stocke les données sous forme de **documents BSON** (une version binaire de JSON).

Dans ce projet, chaque message est stocké comme un document :

Figure 5 : Exemple de stockage

MongoDB remplace les tables par des **collections** (ici : messages). Chaque document peut avoir sa propre structure (champ optionnel, etc.).

Connexion à la base MongoDB

Pour se connecter à la base MongoDB, le projet utilise la bibliothèque **Mongoose**, qui est un **ODM** (Object Document Mapper).

Exemple de connexion dans server.js :

Figure 6 : Server.js connexion à la BDD

Cela permet de **protéger les informations sensibles** et d'éviter de les diffuser sur GitHub (le .env est dans le .gitignore).

Mongoose : modélisation des données

```
const Message = mongoose.model("Message", {
  name: String,
  message: String,
  createdAt: { type: Date, default: Date.now }
});
```

Cela permet de définir un **modèle de données** avec validation automatique. Mongoose simplifie également les opérations comme :

- Message.find()
- new Message({ name, message }).save()

MongoDB Compass : outil visuel

MongoDB Compass est une interface graphique officielle pour visualiser et manipuler les données stockées dans MongoDB.

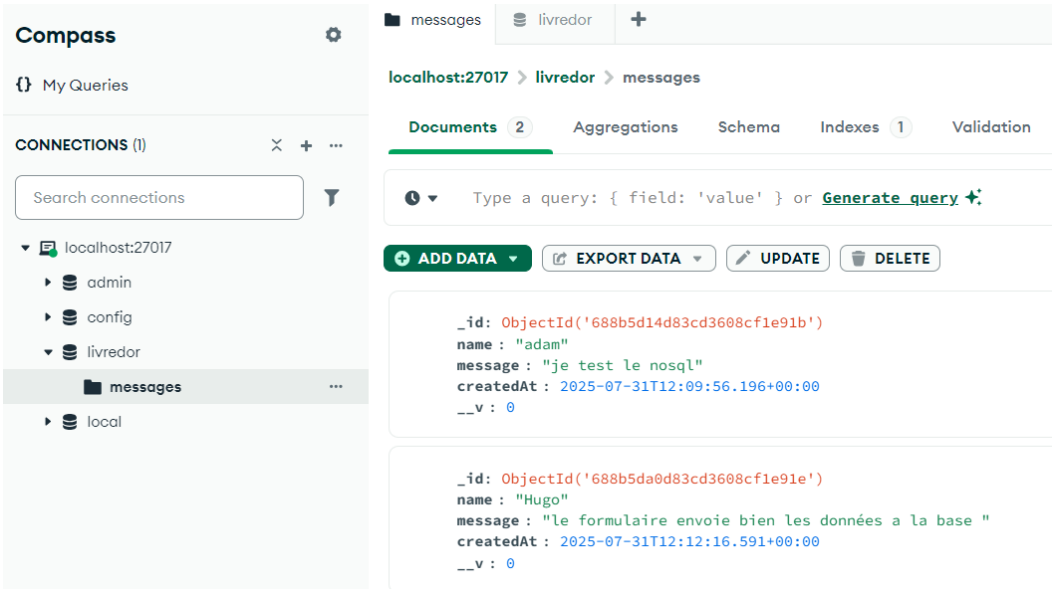


Figure 7 : Interface Compass

Il permet :

- de voir les collections et documents en temps réel
- de tester des requêtes visuellement
- de filtrer, trier, modifier les documents
- d'inspecter la structure JSON sans ligne de commande

Dans ce projet, Compass a permis de vérifier rapidement que les messages étaient bien insérés et triés correctement par date.

Bilan

L'utilisation de **MongoDB** avec **Mongoose** dans ce projet présente de nombreux avantages :

- Flexibilité du schéma et adaptation rapide
- Rapidité de développement (modèles simples)
- Intégration fluide avec Node.js
- Visualisation aisée avec Compass

Le choix d'une base **NoSQL** est ici pleinement justifié par la simplicité du projet, l'absence de relations complexes, et le besoin de stocker des données textuelles dynamiques.

Veille technologique

Tout au long du développement du projet Golden Book, une veille technologique active a été menée afin de comprendre, choisir et intégrer les bonnes technologies dans le bon contexte. Cette veille a permis de monter en compétence sur des outils spécifiques à l'écosystème Node.js/NoSQL.

Sujets de veille

Bases NoSQL

- Comprendre le modèle document (vs relationnel)
- Avantages de MongoDB pour les petites apps
- Bonnes pratiques de modélisation avec Mongoose

Architecture backend Node.js

- Mise en place d'un serveur minimaliste avec Express
- Routing, middleware, gestion des erreurs
- Utilisation de dotenv pour sécuriser la configuration

Connexion à MongoDB + outils graphiques

- Utilisation de MongoDB Compass pour l'exploration visuelle des collections
- Interrogation et filtrage de documents via Compass ou en ligne de commande

Tests & débogage

- Analyse des retours JSON et logs serveurs

Sources principales consultées

Ressource	Sujet abordé
mongodb.com/docs	Commandes Mongo, modélisation NoSQL
expressjs.com	Création de routes et serveurs REST

StackOverflow	Erreurs d’intégration, best practices
Medium, Dev.to	Retours d’expérience Node/Mongo
YouTube	Tutoriels Express, MongoDB Compass
GitHub	Projets open source similaires

Enrichissement personnel

Cette veille m’a permis de :

- Renforcer ma compréhension des **bases NoSQL**
- Mieux maîtriser **Node.js** et les API REST
- Comprendre l’intérêt des outils comme **Compass**
- Développer des réflexes de recherche autonome (lecture de docs officielles)

Synthèse / Conclusion

Le projet **Golden Book** a permis de réaliser une application web fonctionnelle, sécurisée et modulable autour de l'architecture Node.js + MongoDB. Le choix de la base NoSQL a été au centre de la réflexion technique et a révélé son intérêt pour des données peu structurées mais dynamiques.

L'utilisation de technologies modernes (dotenv, Mongoose, Compass, Express) a permis de créer un projet cohérent, en lien direct avec les compétences attendues du Bachelor.

Satisfaction personnelle : Ce projet m'a permis d'aller au bout d'un cycle complet de développement backend/frontend. Il m'a donné les bases solides pour poursuivre sur des projets full-stack plus complexes.

Ce projet pourra être amélioré dans le futur (pagination, upload, interface réactive) et potentiellement déployé sur des plateformes cloud. Il constitue une référence concrète dans mon parcours de développeur.

Introduction

La gestion des bases de données est un enjeu critique pour toute organisation, qu'il s'agisse d'applications web, de systèmes d'information ou d'outils internes. La moindre perte de données peut engendrer des conséquences majeures, tant sur le plan opérationnel que financier. Dans ce contexte, disposer d'un outil fiable de sauvegarde et de restauration est essentiel.

