ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

PROJET DE FIN D'ANNEE

4éme Année en Ingénierie Informatique et Réseaux

Mini-CI/CD : Une Plateforme d'Intégration et de Déploiement Continus pour Projets Java Maven

Réalisé par :

Omar Elbakouri
Mohammed Bensellam

Tuteur (s) :

Encadrant : Bakhouyi Abdellah

Année universitaire : 2024/2025

Remerciements

Tout d'abord, je tiens à saluer l'EMSI pour son engagement indéfectible envers l'excellence académique et sa vision avant-gardiste de former la prochaine génération de professionnels talentueux. Cette institution m'a fourni une base solide de connaissances et de compétences, indispensables pour le succès de mon stage. L'enseignement de qualité dispensé à l'EMSI a été le socle sur lequel j'ai pu bâtir mon expérience professionnelle, et je suis profondément reconnaissant pour les valeurs et l'éthique de travail qui m'ont été inculquées.

Avant de présenter mon travail, je souhaite exprimer ma profonde reconnaissance à toutes les personnes qui, de près ou de loin, m'ont apporté leur soutien tout au long de ce parcours. Que chacun et chacune trouve ici l'expression sincère de ma gratitude, tant sur le plan collectif qu'individuel.

Je tiens également à remercier chaleureusement M. *Bakhouyi Abdellah,* mon encadrant. Sa confiance en mes compétences, ainsi que sa patience dans l'explication des processus et des tâches, m'ont permis de progresser et de me développer de manière significative.

Enfin, je souhaite également exprimer toute ma gratitude à l'ensemble du corps professoral de l'EMSI pour l'intérêt soutenu qu'il accorde à la formation des futurs ingénieurs. Leur dévouement à l'enseignement et leur engagement envers la réussite de chaque étudiant sont des éléments essentiels qui ont contribué à faire de mon parcours une expérience enrichissante et formatrice.

Table des matières

Table des matières

Introduction générale

Dans un monde où le développement logiciel évolue à un rythme sans précédent, l'intégration continue et le déploiement continu (CI/CD) sont devenus des pratiques essentielles pour garantir la qualité, la fiabilité et la rapidité de livraison des applications. La capacité à automatiser les processus de build, de test et de déploiement représente aujourd'hui un avantage concurrentiel majeur pour les équipes de développement, permettant de réduire les erreurs humaines et d'accélérer le cycle de vie des logiciels.

Dans ce contexte, notre projet a consisté à concevoir et développer une plateforme légère et efficace, baptisée Mini-CI/CD, dédiée à l'automatisation des processus de build pour les projets Java Maven. Réalisé dans le cadre de notre formation en ingénierie informatique, ce projet vise à démontrer notre capacité à mettre en œuvre une architecture moderne basée sur le framework Spring Boot, tout en intégrant des technologies avancées telles que Docker pour l'isolation des environnements de build.

L'objectif principal de Mini-CI/CD est de proposer une solution intuitive et robuste permettant aux développeurs de configurer des projets, déclencher des builds, et visualiser clairement les résultats de ces builds. L'application répond à une problématique concrète : comment offrir une plateforme d'intégration continue accessible et efficace, capable de gérer les spécificités des projets Java Maven tout en garantissant l'isolation des environnements d'exécution.

Le projet s'est articulé autour de plusieurs axes majeurs : • Présentation du Contexte et des Objectifs : où nous détaillons la vision du projet, ses enjeux, et les motivations techniques qui le sous-tendent. • Technologies et Méthodologies : où nous expliquons les outils et frameworks sélectionnés, ainsi que l'approche multicouche adoptée pour assurer modularité et maintenabilité. • Développement de l'Application : où nous décrivons la structure du projet, les fonctionnalités implémentées (gestion des projets, exécution des builds, visualisation des résultats), ainsi que les défis techniques surmontés. • Résolution des Problèmes : où nous présentons les solutions innovantes mises en place pour résoudre les problèmes courants des systèmes CI/CD, notamment la gestion des répertoires de travail, l'accès aux dépôts Git, et la séparation claire des résultats de build. • Analyse des Résultats et Perspectives : où nous analysons les performances obtenues et proposons des pistes d'amélioration futures.

Ce projet nous a permis de mobiliser l'ensemble des compétences acquises durant notre cursus, tant sur le plan technique que méthodologique, tout en répondant à une problématique concrète dans le domaine de l'automatisation des processus de développement logiciel. Mini-CI/CD représente non seulement un outil fonctionnel, mais également une démonstration de notre capacité à concevoir et implémenter des solutions techniques complexes répondant à des besoins réels

numérique appliqué au tourisme.

Chapitre 1

Présentation du Contexte et des Objectifs

Introduction

Ce chapitre a pour objectif de présenter le cadre académique dans lequel s'inscrit notre projet Mini-CI/CD. Il introduit la problématique abordée, justifie les choix technologiques et présente le modèle de développement adopté. Nous aborderons également une étude des solutions existantes dans le domaine des plateformes d'intégration continue, les limitations constatées, ainsi que le planning prévisionnel mis en place pour structurer notre travail.

Contexte du projet

Aujourd'hui, les équipes de développement ont besoin d'outils d'intégration continue capables d'automatiser et de sécuriser chaque étape du processus de build : clonage de dépôts Git, compilation, exécution des tests, et visualisation claire des résultats. Malgré l'existence de nombreuses plateformes CI/CD sur le marché, beaucoup sont complexes à configurer, coûteuses, ou inadaptées aux projets de taille moyenne. Cela justifie le développement d'une nouvelle application, Mini-CI/CD, pensée pour simplifier ces processus tout en offrant une isolation des environnements de build grâce à la conteneurisation Docker..

Etude de l'existant

Limites des solutions actuelles

Les plateformes CI/CD existantes présentent souvent les limitations suivantes :

- Complexité de configuration et d'utilisation pour les petites équipes ;
- Coûts élevés pour les solutions commerciales complètes ;
- Manque de flexibilité pour s'adapter aux spécificités des projets Java Maven ;
- Problèmes d'isolation entre les différents builds, entraînant des conflits et des échecs ;
- Difficulté à visualiser clairement les résultats des builds par statut (réussis/échoués).

Solution proposée

Nous proposons de développer Mini-CI/CD, une application web basée sur Spring Boot, qui se distingue par :

- Une interface utilisateur intuitive pour la gestion des projets et des builds;
- Un système d'exécution de builds isolés dans des conteneurs Docker;
- Une gestion intelligente des dépôts Git, avec conversion automatique des URLs HTTPS en URLs Git ;
- Un mécanisme de nettoyage efficace des répertoires de travail entre les builds ;
- Une séparation claire des builds par statut (SUCCESS/FAILURE) pour une meilleure lisibilité ;
- La possibilité de créer des projets de démonstration directement dans le conteneur.

Cette solution vise à fournir une expérience simplifiée d'intégration continue, avec une isolation robuste des environnements, une gestion efficace des erreurs courantes et une visualisation claire des résultats..

Choix de modèle de développement

Nous avons adopté une méthodologie agile avec des cycles de développement itératifs. Ce choix permet :

- Une adaptation continue aux problèmes identifiés durant le développement ;
- Une amélioration progressive par feedback entre les membres de l'équipe ;
- Une résolution rapide des bugs et des limitations techniques rencontrés.

Planning prévisionnel

Le planning prévisionnel pour notre stage de six semaines est structuré comme suit :

Semaine	**Activité principale**
1	Analyse des besoins fonctionnels et non-fonctionnels
2	Conception de l'architecture (Spring Boot, Docker, base de données)
3-4	Développement des modules principaux (gestion des projets, exécution des builds
5	Résolution des problèmes techniques (isolation des builds, gestion des URLs Git)
6	Amélioration de l'interface utilisateur (séparation des builds par statut)
7	Tests, documentation technique, préparation du rapport

Ce planning a été conçu pour structurer efficacement le travail tout au long du projet, en permettant une progression continue et un aboutissement dans les délais impartis.