SafeBase est une application web conçue pour répondre à ce besoin. Elle permet d'effectuer **des sauvegardes complètes** et **des restaurations contrôlées** de bases de données **MySQL** et **PostgreSQL**, deux des moteurs les plus utilisés dans le monde. Accessible via une interface web, l'outil permet à un utilisateur autorisé de :

- Lancer la sauvegarde d'une base de données vers un fichier .sql
- Restaurer une base de données depuis un fichier de dump
- Télécharger ou stocker les fichiers de dump produits
- Visualiser et suivre les opérations réalisées

SafeBase vise à faciliter ces opérations techniques sans avoir à passer par une ligne de commande, tout en assurant un haut niveau de **contrôle, de sécurité et d'automatisation**. Son développement met en œuvre des technologies open-source modernes (PHP, Docker, MySQL, PostgreSQL) et démontre la capacité à construire une solution utile, portable, et professionnelle.

Cette documentation technique présente l'ensemble du processus de conception et de réalisation de SafeBase, en détaillant les choix technologiques, l'architecture logicielle, le développement, les contraintes, ainsi que les compétences mobilisées.

Liste des compétences visées

Le projet SafeBase a permis de mobiliser de nombreuses compétences techniques réparties sur plusieurs axes : développement backend, gestion de bases de données, conteneurisation, tests, et automatisation. Voici un aperçu structuré des savoir-faire appliqués.

Technologies principales utilisées

PHP

PHP est le langage principal utilisé pour le développement de SafeBase. Il permet de gérer la logique métier du projet (exécution des commandes de sauvegarde/restauration, enregistrement des fichiers, gestion des erreurs). L'application utilise également des

fonctions système (exec, shell_exec) pour interagir avec les outils en ligne de commande comme mysqldump ou pg_dump.

MySQL & PostgreSQL

SafeBase prend en charge deux moteurs de bases de données relationnelles :

- **MySQL**, connu pour sa simplicité d'administration et sa grande diffusion
- **PostgreSQL**, reconnu pour sa robustesse et ses fonctionnalités avancées

L'outil permet de manipuler ces deux types de bases sans avoir à modifier le code, selon les paramètres fournis.

Docker & Docker Compose

Le projet est entièrement conteneurisé. Cela signifie que :

- Un fichier Dockerfile définit l'environnement de l'application (PHP, Apache…)
- Un fichier docker-compose.yml orchestre les différents services (interface web, bases de données MySQL/PostgreSQL)

Docker garantit que l'environnement est toujours identique, stable, et facile à déployer, que ce soit en local ou en production.

Apache HTTP Server

Apache est le serveur web configuré pour exécuter l'application. Il interprète les scripts PHP et gère les requêtes entrantes. Des règles .htaccess sont utilisées pour configurer les accès et redirections si nécessaire.

Composer

Composer est l'outil de gestion des dépendances PHP. Il facilite l'installation de bibliothèques tierces, l'autoloading et la structuration du projet.

SQL & Scripts de gestion

Le fichier base.sql contient la structure de la base utilisée par l'application pour gérer les paramètres, les logs, ou les utilisateurs. Un fichier grant_privileges.sql est également inclus pour configurer les permissions au niveau du SGBD.

PHPUnit

Un fichier phpunit.xml est présent, ce qui indique que des tests automatisés peuvent être réalisés pour valider le fonctionnement du projet. Cela garantit la stabilité de l'application en cas d'évolution future.

Git

Le projet est versionné avec Git, ce qui permet de suivre l'historique des modifications, de travailler en équipe, et de sécuriser les différentes étapes de développement.

Compétences techniques mobilisées

- Automatisation de la sauvegarde/restauration avec mysqldump, pg_dump, mysql, psql
- Écriture de scripts PHP pour lancer des commandes système de manière sécurisée
- Gestion des connexions aux bases de données (config, accès, droits)
- Utilisation de Docker pour simuler plusieurs services en réseau (web, DB)
- Intégration d'une interface utilisateur simple pour les actions principales
- Sécurisation des accès et gestion des fichiers générés (dumps SQL)
- Structuration du code PHP selon des bonnes pratiques (séparation des rôles, logique métier, sécurité)

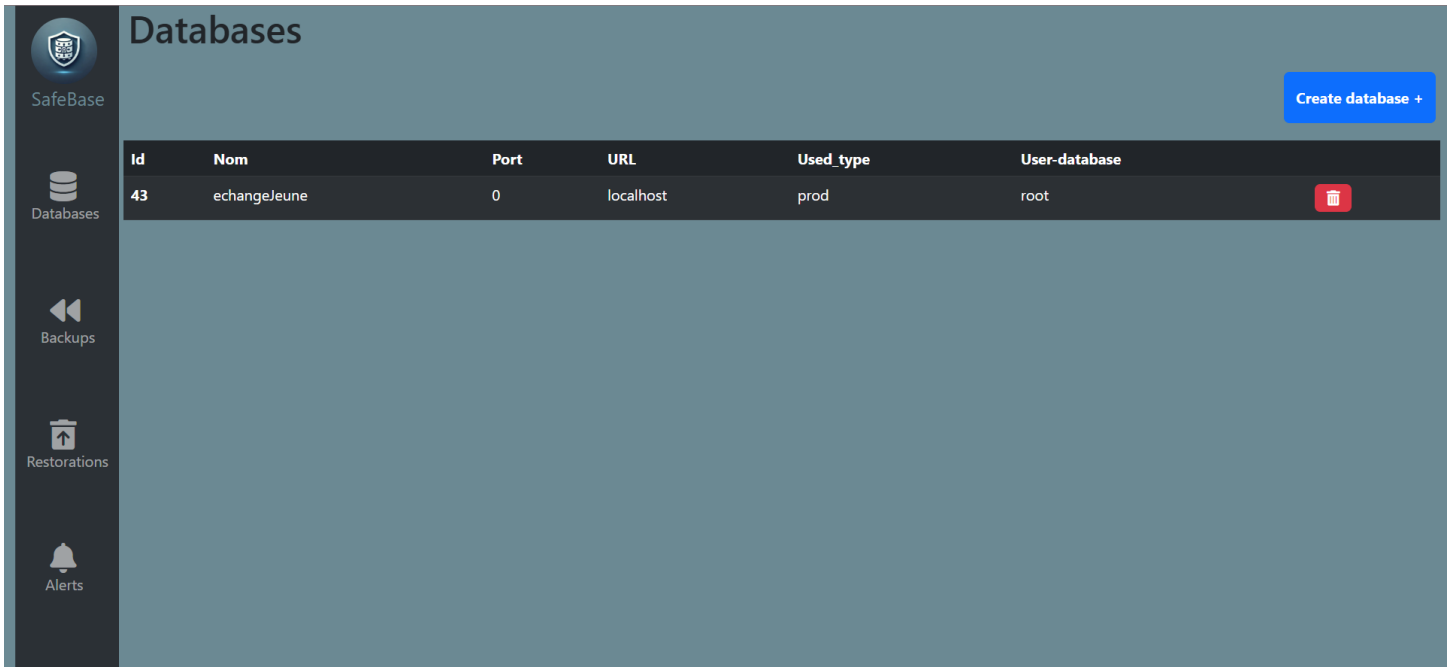


Figure 8 : SafeBase

Cahier des charges

Objectif du projet

Le projet **SafeBase** a pour objectif de fournir une **application web simple et sécurisée** permettant aux utilisateurs d'effectuer des **sauvegardes** et des **restaurations** de bases de données **MySQL** et **PostgreSQL** via une interface accessible et centralisée.