Conclusion

Ce premier chapitre a permis de poser les fondations du projet Mini-CI/CD, en présentant le besoin auquel répond notre application et les limites des solutions concurrentes. La solution proposée est à la fois légère, efficace et orientée développeur. Le modèle de développement choisi (agile) et le planning défini nous permettent d'aborder les prochaines étapes de manière structurée et progressive, en assurant la cohérence entre l'analyse des besoins, la conception, l'implémentation et la résolution des problèmes techniques rencontrés.

Chapitre 2

Spécification des besoins

Introduction

Dans ce chapitre, nous explorons le contexte et les exigences du projet de développement de Mini-CI/CD, réalisé dans le cadre de notre formation en ingénierie informatique. Ce projet se concentre sur la création d'une plateforme d'intégration continue légère et efficace, destinée à automatiser les processus de build pour les projets Java Maven, tout en résolvant les problèmes courants rencontrés dans ce domaine.

Pour atteindre les objectifs du projet, nous avons défini les besoins en deux grandes catégories :

1. **Besoins fonctionnels** : Ces besoins concernent les fonctionnalités spécifiques que l'application doit offrir, comme la gestion des projets, l'exécution des builds dans des environnements isolés, et la visualisation des résultats par statut. Ces exigences constituent le cœur des fonctionnalités de la plateforme.

2. **Besoins non-fonctionnels** : Ces besoins englobent les contraintes et critères de performance essentiels, comme la fiabilité des builds, l'isolation des environnements, la sécurité, et l'ergonomie de l'interface utilisateur.

Ce chapitre explique comment ces besoins ont été identifiés, analysés, et intégrés dans le cahier des charges, ainsi que les solutions techniques mises en œuvre pour y répondre.

Spécification des besoins fonctionnels

Les besoins fonctionnels définissent les fonctionnalités essentielles de Mini-CI/CD, regroupés en besoins globaux avec des sous-besoins spécifiques.

Besoin fonctionnel 1 : Gestion des destinations et attractions

Ce besoin concerne la gestion des informations sur les projets à construire et leurs configurations.

Sous-besoin 1 : Gestion CRUD des projets

- Enregistrement des informations (nom, URL du dépôt Git, branche).
- Affichage de la liste des projets avec leurs statuts.
- Modification et suppression des projets existants .

Sous-besoin 2 : Gestion **des dépôts Git**

- Support des URLs HTTPS et Git pour les dépôts.
- Conversion automatique des URLs HTTPS en URLs Git pour contourner les problèmes d'authentification.
- Support des projets de démonstration sans dépôt Git externe.

Besoin fonctionnel 2 : Exécution des builds

Ce besoin global concerne l'exécution des builds dans des environnements isolés et la gestion du processus de build.

Sous-besoin 1 : Déclenchement des builds

- Bouton pour déclencher manuellement un build pour un projet.
- Affichage du statut en temps réel du build (PENDING, RUNNING, SUCCESS, FAILURE).
- Redirection vers la page de détails du build après déclenchement.

Sous-besoin 2 : Isolation des environnements de build

- Utilisation de conteneurs Docker pour isoler chaque build.
- Création d'un répertoire de travail unique pour chaque build.
- Nettoyage efficace des répertoires de travail entre les builds.

2.3.Sous-besoin 2 : **Gestion des logs**

- Capture et affichage des logs standard et d'erreur du processus de build .
- *Formatage des logs pour une meilleure lisibilité.*
- *Horodatage des événements importants du build.*

Besoin fonctionnel 3: Visualisation des resultats

Ce besoin concerne la présentation claire et organisée des résultats des builds.

- **Sous-besoin 1 : Séparation des builds par statut**
  - Affichage séparé des builds réussis (SUCCESS) et échoués (FAILURE).
  - Pages dédiées pour chaque type de build (/builds/success et /builds/failure).
  - Menu déroulant dans la navigation pour accéder directement aux différents types de builds.
- **Sous-besoin 2 : Tableau de bord**
  - Vue d'ensemble des builds récents par statut.
  - Statistiques sur les builds (nombre de réussites/échecs).
  - Liens rapides vers les projets et leurs builds.

Spécification des besoins non-fonctionnels

Les besoins non-fonctionnels améliorent la qualité globale de Mini-CI/CD.

- **Performance et fiabilité**:
  - Exécution asynchrone des builds pour ne pas bloquer l'interface utilisateur.
  - Isolation complète des environnements de build pour éviter les interférences.
  - Gestion efficace des ressources Docker pour optimiser les performances.
- **Sécurité** :
  - Isolation des builds dans des conteneurs Docker pour éviter les problèmes de sécurité).
  - Limitation des privilèges des conteneurs de build.
  - Nettoyage des ressources après chaque build.
- **Maintenabilité**:
  - Architecture en couches (contrôleurs, services, repositories).
  - Séparation claire des responsabilités.
- **Ergonomie et expérience utilisateur** :
  - Interface utilisateur intuitive avec Spring Boot et Thymeleaf.
  - Navigation claire entre les différentes sections de l'application.
  - Affichage explicite des statuts de build avec codes couleur.
  - Feedback immédiat lors du déclenchement d'un build .

Présentation des cas d'utilisation

Présentation des acteurs

1. **Développeur :**

- **Description : Utilisateur principal qui crée des projets, déclenche des builds et consulte les résultats.**
- **Objectifs :**
    - Configurer des projets avec leurs dépôts Git.
    - Déclencher des builds manuellement.
    - Consulter les résultats des builds et leurs logs.
    - Identifier rapidement les builds réussis et échoués.

2. **Administrateur :**

- **Description :** Gère la configuration globale du système et surveille son fonctionnement.
- **Objectifs :**
    - Configurer les paramètres Docker.
    - Gérer les ressources système.
    - Surveiller les performances de la plateforme.

Description des cas d’utilisation

Le diagramme de cas d'utilisation définit l'interopérabilité entre le système et les acteurs, soit toutes les fonctionnalités devant être fournies par le système. Ci-dessous, nous présentons les principaux cas d'utilisation:

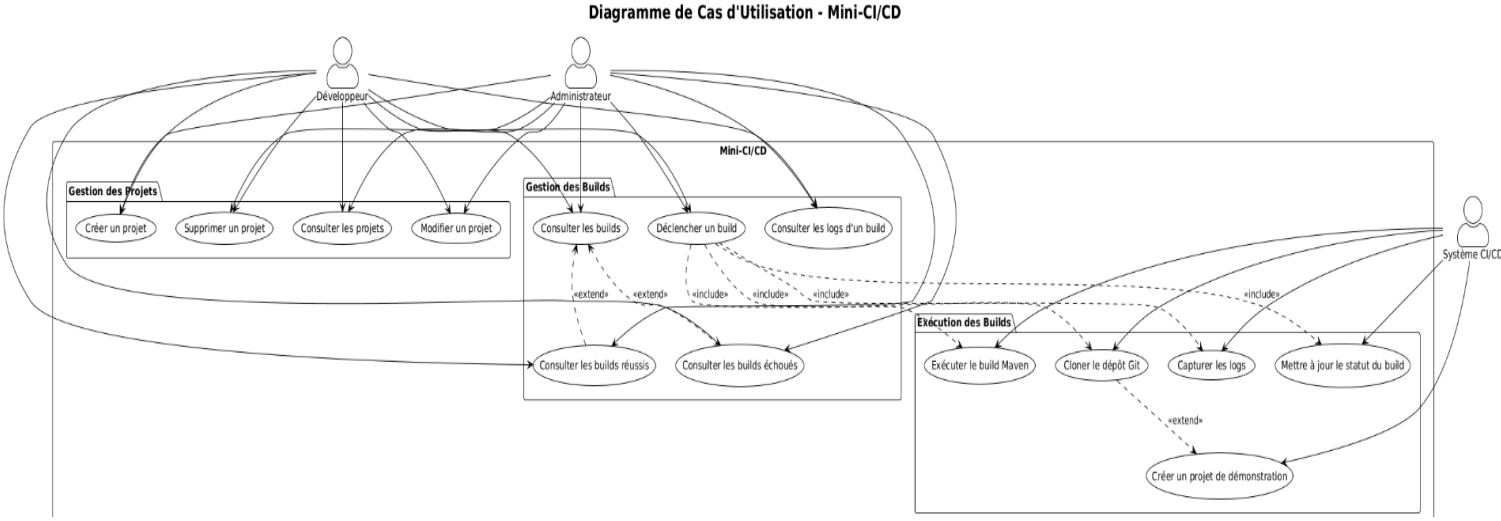


Figure 2 : diagramme de cas d’utilisation

Acteur(s) :	*Utilisateur*
Objectif :	Déclencher un build pour un projet spécifique afin de vérifier sa compilation et ses test.
Postcondition(s) :	Le projet est configuré avec une URL de dépôt Git valide et une branche .
Scénario nominal :	• Le développeur accède à la page du projet. • Il clique sur le bouton "Build". • Le système crée un nouvel enregistrement de build avec le statut PENDING. • Le système lance le processus de build dans un conteneur Docker isolé. • Le système capture les logs et détermine le statut final (SUCCESS ou FAILURE). • Le développeur est redirigé vers la page de détails du build.
Scénario alternatif :	• **Erreur de disponibilité** Si l'URL du dépôt ou la branche sont invalides, afficher un message d'erreur. • **Erreur Docker**: Si les informations sont invalides (par exemple, format de date incorrect), afficher un message d’erreur et inviter à corriger.

Tableau 1 : Fonctionnalités de l'Utilisateur avec travelapp

Chapitre 3

Conception du système

Introduction

Dans ce chapitre, nous allons modéliser notre application Mini-CI/CD à la fois d'un point de vue statique et dynamique. Pour la modélisation dynamique, nous utiliserons des diagrammes de séquence qui illustreront le processus de build et les interactions entre les différents composants du système. Pour modéliser l'aspect statique, nous présenterons un diagramme de classes détaillant la structure de notre application.

Modélisation dynamique

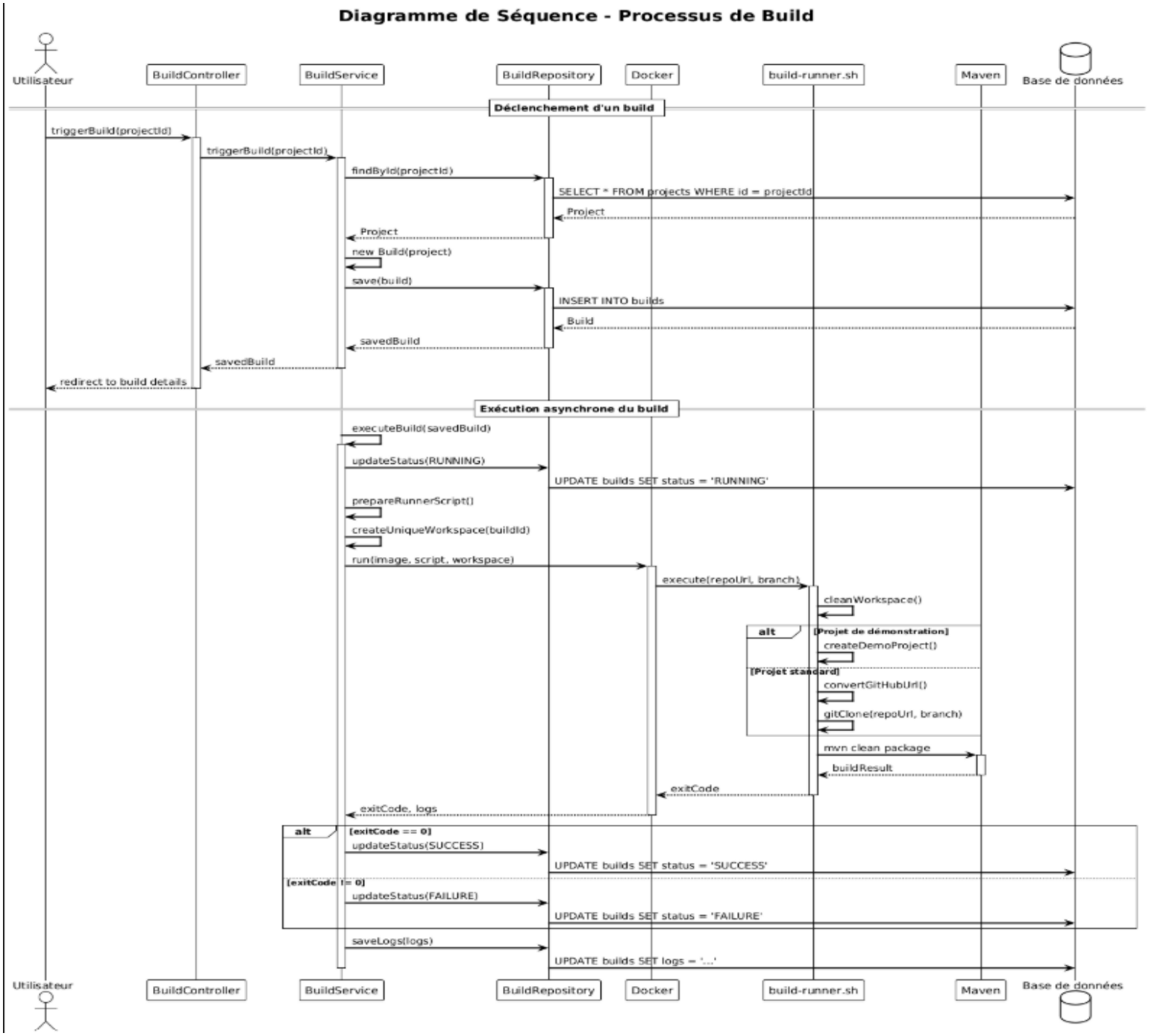


Figure 3 : diagramme séquence

Ce diagramme illustre les étapes suivantes :

1. **Déclenchement d'un build** :
    - L'utilisateur initie le processus en appelant la méthode triggerBuild du contrôleur avec l'ID du projet.
    - Le contrôleur délègue cette demande au service de build.
    - Le service récupère le projet correspondant, crée un nouvel objet Build avec le statut PENDING, et le sauvegarde en base de données.
    - L'utilisateur est redirigé vers la page de détails du build.
2. **Exécution asynchrone du build** :
    - Le service exécute le build de manière asynchrone avec CompletableFuture .
    - Le statut du build est mis à jour à RUNNING.
    - Un script runner est préparé et un espace de travail unique est créé pour éviter les conflits.
    - Un conteneur Docker est lancé avec le script, l'espace de travail monté, et les paramètres du projet.
3. **Exécution du script dans Docker** :
    - Le script nettoie d'abord l'espace de travail pour éviter les problèmes de fichiers résiduels.
    - Selon l'URL du projet, il crée soit un projet de démonstration, soit clone le dépôt Git.
    - Pour les dépôts GitHub, les URLs HTTPS sont converties en URLs Git pour éviter les problèmes d'authentification.
    - Maven est exécuté pour construire le projet.
4. **Finalisation du build** :
    - Le code de sortie du processus détermine le statut final du build (SUCCESS ou FAILURE).
    - Les logs sont capturés et sauvegardés dans l'objet Build.
    - Le statut final et les logs sont enregistrés en base de données.