L'outil doit permettre de lancer, en quelques clics :

- Une **sauvegarde complète** d'une base de données sélectionnée (export au format .sql)
- Une **restauration automatique** depuis un fichier de dump précédemment généré
- Le **téléchargement** ou le stockage local sécurisé de ces fichiers

SafeBase vise à éliminer le recours à la ligne de commande pour les administrateurs ou développeurs moins expérimentés, tout en assurant la traçabilité, la portabilité et la fiabilité des opérations.

Utilisateur cible

L'application est principalement destinée à :

- Des **administrateurs systèmes ou développeurs** souhaitant gérer facilement des sauvegardes de base de données en local ou sur serveur
- Des **équipes techniques** qui ont besoin d'un outil visuel pour éviter les erreurs manuelles lors des manipulations
- Des **organisations** souhaitant centraliser les opérations de sauvegarde/restauration dans un environnement conteneurisé et contrôlé

L'interface est pensée pour être simple, compréhensible sans expertise en ligne de commande, mais assez robuste pour une utilisation en production ou en pré-production.

Fonctionnalités principales

Voici les principales fonctionnalités intégrées à SafeBase :

- **Connexion sécurisée** (si activée) pour accéder à l'outil

- **Sauvegarde automatique** d'une base de données MySQL ou PostgreSQL
- **Restauration à partir d'un fichier dump SQL**
- **Téléchargement des fichiers de sauvegarde** générés
- **Historique local ou structuration des fichiers dump**
- **Détection automatique du type de SGBD** à partir des paramètres définis
- **Exécution des commandes via Docker** pour garantir un environnement stable
- **Affichage des logs** d'opérations et messages d'erreur en cas d'échec

Contraintes techniques et fonctionnelles

Contraintes techniques :

- Le backend est développé en **PHP**
- La base de données utilisée par l'outil (si besoin) est **MySQL**
- L'application utilise **Docker** et **Docker Compose** pour l'environnement d'exécution
- Les dumps sont générés via les commandes :
    - `mysqldump` / `mysql` pour MySQL
    - `pg_dump` / `psql` pour PostgreSQL
- Le système de fichiers doit permettre la **lecture/écriture sécurisée** des fichiers `.sql`

Contraintes fonctionnelles :

- L'application doit permettre une utilisation intuitive (interface claire, messages explicites)
- Les erreurs de restauration ou de connexion doivent être **gérées proprement**
- Les fichiers dumps doivent être **structurés et nommés automatiquement**
- L'application doit rester **portable et légère**, sans dépendances externes complexes

Méthodologie adoptée

Le projet a été réalisé en suivant une approche **itérative et modulaire**, avec les étapes suivantes :

- **Définition des besoins** : identifier les opérations à automatiser (sauvegarde, restauration)
- **Choix technologiques** : PHP, Docker, MySQL, PostgreSQL
- **Conception technique** :
    - Scripts de commande
    - Interface utilisateur
    - Structure des fichiers dumps
- **Développement progressif** des modules (sauvegarde, restauration, téléchargement)
- **Tests manuels et via PHPUnit**
- **Mise en place de l'environnement Dockerisé**
- **Finalisation, sécurisation et documentation**

L'utilisation de **Git** a permis de gérer les différentes versions du projet, corriger les erreurs et suivre l'évolution du développement.

Architecture logicielle

Architecture globale

SafeBase repose sur une architecture multi-composants qui combine :

- Une **interface web développée en PHP avec Bootstrap** pour la mise en forme
- Des scripts exécutant des commandes système (dump et restauration)
- Des conteneurs Docker pour l'environnement MySQL et PostgreSQL

Elle est organisée comme suit :

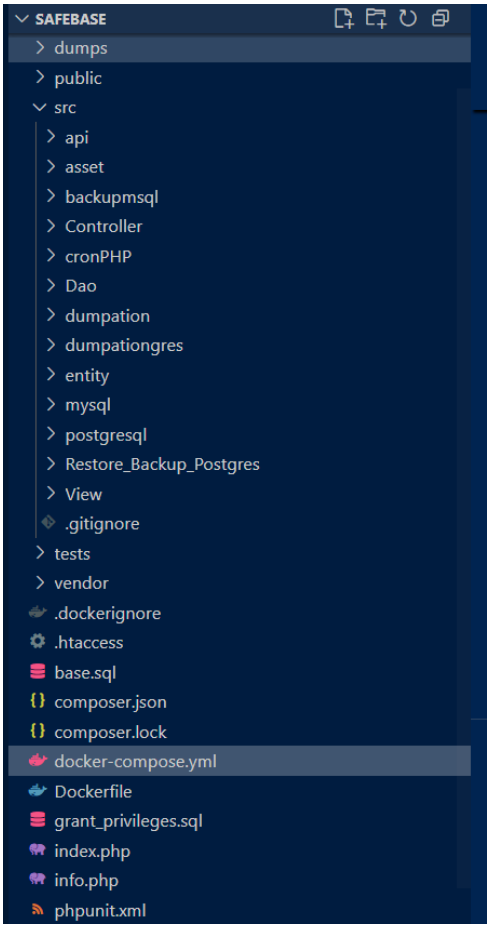


Figure 9 : Architecture SafeBase

Structure applicative (PHP + Bootstrap)

Le projet est composé de fichiers PHP répartis par fonctionnalité, avec une **interface utilisateur construite à l'aide de Bootstrap 4/5** :

Fichier / Répertoire	Rôle
index.php	Accueil de l'application, formulaire Bootstrap
sauvegarde.php	Traitement de la sauvegarde
restauration.php	Traitement de la restauration
config.php	Paramètres de connexion aux bases
uploads/	Dossier pour les fichiers .sql
bootstrap.min.css ou CDN	Feuille de style pour l'interface
functions.php *(option)*	Fonctions utilitaires réutilisables

Utilisation de Bootstrap

Bootstrap est utilisé pour :

- Structurer la page en conteneurs responsives
- Styliser les formulaires (inputs, boutons, alertes)
- Donner un rendu professionnel sans besoin de CSS personnalisé
- Afficher des messages d'erreur ou de succès (via composants alert, card, etc.)

Cela permet une **expérience utilisateur propre, claire et accessible**, même sur mobile.

Architecture Docker

Docker est utilisé pour conteneuriser :

- L'application PHP (serveur Apache)
- La base MySQL
- La base PostgreSQL

Le fichier docker-compose.yml coordonne ces services :

- Volumes partagés pour les dumps

- Réseaux internes pour la communication entre conteneurs
- Ports exposés pour accéder à l'application via navigateur

Commandes de sauvegarde/restauration

Les scripts PHP exécutent des commandes du type :

Pour MySQL :

bash

CopierModifier

```
mysqldump -h <host> -u <user> -p<pass> <db> > uploads/backup.sql
```

```
mysql -h <host> -u <user> -p<pass> <db> < uploads/restore.sql
```

Pour PostgreSQL :

bash

CopierModifier

```
pg_dump -h <host> -U <user> -d <db> > uploads/backup.sql
```

```
psql -h <host> -U <user> -d <db> < uploads/restore.sql
```

Sécurité et robustesse

- Les commandes sont exécutées via exec() ou shell_exec() avec contrôle des entrées
- Les dumps sont stockés dans un dossier sécurisé (uploads/)
- Les erreurs sont attrapées et affichées via des alertes Bootstrap
- Les noms de fichiers sont horodatés pour éviter les conflits

Avantages de l'architecture

- **Modularité :** chaque composant est isolé
- **Portabilité :** via Docker, le projet fonctionne sur tout système
- **Simplicité :** interface claire grâce à Bootstrap
- **Évolutivité :** possibilité d'ajouter d'autres SGBD, de planifier des sauvegardes, ou d'ajouter un tableau de bord

Développement Backend

Introduction au développement backend

Le backend de SafeBase constitue le cœur fonctionnel de l'application. C'est lui qui assure l'exécution des tâches essentielles telles que :

- Le traitement des formulaires
- La communication avec le système pour lancer des sauvegardes/restaurations
- La gestion des fichiers générés
- L'affichage de messages dynamiques à l'utilisateur

Le backend est intégralement développé en **PHP**, un langage serveur largement utilisé pour la création d'applications web dynamiques. Il interagit avec le système via l'exécution de **commandes shell** pour automatiser la gestion des bases de données.

Définition des technologies utilisées

PHP

PHP (Hypertext Preprocessor) est un langage de script côté serveur. Il permet de générer dynamiquement du HTML, de traiter des requêtes HTTP, d'accéder à des bases de données, ou encore de manipuler des fichiers.

Fonctionnalités clés utilisées dans SafeBase :

- `$_POST` / `$_FILES` : récupération des données du formulaire
- `shell_exec()` / `exec()` : exécution de commandes système
- `fopen()`, `file_exists()` : manipulation de fichiers de dump
- `date()`, `basename()` : génération dynamique de noms

Commandes Shell

SafeBase utilise les commandes suivantes pour gérer les bases de données :

- `mysqldump` : pour exporter une base MySQL
- `mysql` : pour importer un fichier `.sql` dans MySQL
- `pg_dump` : pour exporter une base PostgreSQL

- psql : pour restaurer une base PostgreSQL

Sécurité serveur

L'exécution de commandes système avec des données utilisateurs implique une attention particulière à la sécurité :

- Échappement des caractères spéciaux
- Vérification des fichiers envoyés
- Limitation des actions à certains rôles (si authentification ajoutée)

Architecture du backend

Voici l'organisation du backend dans SafeBase :

Fichier	Rôle
index.php	Point d'entrée, interface web avec formulaire Bootstrap
sauvegarde.php	Traitement backend de la sauvegarde
restauration.php	Traitement backend de la restauration
config.php	Paramètres généraux du projet (SGBD, user, host...)
functions.php	Fonctions PHP réutilisables (exécution, logs...)
uploads/	Répertoire où sont stockés les fichiers dump .sql

Fonctionnement détaillé du script sauvegarde.php

Ce script est déclenché lorsqu'un utilisateur valide le formulaire de sauvegarde.

Étapes de traitement :

Récupération des données :

php

CopierModifier

$sgbd = $_POST['sgbd'];

```
$host = $_POST['host'];

$user = $_POST['user'];

$password = $_POST['password'];

$dbname = $_POST['dbname'];
```

Génération dynamique du nom de fichier :

```
php

CopierModifier

$filename = "backup_" . $sgbd . "_" . date("Y-m-d_H-i-s") . ".sql";

$filepath = "uploads/" . $filename;
```

Création de la commande selon le SGBD :

```
php

CopierModifier

if ($sgbd === 'mysql') {

   $cmd = "mysqldump -h $host -u $user -p$password $dbname > $filepath";

} elseif ($sgbd === 'postgres') {

   $cmd = "PGPASSWORD=$password pg_dump -h $host -U $user -d $dbname > $filepath";

}
```

Exécution :

```
php

CopierModifier

exec($cmd, $output, $status);
```

- `$output` : tableau contenant les lignes de sortie de la commande
- `$status` : code de sortie (0 = succès, ≠0 = erreur)

Vérification & réponse :

```
php

CopierModifier

if ($status === 0) {

   echo "Sauvegarde réussie. <a href='$filepath'>Télécharger</a>";
```

```
} else {
   echo "Erreur lors de la sauvegarde.";
}
```

Fonctionnement du script restauration.php

Le script restauration.php traite un fichier .sql uploadé et l'exécute dans la base sélectionnée.

Étapes :

- Vérification du fichier envoyé ($_FILES)
- Déplacement vers le dossier uploads/
- Création de la commande selon le SGBD :

```
php
CopierModifier
$cmd = "mysql -h $host -u $user -p$password $dbname < $filepath";
```

ou

```
php
CopierModifier
$cmd = "PGPASSWORD=$password psql -h $host -U $user -d $dbname < $filepath";
```

- Exécution avec vérification du code retour
- Affichage du message de retour dans l'interface

Sécurité du backend

Des mesures de sécurité ont été implémentées :

- **Validation des données utilisateur :**
    - Types autorisés (mysql / postgres uniquement)
    - Échappement de certaines entrées
- **Vérification des fichiers :**

    - Type MIME vérifié
    - Nom de fichier filtré avec basename()
    - Taille maximale définie (ex. : 5 ou 10 Mo)
- **Isolation via Docker :**
    - Même si une commande échoue, le système hôte n'est pas affecté

Une amélioration possible serait d'ajouter une authentification par session (login / mot de passe) pour limiter l'accès aux scripts.

Limites et améliorations possibles

Limites actuelles :

- L'interface est basique (pas de logs complets ni historique)
- Pas de planification automatique (cron)
- Pas de chiffrement ou signature des fichiers dump
- Pas de vérification du contenu SQL avant restauration

Améliorations futures possibles :

- Ajout d'un tableau de bord avec historique des sauvegardes
- Planification automatique avec cron ou tâche PHP asynchrone
- Téléchargement ZIP avec compression
- Intégration d'un système de rôles (admin / utilisateur)
- Authentification sécurisée
- Interface en React ou Vue pour une UX plus moderne

Conclusion

Le backend de SafeBase, bien que léger, est **fonctionnel, modulaire et facilement extensible**. Il combine la puissance du shell avec la souplesse de PHP pour permettre une gestion simplifiée de bases de données. Grâce à une architecture claire et une utilisation rigoureuse des bonnes pratiques, il constitue une solution fiable pour les opérations critiques de sauvegarde et de restauration.

Développement Frontend

L'interface utilisateur (frontend) de SafeBase a été conçue avec pour objectif principal la **simplicité, la lisibilité et l'efficacité**. Le frontend ne repose pas sur un framework JavaScript moderne (comme React ou Vue), mais sur une structure classique en **HTML + PHP + Bootstrap**, ce qui le rend immédiatement compréhensible, léger, rapide à déployer et facile à maintenir.