Ce diagramme met en évidence les améliorations apportées au système, notamment :

- L'utilisation d'un espace de travail unique pour chaque build
- Le nettoyage efficace de l'espace de travail
- La conversion des URLs HTTPS en URLs Git
- La gestion des projets de démonstration

Modélisation statique

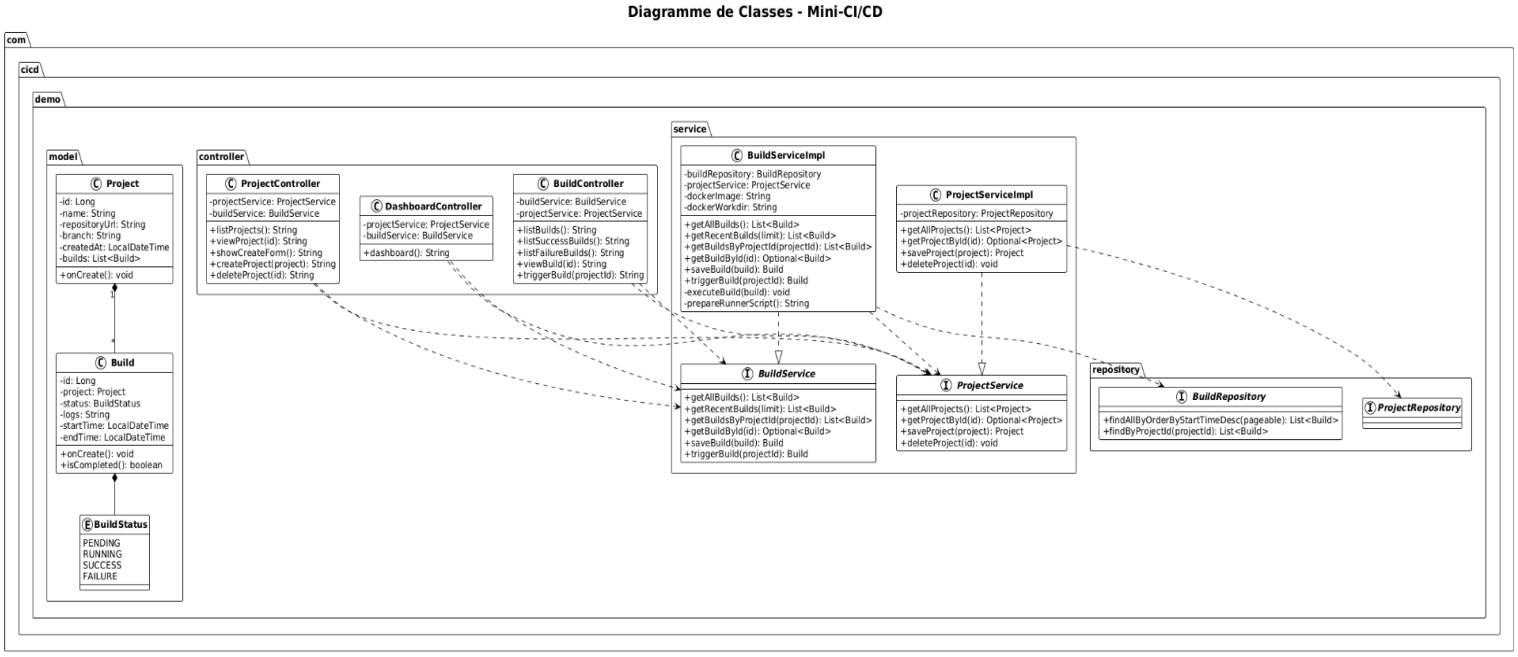


Figure 4 : diagramme de classe

Ce diagramme est organisé en quatre packages principaux :

1. **com.cicd.demo.model** :
    - **Project** : Représente un projet avec ses attributs (nom, URL du dépôt, branche) et une relation one-to-many avec Build.
    - **Build** : Représente un build avec son statut, ses logs, et ses horodatages.
    - **BuildStatus** : Énumération des statuts possibles (PENDING, RUNNING, SUCCESS, FAILURE).
2. **com.cicd.demo.controller** :
    - **BuildController** : Gère les requêtes liées aux builds, avec des méthodes spécifiques pour afficher les builds réussis et échoués séparément.
    - **ProjectController** : Gère les requêtes liées aux projets.
    - **DashboardController** : Gère l'affichage du tableau de bord.
3. **com.cicd.demo.service** :
    - **BuildService** (interface) et **BuildServiceImpl** : Définissent et implémentent les opérations liées aux builds, notamment le déclenchement et l'exécution des builds.
    - **ProjectService** (interface) et **ProjectServiceImpl** : Définissent et implémentent les opérations liées aux projets.
4. **com.cicd.demo.repository** :
    - **BuildRepository** et **ProjectRepository** : Interfaces pour l'accès aux données des builds et des projets.

Ce diagramme met en évidence l'architecture en couches de l'application (contrôleurs, services, repositories) et les relations entre les différentes classes. La séparation claire des responsabilités facilite la maintenance et l'évolution du système

Conclusion

Dans ce chapitre, nous avons modélisé notre application Mini-CI/CD en utilisant deux types de diagrammes UML : le diagramme de séquence et le diagramme de classes.
Le diagramme de séquence a permis de visualiser les interactions chronologiques entre les différents acteurs et les composants de notre système, montrant comment les messages sont échangés pour réaliser le processus de build. Cette modélisation dynamique nous a aidés à comprendre le flux de contrôle et à identifier les points d'amélioration pour résoudre les problèmes rencontrés, notamment :

- La création d'espaces de travail uniques pour chaque build
- Le nettoyage efficace des répertoires
- La conversion des URLs GitHub pour faciliter l'accès aux dépôts

Le diagramme de classes a fourni une vue statique de la structure interne de l'application, en détaillant les classes principales, leurs attributs, méthodes et les relations qui les lient. Ce diagramme a été essentiel pour identifier les différentes entités du système et leurs interactions structurelles, facilitant ainsi la conception et l'implémentation de l'application.
Ces deux types de diagrammes ont fourni une compréhension claire et détaillée des aspects dynamiques et statiques de notre application, préparant ainsi le terrain pour une implémentation cohérente et efficace qui répond aux besoins identifiés dans le chapitre précédent.

Chapitre 4

Réalisation du système

1. Introduction

Dans ce chapitre, nous allons présenter l'environnement matériel et logiciel adopté pour le développement de l'application, ainsi que les principales interfaces graphiques. Ce chapitre se compose de deux parties : la première partie détaillera l'environnement de développement utilisé tout au long du projet, tandis que la deuxième partie se concentrera sur la mise en œuvre de la solution proposée en décrivant les interfaces et les principales fonctionnalités implémentées.