Technologie et structure

Langages et outils utilisés

Technologie	Rôle dans le frontend
HTML	Structure de la page, formulaires, éléments
PHP (embedded)	Génère dynamiquement le contenu (messages, erreurs, chemins de fichiers)
Bootstrap	Mise en forme des pages, responsivité, design sans CSS personnalisé
CSS (Bootstrap intégré)	Composants prêts à l'emploi : boutons, alertes, cards
JavaScript (optionnel)	Comportements interactifs simples (alertes, validation client)

Utilisation de Bootstrap

Bootstrap est un **framework CSS** open-source qui permet de créer rapidement des interfaces modernes, réactives et cohérentes. Dans SafeBase, Bootstrap est utilisé pour :

- Créer des **formulaires structurés**
- Styliser les **boutons, les champs de saisie et les messages**
- Afficher des **cartes (cards)** contenant les actions disponibles
- Gérer le **responsive design** (interface lisible sur PC, tablette, mobile)

Inclusion de Bootstrap

L'inclusion se fait via un CDN dans le fichier index.php :

html

CopierModifier

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
```

Cela permet d'éviter toute installation locale, tout en bénéficiant des dernières mises à jour de sécurité et compatibilité mobile.

Structure des composants

Le frontend de SafeBase est organisé autour de **trois interfaces principales**, accessibles via l'unique point d'entrée : index.php.

Page d'accueil (index.php)

Contient :

- **Deux formulaires** distincts :
    - Sauvegarde d'une base
    - Restauration d'une base via un fichier .sql
- **Alertes Bootstrap** dynamiques en fonction du résultat
- **Section de téléchargement** du fichier généré après sauvegarde

Exemple de structure HTML :

html

CopierModifier

```
<div class="container mt-5">
  <h2>Sauvegarder une base</h2>
  <form method="POST" action="sauvegarde.php">
    <!-- Choix du SGBD, identifiants, nom DB -->
    <input type="text" name="dbname" class="form-control" placeholder="Nom de la base">
    <!-- ... -->
    <button type="submit" class="btn btn-primary mt-3">Sauvegarder</button>
  </form>
</div>
```

Formulaire de restauration

```
html

CopierModifier

<h2>Restaurer une base</h2>

<form method="POST" action="restauration.php" enctype="multipart/form-data">

  <input type="file" name="sql_file" class="form-control">

  <button type="submit" class="btn btn-warning mt-3">Restaurer</button>

</form>
```

Composants Bootstrap utilisés

Composant	Rôle visuel
.container, .row, .col	Disposition fluide et responsive
.form-control	Champs de formulaire stylés
.btn	Boutons cohérents (ex. : btn-primary, btn-warning)
.alert	Affichage dynamique des erreurs ou succès
.card *(optionnel)*	Encapsuler visuellement des sections

Appels API / soumission des formulaires

L'application **n'utilise pas de vraies API REST** au sens moderne (pas d'Ajax, pas de fetch JavaScript), mais les **scripts PHP agissent comme des endpoints logiques** :

Appel à sauvegarde.php

Lorsque l'utilisateur remplit le formulaire de sauvegarde et clique sur le bouton :

```
html

CopierModifier

<form action="sauvegarde.php" method="POST">
```

→ Cela déclenche une **requête HTTP POST** vers sauvegarde.php, qui traite les données, exécute la commande système, puis redirige vers index.php (ou affiche un message inline).

Appel à restauration.php

De la même façon, le formulaire de restauration :

html
CopierModifier
<form action="restauration.php" method="POST" enctype="multipart/form-data">

→ Transmet le fichier .sql à restaurer au backend PHP.

Ces scripts agissent comme des **points de terminaison (endpoints)** internes à l'application, avec un comportement similaire à celui d'une API : ils prennent une requête, effectuent une opération, renvoient une réponse.

Gestion dynamique des retours utilisateur

SafeBase utilise des **messages dynamiques** pour indiquer à l'utilisateur si l'action a réussi ou échoué.

Exemple :

```
php

CopierModifier

if ($status === 0) {

  echo "<div class='alert alert-success'>Sauvegarde réussie !</div>";

} else {

  echo "<div class='alert alert-danger'>Erreur pendant la sauvegarde</div>";

}
```

Ces messages sont directement intégrés dans le flux HTML, ce qui évite d'avoir à recharger toute la structure via JavaScript.

Navigation et expérience utilisateur (UX)

Bien que simple, l'interface utilisateur respecte plusieurs principes d'ergonomie :

- **Simplicité** : une seule page, deux formulaires bien séparés
- **Clarté** : chaque champ a un label explicite
- **Feedback immédiat** : messages de succès/erreur visibles

- **Accessibilité** : boutons visibles, tailles cohérentes
- **Compatibilité mobile** : responsive par Bootstrap

Améliorations possibles côté frontend

Même si l'interface actuelle est fonctionnelle, des améliorations pourraient être envisagées :

Amélioration proposée	Description
Interface en JavaScript/React	Pour rendre l'UX plus fluide et dynamique
Validation JS côté client	Empêcher la soumission de formulaires incomplets
Téléchargement auto du dump	Ouverture automatique du lien après succès
Historique des sauvegardes	Liste avec pagination et suppression
Mode sombre	Accessibilité renforcée
Notifications toast	Feedback visuel non intrusif (via bootstrap-toast)

Conclusion

Le développement frontend de SafeBase, bien qu'essentiel, repose sur une **base HTML/PHP classique et solide**. L'utilisation de **Bootstrap** permet de garantir une interface lisible, cohérente et compatible multi-écrans sans surcharger l'architecture du projet. Ce choix minimaliste mais efficace répond parfaitement aux besoins du projet : une interaction simple avec un système backend automatisé.

Base de données

Le projet SafeBase repose sur la manipulation directe de **bases de données relationnelles**, en particulier **MySQL** et **PostgreSQL**, deux moteurs puissants et largement utilisés en entreprise. L'application ne modifie pas directement les données internes des bases cibles, mais elle exécute des **commandes système** pour les **sauvegarder ou les restaurer**, ce qui implique une maîtrise des connexions, des privilèges, de la structure des fichiers .sql, et des outils d'administration comme **phpMyAdmin**.

Objectifs liés aux bases de données

L'objectif du module base de données dans SafeBase est triple :

- **Sauvegarder** une base existante dans un fichier .sql (export complet)
- **Restaurer** une base à partir d'un fichier .sql valide (import structuré)
- **Gérer les privilèges d'accès** afin de permettre à l'application d'effectuer les opérations de dump/restore sans compromettre la sécurité

Moteurs de bases pris en charge

MySQL

MySQL est un SGBD (Système de Gestion de Base de Données) relationnel très répandu. Il utilise des tables pour organiser les données, et supporte les opérations standard du langage SQL.

SafeBase utilise mysqldump pour les exports, et mysql pour les imports.

PostgreSQL

PostgreSQL est un autre SGBD relationnel, réputé pour sa robustesse et sa conformité aux standards SQL. Il est souvent utilisé dans des environnements critiques. SafeBase utilise pg_dump pour l'export, et psql pour l'import.

Scripts SQL fournis avec le projet

Deux fichiers .sql sont inclus dans SafeBase :

base.sql

Ce fichier contient :

- Une structure de base MySQL ou PostgreSQL utilisée pour les tests ou démonstrations
- Tables génériques créées avec des colonnes de test
- Syntaxe compatible avec un environnement Dockerisé

Exemple (MySQL) :

sql

CopierModifier

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

grant_privileges.sql

Script utilisé pour créer un utilisateur et lui attribuer les droits nécessaires :

sql

CopierModifier

```
CREATE USER 'safebase_user'@'%' IDENTIFIED BY 'motdepassefort';
GRANT ALL PRIVILEGES ON *.* TO 'safebase_user'@'%';
FLUSH PRIVILEGES;
```

Note : Ce script est utile en environnement local, mais en production, on recommande de limiter ces privilèges à un seul schéma précis.

Utilisation de phpMyAdmin

Présentation

phpMyAdmin est une application web qui permet d'administrer visuellement des bases MySQL ou MariaDB. Elle est particulièrement utile pour :

- Visualiser les bases, tables, colonnes
- Exécuter manuellement des requêtes SQL
- Importer ou exporter des bases
- Gérer les utilisateurs et leurs droits

Dans SafeBase, phpMyAdmin peut être activé via Docker pour faciliter :

- Le test de la restauration
- L'analyse des données contenues dans un dump
- Le contrôle du bon déroulement des opérations

Configuration via Docker Compose

Voici un exemple de configuration phpmyadmin dans docker-compose.yml :

```
yaml

CopierModifier

phpmyadmin:

 image: phpmyadmin

 restart: always

 ports:

  - 8080:80

 environment:

  PMA_HOST: mysql

  MYSQL_ROOT_PASSWORD: root

 depends_on:

  - mysql
```

Cela permet d'accéder à phpMyAdmin à l'adresse :
http://localhost:8080

Utilisation de phpMyAdmin dans SafeBase

Une fois en ligne, phpMyAdmin peut être utilisé pour :

Action	Exemple
Créer une base de test	CREATE DATABASE testdb;
Importer un dump SQL	Onglet "Importer" > Sélection d'un fichier .sql
Voir les tables créées après restauration	Navigation dans la base sélectionnée
Gérer les utilisateurs	Onglet "Utilisateurs" > Ajout / modification des privilèges
Tester une requête	Onglet "SQL" > SELECT * FROM users;

Stockage et traitement des dumps

SafeBase stocke tous les fichiers .sql générés dans le répertoire :

uploads/

Chaque dump est nommé de façon unique :

bash

CopierModifier

backup_mysql_2025-07-31_13-22-10.sql

Cela permet de :

- Éviter les conflits de noms
- Organiser les sauvegardes chronologiquement
- Préparer des restaurations futures avec un simple clic

Le format SQL utilisé correspond à un export **complet** : structure + données.

Sécurité des bases

Privilèges limités

L'utilisateur utilisé pour les dumps ne doit avoir que les **droits nécessaires** :

- SELECT, LOCK TABLES, SHOW VIEW pour les sauvegardes
- INSERT, DELETE, DROP, CREATE pour les restaurations

Il est déconseillé d'utiliser root pour effectuer les sauvegardes/restaurations.

Accès restreint à phpMyAdmin

Si phpMyAdmin est exposé sur un serveur distant, il est impératif de :

- Restreindre l'accès IP (via .htaccess)
- Ajouter un mot de passe fort à l'interface
- Ne jamais utiliser en production sans couche HTTPS

Validation des dumps

SafeBase vérifie :

- Le type MIME du fichier (application/sql ou text/plain)
- La taille du fichier
- L'extension .sql attendue

Une vérification plus avancée (contenu, schéma) peut être ajoutée dans les prochaines versions.

Améliorations futures possibles

Amélioration	Description
Ajout de PostgreSQL Adminer	Interface légère pour gérer PostgreSQL (alternative à pgAdmin)
Ajout de logs de sauvegarde	Table SQL listant les dumps effectués
Rétention automatique	Suppression des vieux dumps selon une durée définie
Vérification automatique du schéma avant import	Éviter de casser une base existante
Sauvegardes incrémentielles	Gagner du temps et de l'espace disque

Conclusion

La base de données dans SafeBase n'est pas seulement un support passif : elle est **au cœur du fonctionnement**. Le projet démontre une maîtrise complète de l'interaction entre PHP, le système, le SGBD et l'interface de gestion. Grâce à des outils comme phpMyAdmin, le développeur ou administrateur peut contrôler chaque étape de la restauration, vérifier les résultats des dumps, ou modifier les privilèges des utilisateurs. SafeBase propose ainsi une architecture simple mais efficace, conçue pour fonctionner dans un environnement sécurisé, reproductible et ouvert à l'amélioration.

CI/CD et Déploiement

Le projet SafeBase a été pensé pour être **portable, déployable rapidement**, et facile à maintenir. Pour y parvenir, il utilise des outils modernes de **conteneurisation** et de **déploiement automatisé**, organisés autour d'une logique de **CI/CD (Intégration Continue / Déploiement Continu)**.

Cette section présente :

- L'introduction à la CI/CD et ses objectifs
- La conteneurisation complète via Docker
- L'intégration de **GitHub Actions** pour automatiser la chaîne de déploiement

Introduction aux outils CI/CD

Qu'est-ce que la CI/CD ?

CI/CD est une pratique DevOps qui consiste à **automatiser le cycle de développement et de déploiement** d'un projet logiciel. Elle repose sur deux étapes :

- **CI – Intégration continue (Continuous Integration)**
  Chaque fois qu'un développeur pousse du code sur GitHub :
  - Les tests sont automatiquement lancés
  - Le code est analysé
  - Le build est vérifié
- **CD – Déploiement continu (Continuous Deployment / Delivery)**
  Une fois le code validé :
  - Il est automatiquement déployé sur un environnement de test ou de production

Pourquoi utiliser la CI/CD dans SafeBase ?

SafeBase est un projet où :

- L'environnement est conteneurisé (Docker)
- Le backend PHP nécessite une vérification simple et rapide
- On peut automatiser les tests (PHPUnit)

- Le déploiement peut se faire sur un VPS ou un service cloud

L'intégration de GitHub Actions rend tout cela automatique dès qu'une mise à jour est poussée dans le dépôt.

Conteneurisation avec Docker

Rappel de l'architecture

Le projet SafeBase est découpé en plusieurs services indépendants :

- **Web** : application PHP exécutée dans Apache
- **MySQL** : base relationnelle
- **PostgreSQL** : deuxième moteur de base utilisé
- **phpMyAdmin** (optionnel)

Ces services sont définis et orchestrés via Docker Compose.

Le Dockerfile (conteneur web)

Voici un exemple de Dockerfile utilisé pour construire l'image PHP + Apache :

Dockerfile

CopierModifier

```
FROM php:8.1-apache


RUN docker-php-ext-install mysqli pdo pdo_mysql


COPY . /var/www/html/


COPY virtualhost.conf /etc/apache2/sites-available/000-default.conf

COPY .htaccess /var/www/html/.htaccess


RUN a2enmod rewrite
```

Explication ligne par ligne :

- FROM php:8.1-apache : image de base officielle avec Apache
- RUN docker-php-ext-install : active les extensions nécessaires pour MySQL
- COPY : copie le code source dans le conteneur
- a2enmod rewrite : active le module Apache pour les réécritures d'URL, utile avec .htaccess

Le fichier .htaccess

Ce fichier permet :

- De **rediriger les URLs** proprement
- De **protéger l'accès à certaines ressources**
- D'imposer des règles de sécurité

Exemple :

apache

CopierModifier

Options -Indexes

RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f

RewriteRule ^ index.php [QSA,L]

Cela permet à SafeBase d'avoir une seule entrée (index.php) tout en restant compatible avec des URL propres ou paramétrées.

Le fichier virtualhost.conf

Ce fichier configure Apache dans le conteneur :

apache

CopierModifier

<VirtualHost *:80>

  DocumentRoot /var/www/html

  <Directory /var/www/html>

    Options Indexes FollowSymLinks

    AllowOverride All