2. Environnement de développement

2.1. Environnement matériel

Nous avons développé notre application en utilisant l'équipement suivant :

- **Processeur** : Intel Core i7 11e génération, 2,8 GHz
- **Mémoire centrale (RAM)** : 16 Go DDR4
- **Stockage** : 512 Go SSD
- **Carte graphique** : Intel® UHD GRAPHICS 620
- **Système d'exploitation** : Windows 11

2.2. Environnement logiciel

Nous avons utilisé les outils suivants pour le développement :

- **IntelliJ IDEA** :

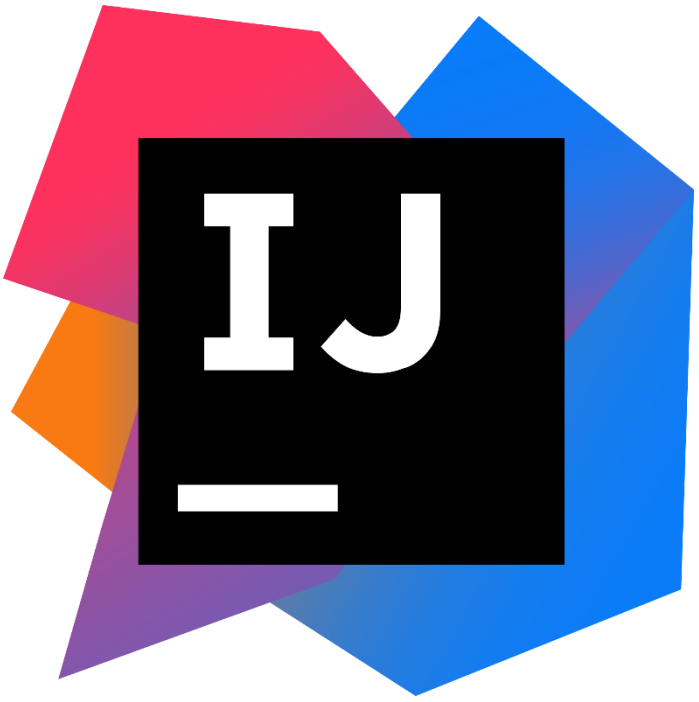


Figure 5 : logo de IntelliJ IDEA

Un IDE de JetBrains pour Java/Spring Boot, avec autocomplétion, débogage, refactorisation, et intégration Maven/Git. Idéal pour les applications web modernes.

- **Langage de programmation** : Java 17,Thymeleaf,HTML,JavaScript

Figure 6 : Logo de Java

Langage orienté objet, version LTS, utilisé pour la logique métier via Spring Boot.

Figure 7 : Logo de HTML

HTML (HyperText Markup Language) : HTML est le langage standard pour structurer et présenter le contenu sur le web. Il permet de définir les éléments d'une page web (titres, paragraphes, liens, images, etc.). C’est la base de la création de pages web statiques.

- **Service d’intelligence artificielle**

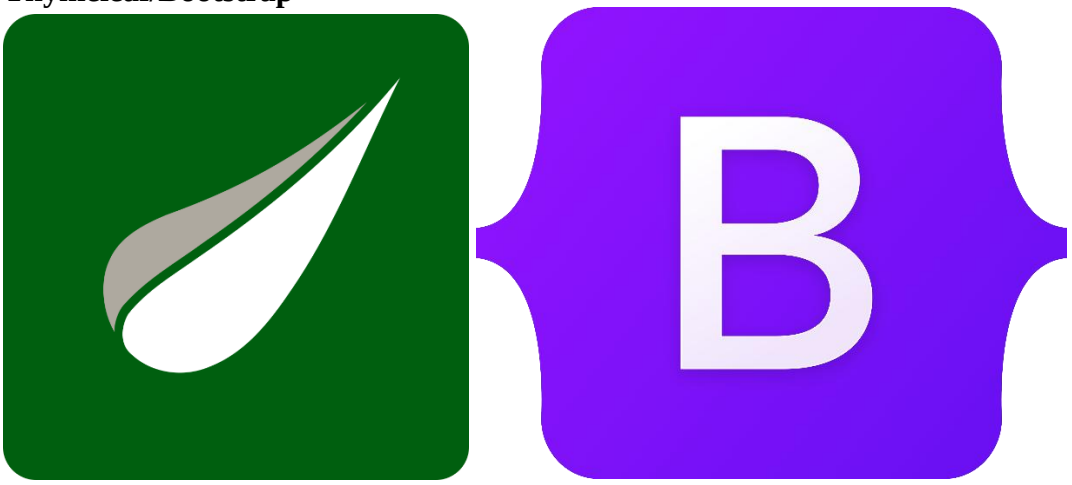
PostgreSQL est le système de gestion de base de données relationnelle utilisé dans notre application Mini-CI/CD. Il s'agit d'une base de données open-source robuste, conforme aux normes SQL et reconnue pour sa fiabilité, son intégrité des données et sa capacité à gérer des charges de travail complexes.

- **Framework** :

Figure 8 : Logo de springboot

Simplifie le développement web avec configuration automatique, API REST, et intégration Spring Security/JPA. Gère le back-end de TravelApp.

- **Thymeleaf/Bootstrap**



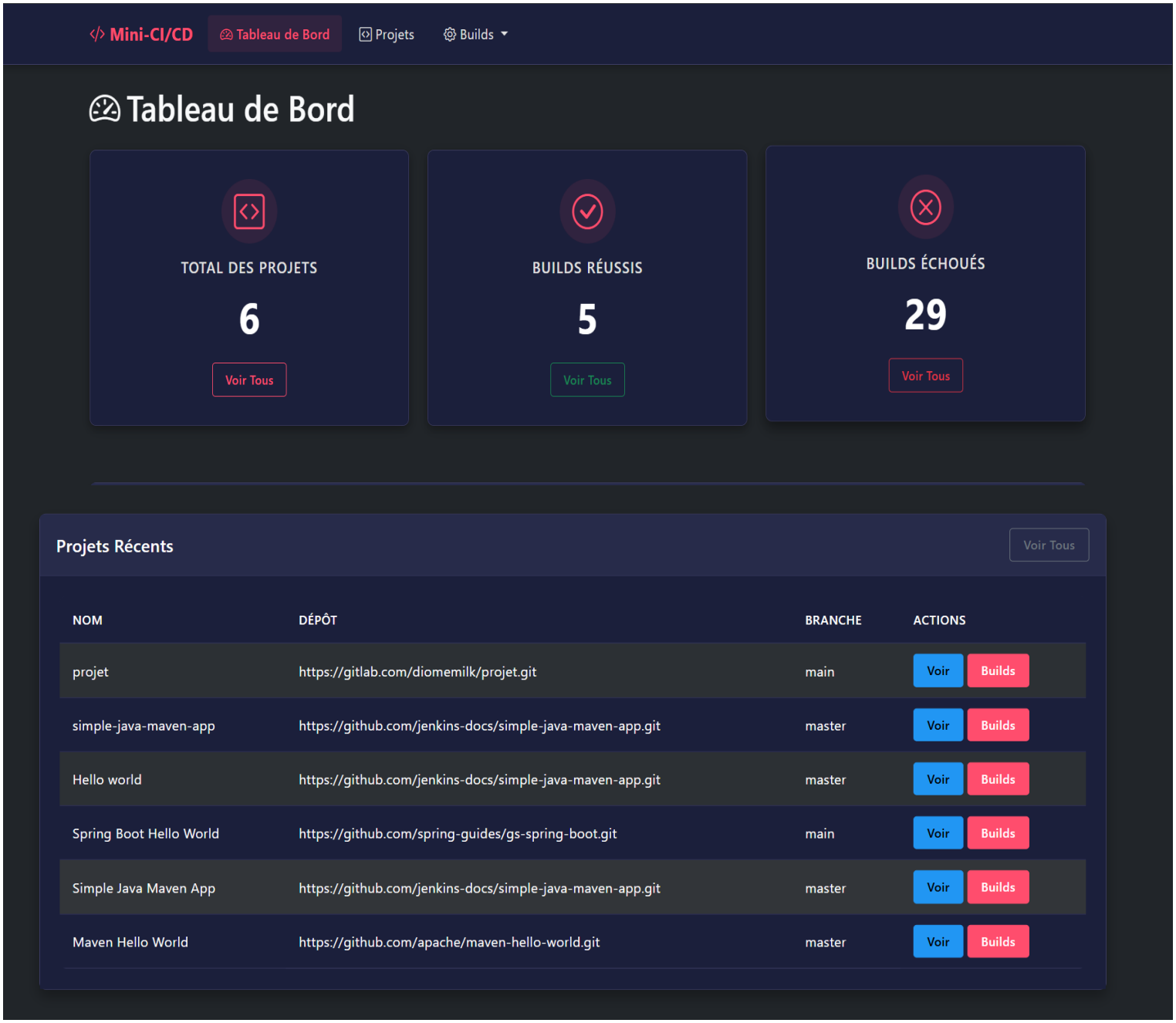
Thymeleaf rend les vues dynamiques, Bootstrap offre un design responsive.