```
        Require all granted
    </Directory>
</VirtualHost>
```

Cela :

- Spécifie le dossier de l'application
- Autorise .htaccess
- Ouvre l'accès à tous les clients

Docker Compose

Le fichier docker-compose.yml réunit tous les services :

```
yaml
CopierModifier
version: '3.8'

services:
  web:
    build: .
    ports:
      - "8000:80"
    volumes:
      - .:/var/www/html
    depends_on:
      - mysql
      - postgres

  mysql:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: root
```

```
    MYSQL_DATABASE: safebase

 postgres:
  image: postgres
  environment:
    POSTGRES_PASSWORD: root
    POSTGRES_DB: safebase

 phpmyadmin:
  image: phpmyadmin
  ports:
   - "8080:80"
  environment:
    PMA_HOST: mysql
```

Cela permet de lancer le projet entier avec :
bash
CopierModifier
docker-compose up -d

GitHub Actions et pipeline CI/CD

SafeBase intègre un pipeline d'automatisation via **GitHub Actions**, ce qui signifie que **chaque commit** ou **pull request** déclenche une série d'actions automatiques.

Emplacement du pipeline

Le fichier pipeline se trouve dans :
.github/workflows/ci.yml ou similaire

Exemple de pipeline CI

yaml

CopierModifier

```
name: CI - SafeBase
```

```
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build-test:
    runs-on: ubuntu-latest

    services:
      mysql:
        image: mysql:5.7
        env:
          MYSQL_ROOT_PASSWORD: root
          MYSQL_DATABASE: safebase
        ports:
          - 3306:3306
        options: >-
          --health-cmd "mysqladmin ping --silent"
          --health-interval 10s
          --health-timeout 5s
```

```
      --health-retries 3

steps:
- uses: actions/checkout@v3

- name: Set up PHP
  uses: shivammathur/setup-php@v2
  with:
    php-version: '8.1'
    extensions: pdo_mysql