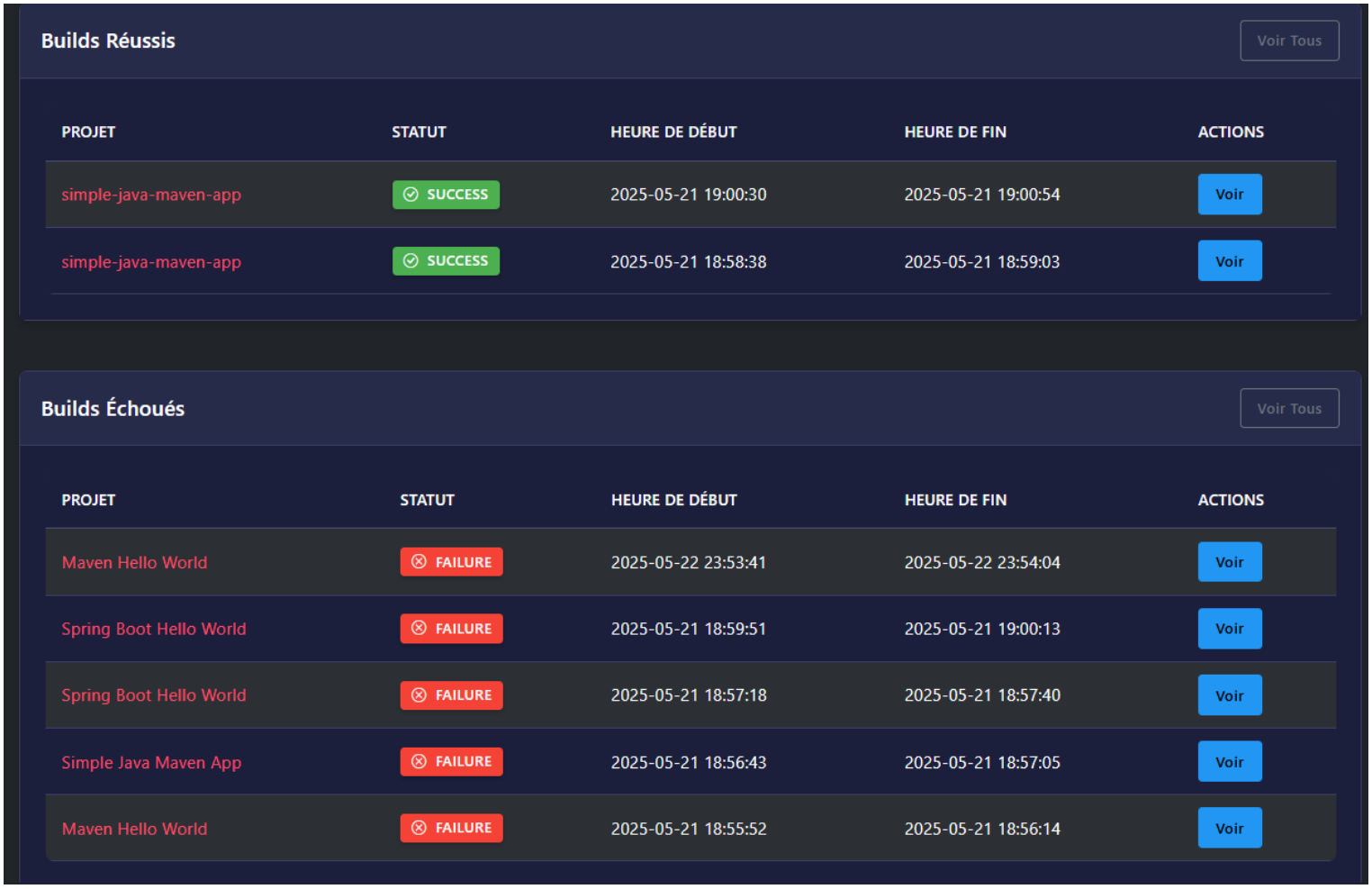
- **Outils de versionnement** :

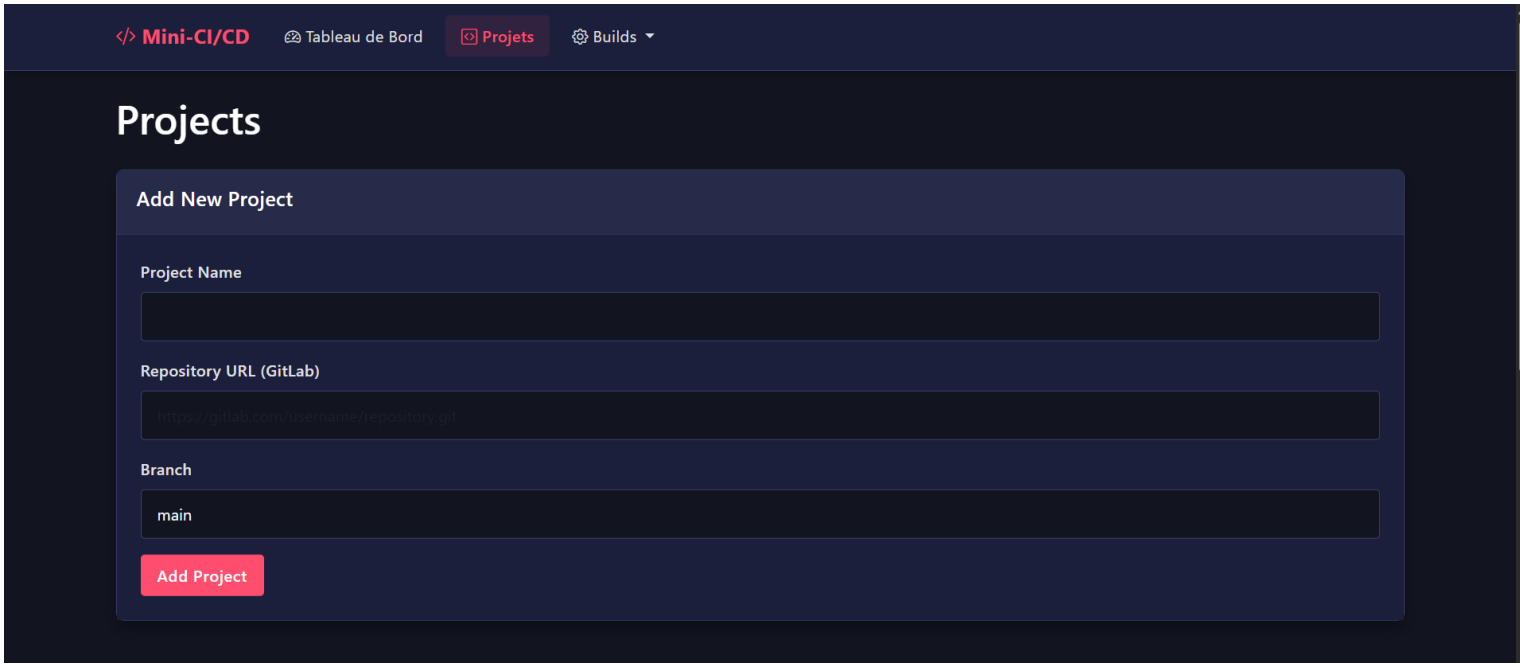
GitHub : GitHub est une plateforme d'hébergement pour les projets utilisant Git. Elle offre une interface conviviale pour héberger des dépôts de code, collaborer avec d'autres développeurs, suivre les problèmes et gérer les demandes de modification de code (pull requests). C’est aussi une plateforme sociale pour les développeurs, favorisant le partage de code et la collaboration open source

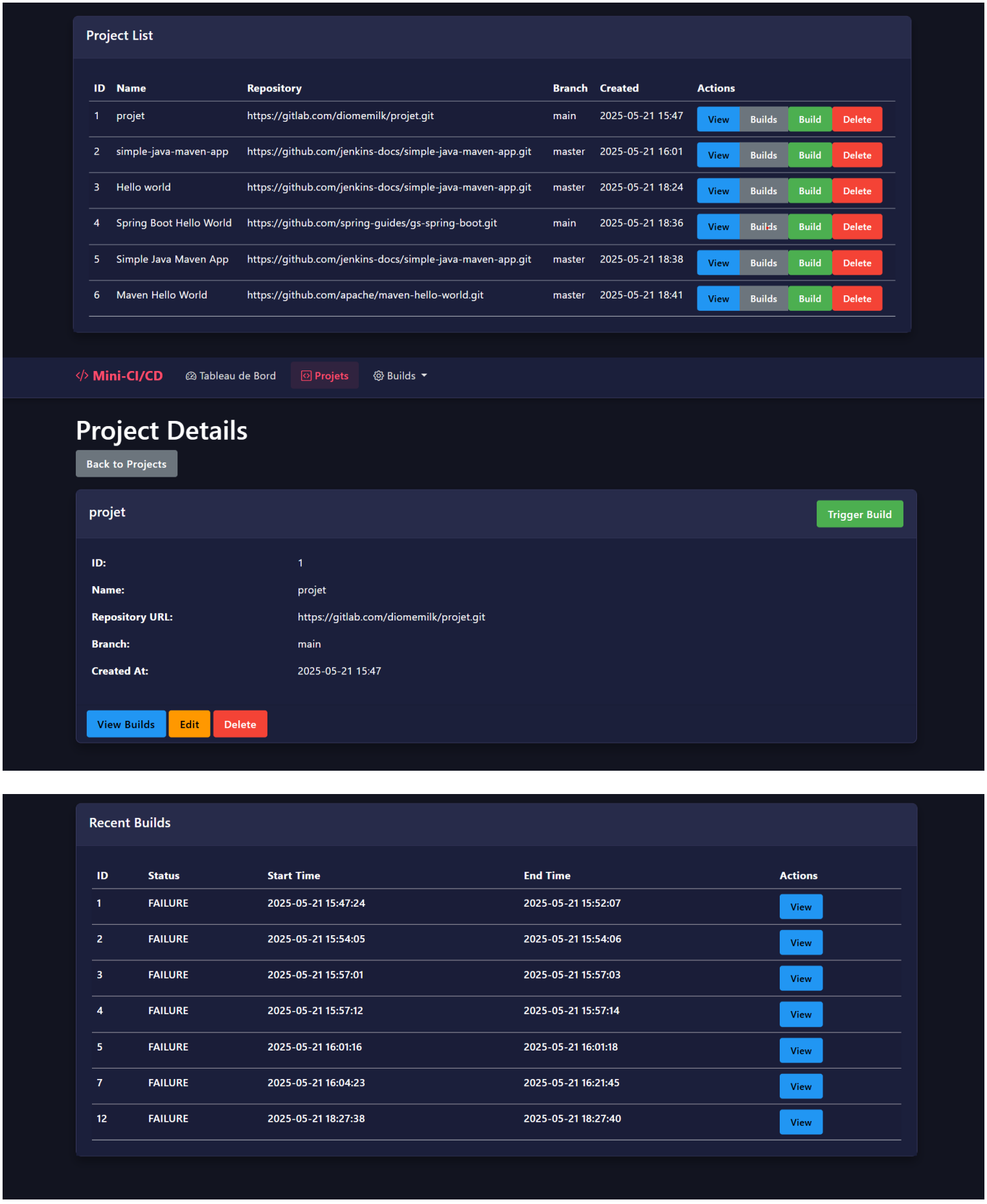
Cet environnement a permis de mettre en place un cadre de développement adéquat pour le bon déroulement de notre projet.