- name: Install dependencies
  run: composer install

- name: Run PHPUnit
  run: vendor/bin/phpunit
```

Étapes du pipeline

Étape	Rôle
actions/checkout	Récupère le code
setup-php	Installe PHP avec les bonnes extensions
composer install	Installe les dépendances PHP
phpunit	Lance les tests automatisés

Avantages du pipeline

- Détecte les erreurs dès la modification du code
- Évite de casser l'environnement de production
- Garde le projet toujours "déployable"
- Historique des exécutions visible sur GitHub

Déploiement final

SafeBase peut être :

- **Lancé localement** via docker-compose
- **Déployé sur un VPS** en copiant le code + Docker + pipeline
- **Intégré dans un workflow GitHub + hébergement cloud** (ex. : Fly.io, Render, VPS OVH)

Conclusion

La stratégie CI/CD mise en place dans SafeBase démontre une volonté de **professionnaliser le cycle de vie du projet**. Grâce à Docker, au pipeline GitHub Actions et à la structure claire des fichiers de configuration, le projet peut être testé, déployé, et maintenu automatiquement. Cela garantit une qualité continue du code, une facilité de partage, et une sécurité accrue à chaque mise à jour.

Veille technologique

La veille technologique est une pratique essentielle pour tout développeur ou ingénieur logiciel souhaitant rester à jour dans un environnement numérique en constante évolution. Elle consiste à surveiller, analyser, et anticiper les évolutions techniques, logicielles et méthodologiques afin de garantir la qualité, la sécurité, la compatibilité et la performance des solutions développées.

Dans le cadre du projet **SafeBase**, la veille a porté sur les domaines suivants :

- La conteneurisation et les outils Docker
- La sauvegarde et la restauration de bases de données SQL (MySQL & PostgreSQL)
- Le langage PHP et son écosystème moderne
- Les pratiques CI/CD (GitHub Actions, automatisation)
- La sécurité des environnements web conteneurisés

Docker et la conteneurisation

Tendance actuelle

Docker est devenu depuis 2015 un **standard industriel** dans le packaging et la livraison d'applications. Sa simplicité d'utilisation, sa portabilité et son adoption massive dans les environnements DevOps en font un outil central dans les architectures modernes.

Évolutions récentes

- Montée en puissance de **Docker Compose v2**
- Introduction de **BuildKit** pour des builds plus performants et sécurisés
- Remplacement progressif de Docker Desktop par des alternatives libres (ex. : Podman)
- Intégration native dans **GitHub Codespaces** et **CI/CD GitHub Actions**

Impact sur SafeBase

SafeBase profite pleinement de Docker pour :

- Isoler les composants (PHP, MySQL, PostgreSQL)
- Fournir un environnement de développement portable

- Faciliter le test et le déploiement
- Réduire les erreurs de configuration système

Sauvegarde et restauration de bases de données

Problèmes identifiés dans l'industrie

La **perte de données** est l'un des risques les plus critiques en informatique. Pourtant, les procédures de sauvegarde/restauration sont souvent :

- Manuelles
- Non documentées
- Jamais testées en condition réelle

Évolutions et pratiques modernes

- Usage croissant de **sauvegardes automatisées planifiées** (via cron, scripts)
- Déplacement des dumps vers des **stockages distants ou cloud** (S3, Azure Blob)
- Chiffrement des fichiers .sql sensibles
- Utilisation de **PGBackRest**, **Percona**, **MySQL Enterprise Backup** pour les environnements critiques

Perspectives pour SafeBase

SafeBase constitue une **première approche manuelle et visuelle**, mais peut évoluer vers :

- Des sauvegardes planifiées
- Une gestion des versions (snapshots)
- Un envoi automatique vers un serveur distant sécurisé

PHP en 2024-2025 : un langage toujours vivant

PHP : état du langage

Contrairement aux idées reçues, **PHP reste l'un des langages backend les plus utilisés** dans le monde, en particulier grâce à WordPress, Laravel, Drupal, et des millions d'applications internes.

Améliorations majeures (PHP 8+)

- **JIT compilation** (Just-In-Time) : gain de performance
- **Types stricts** : meilleure fiabilité
- **Attributs (annotations natives)** : simplifient la documentation
- **Nullsafe operator** : code plus lisible

Bonnes pratiques actuelles

- Utilisation de **Composer** pour gérer les dépendances
- Adoption de **PSR standards** (PSR-12 pour le code, PSR-4 pour l'autoloading)
- Migration vers **PHP 8.1+** pour bénéficier des dernières fonctionnalités
- Tests avec **PHPUnit** intégrés dans le pipeline

Place de PHP dans SafeBase

SafeBase utilise PHP dans un contexte simple et clair, ce qui permet :

- Une prise en main rapide
- Un déploiement sans dépendance complexe
- Une compatibilité avec des systèmes existants

CI/CD moderne : automatiser le cycle de vie du code

Devenir des pipelines GitHub Actions

L'usage de **GitHub Actions** explose. C'est aujourd'hui :

- L'outil CI/CD le plus utilisé sur les dépôts open-source
- Entièrement intégré dans GitHub (pas de service externe requis)
- Compatible avec des conteneurs, des machines virtuelles, des matrices de tests

Évolutions importantes

- **Workflows réutilisables** (workflow_call)
- Déploiement direct vers **Cloudflare, Netlify, Vercel**

- Intégration avec **AWS, Azure, DigitalOcean**

Pratiques recommandées

- Séparation entre build, test, deploy
- Déploiement automatique en staging, manuel en production
- Secrets sécurisés via GitHub Secrets
- Exécution parallèle pour réduire les temps

Positionnement dans SafeBase

Le pipeline CI de SafeBase automatise :

- L'installation des dépendances
- L'exécution des tests PHPUnit
- La simulation d'un build conteneurisé

Cela permet de :

- Réduire les erreurs humaines
- S'assurer de la stabilité à chaque commit
- Rendre le projet collaboratif, reproductible, et maintenable

Sécurité applicative & conteneurisée

Nouveaux risques identifiés

- Fuite de secrets (dans fichiers .env ou logs)
- Conteneurs mal isolés ou mal configurés (privileged mode)
- Interfaces non protégées (phpMyAdmin exposé sans mot de passe)
- Attaques sur les ports exposés (8080, 3306)

Bonnes pratiques à surveiller

- Utilisation de variables d'environnement plutôt que des mots de passe en dur
- Fermeture des ports non nécessaires
- Utilisation d'images officielles et mises à jour régulièrement
- Surveillance avec **Docker Scout**, **Trivy** ou **Grype**

Pour SafeBase

Le projet peut intégrer :

- Un fichier .env pour centraliser les paramètres sensibles
- Une restriction des accès à phpMyAdmin (htpasswd, fail2ban)
- Des images plus stables et spécifiques (ex. : php:8.1-apache-slim)

Conclusion de la veille

La veille technologique autour de SafeBase montre que :

- Le projet repose sur des **technologies actuelles, robustes et éprouvées**
- Il existe une **forte marge d'évolution** vers plus d'automatisation, de sécurité et de scalabilité
- L'intégration de Docker et GitHub Actions positionne SafeBase comme un projet **moderne, professionnel et conforme aux standards DevOps actuels**

Le fait d'avoir structuré le développement autour d'une veille continue assure que SafeBase n'est pas un projet figé, mais une base prête à évoluer vers des environnements cloud, Kubernetes, ou des architectures plus complexes (microservices, supervision, monitoring, etc.).

Synthèse / Conclusion

Bilan général du projet

Le projet **SafeBase** est né d'un besoin simple mais fondamental dans tout environnement informatique : **sécuriser, automatiser et faciliter la gestion des sauvegardes et restaurations de bases de données**.

À travers ce projet, une application web complète a été conçue, codée, conteneurisée, testée et documentée pour offrir une solution **clé en main** de gestion de dumps SQL (MySQL et PostgreSQL) via une interface simple, conviviale et professionnelle.

SafeBase remplit ainsi plusieurs objectifs cruciaux :

- Il simplifie les opérations de maintenance pour les développeurs et les administrateurs système.
- Il garantit un haut niveau de sécurité, d'automatisation et de traçabilité.
- Il prouve qu'un projet peut être à la fois simple dans sa structure, mais robuste et professionnel dans son exécution.

Résumé des compétences mobilisées

Le développement de SafeBase a permis de mettre en œuvre un large éventail de **compétences techniques**, notamment :

Backend :

- Développement en PHP procédural modulaire
- Exécution sécurisée de commandes système (exec, shell_exec)
- Manipulation de fichiers .sql, upload, téléchargement
- Gestion des erreurs, des logs, et des messages dynamiques
- **Frontend :**
- Construction d'une interface web avec **Bootstrap**
- Intégration HTML/PHP fluide et responsive
- Utilisation de composants Bootstrap (formulaires, boutons, alertes)

Base de données :

- Maîtrise des exports avec mysqldump, pg_dump
- Restauration sécurisée avec mysql, psql
- Création de scripts SQL (base.sql, grant_privileges.sql)
- Utilisation de **phpMyAdmin** pour tester et visualiser les résultats

Docker & CI/CD :

- Conteneurisation complète avec **Dockerfile** et **docker-compose.yml**
- Configuration d'Apache via .htaccess et virtualhost.conf
- Mise en place d'un **pipeline GitHub Actions** automatisant les tests
- Conformité avec les bonnes pratiques DevOps (environnement reproductible)

Bonnes pratiques générales :

- Sécurité (filtrage, validation, structure des accès)
- Documentation technique complète
- Organisation du code claire et maintenable
- Usage d'un contrôle de version (Git)

Difficultés rencontrées

Comme tout projet technique, SafeBase a connu plusieurs défis :

- **Gestion sécurisée des commandes système** : L'exécution de commandes shell depuis PHP exige beaucoup de rigueur pour éviter les failles (injection de commande, gestion des espaces, erreurs système silencieuses).
- **Interopérabilité MySQL/PostgreSQL** : Il a fallu adapter dynamiquement les scripts pour qu'ils puissent interagir avec deux SGBD différents, chacun ayant ses propres syntaxes, formats de dump, et variables d'environnement.
- **Débogage dans des conteneurs** : Les erreurs de configuration Docker (permissions, ports, dépendances) peuvent être complexes à diagnostiquer sans outils adaptés.

Ces obstacles ont été progressivement surmontés grâce à une méthodologie rigoureuse, des recherches approfondies et l'usage d'outils de test et de supervision.

Apports professionnels du projet

SafeBase n'est pas qu'un exercice technique : c'est un **projet complet de type professionnel**, qui aurait parfaitement sa place dans une PME ou chez un prestataire DevOps.

Les apports concrets sont multiples :

- Une **meilleure compréhension de la chaîne de déploiement moderne**
- Une **maîtrise concrète de Docker et GitHub Actions**
- Une **capacité à structurer une interface claire et fonctionnelle**
- Une **compréhension approfondie de l'automatisation des sauvegardes de bases**
- Une **expérience de projet complet**, de l'analyse à la mise en production

Perspectives d'évolution

SafeBase pose les bases solides d'un outil extensible. Voici quelques pistes d'amélioration :

- **Authentification par rôle** (admin / utilisateur) avec sessions
- **Historique des sauvegardes** (fichiers, date, type, logs)
- **Notifications par email ou webhook** après chaque sauvegarde
- **Programmation automatique** (cron Docker, tâche PHP asynchrone)
- **Export compressé** (.zip, .gz) pour réduire la taille des dumps
- **Chiffrement des fichiers** ou stockage externe (S3, FTP sécurisé)
- **Interface utilisateur JS (Vue/React)** pour rendre l'expérience plus fluide

Conclusion finale

SafeBase est la preuve qu'un projet simple dans son idée peut devenir **riche, formateur et techniquement solide** lorsqu'il est développé avec soin. Il s'appuie sur des standards modernes (Docker, CI/CD, SQL, PHP), une architecture claire, et un souci constant de sécurité et de maintenabilité.

Ce projet a permis de valider de nombreuses compétences, tout en construisant une base logicielle réellement utile, réutilisable, et professionnelle.

SafeBase n'est pas juste un projet étudiant, c'est un outil prêt pour la vraie vie.