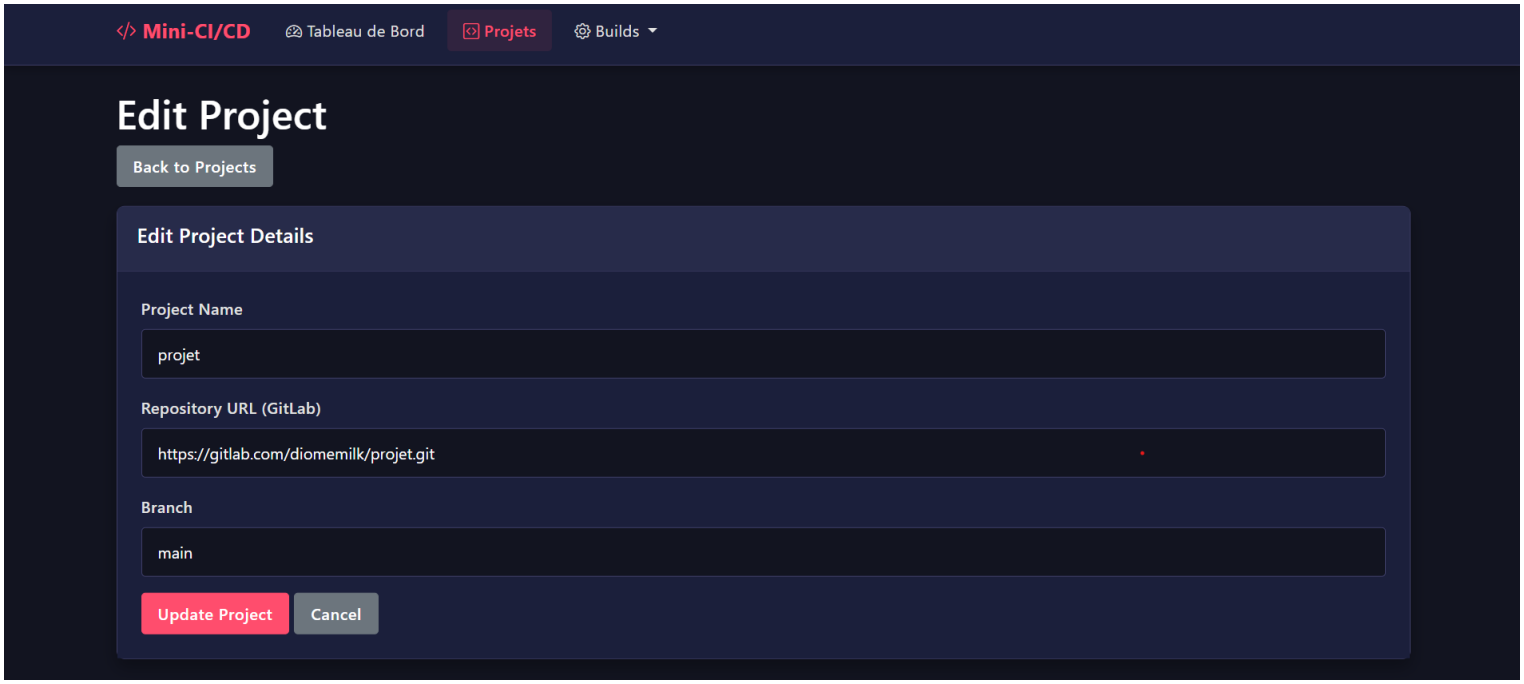
3. Principales interfaces graphiques

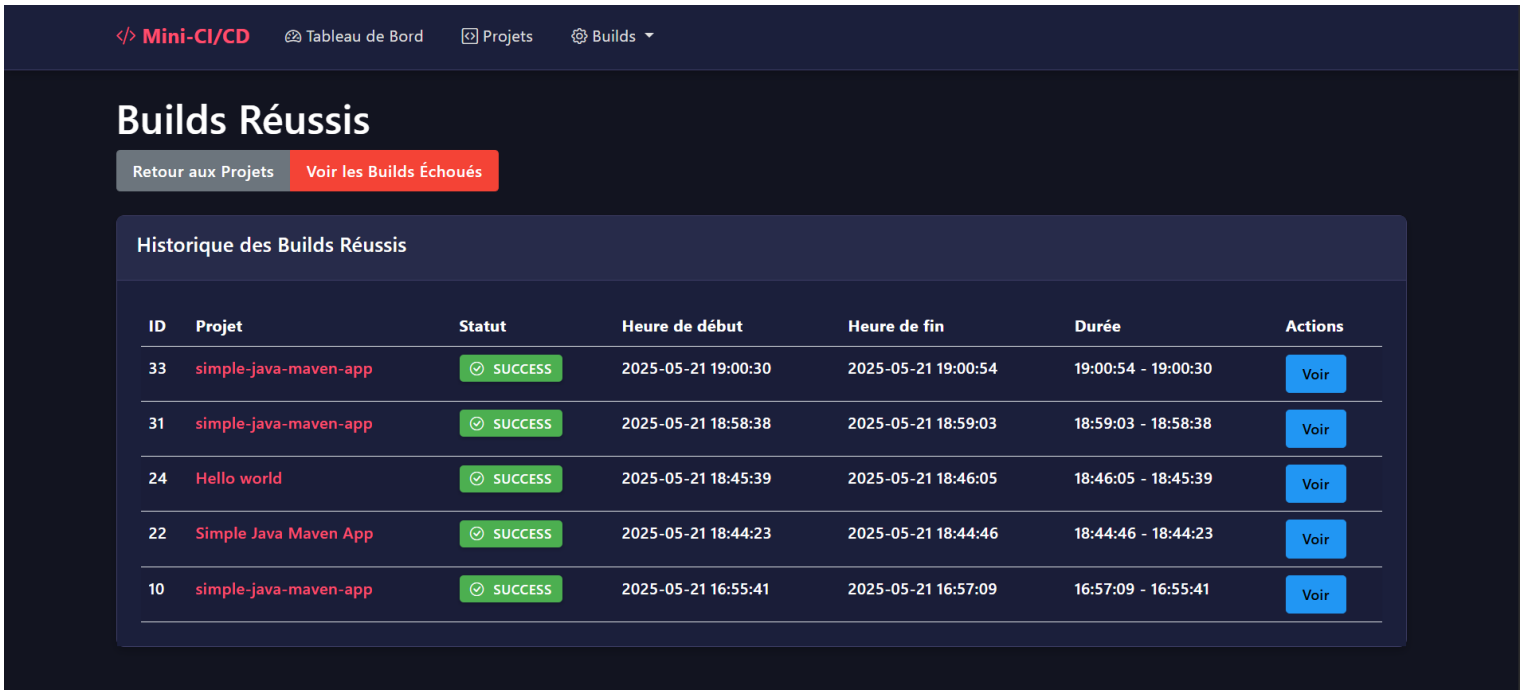


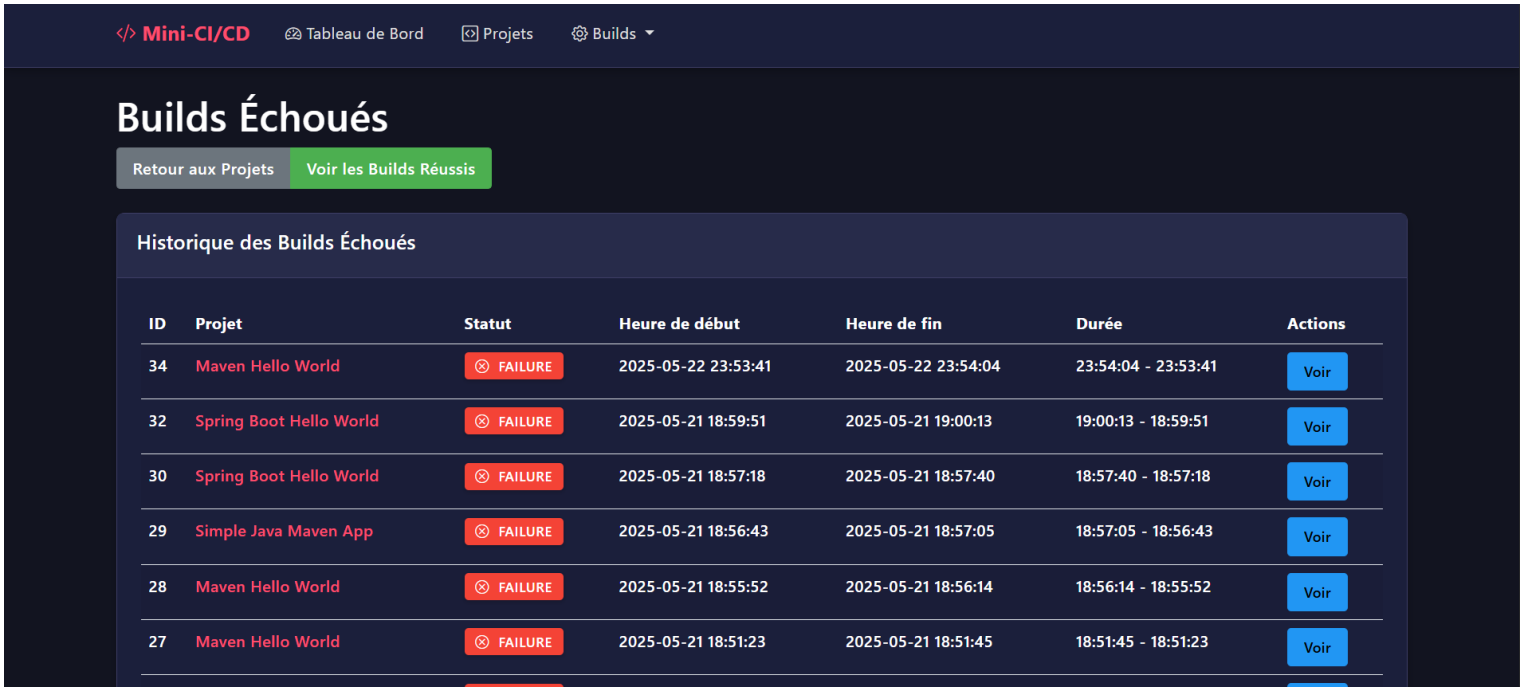

Builds Réussis
Voir Tous
PROJET
STATUT
HEURE DE DÉBUT
HEURE DE FIN
ACTIONS
simple-java-maven-app
SUCCESS
2025-05-21 19:00:30
2025-05-21 19:00:54
Voir
simple-java-maven-app
SUCCESS
2025-05-21 18:58:38
2025-05-21 18:59:03
Voir
Builds Échoués
Voir Tous
PROJET
STATUT
HEURE DE DÉBUT
HEURE DE FIN
ACTIONS
Maven Hello World
FAILURE
2025-05-22 23:53:41
2025-05-22 23:54:04
Voir
Spring Boot Hello World
FAILURE
2025-05-21 18:59:51
2025-05-21 19:00:13
Voir
Spring Boot Hello World
FAILURE
2025-05-21 18:57:18
2025-05-21 18:57:40
Voir
Simple Java Maven App
FAILURE
2025-05-21 18:56:43
2025-05-21 18:57:05
Voir
Maven Hello World
FAILURE
2025-05-21 18:55:52
2025-05-21 18:56:14
Voir

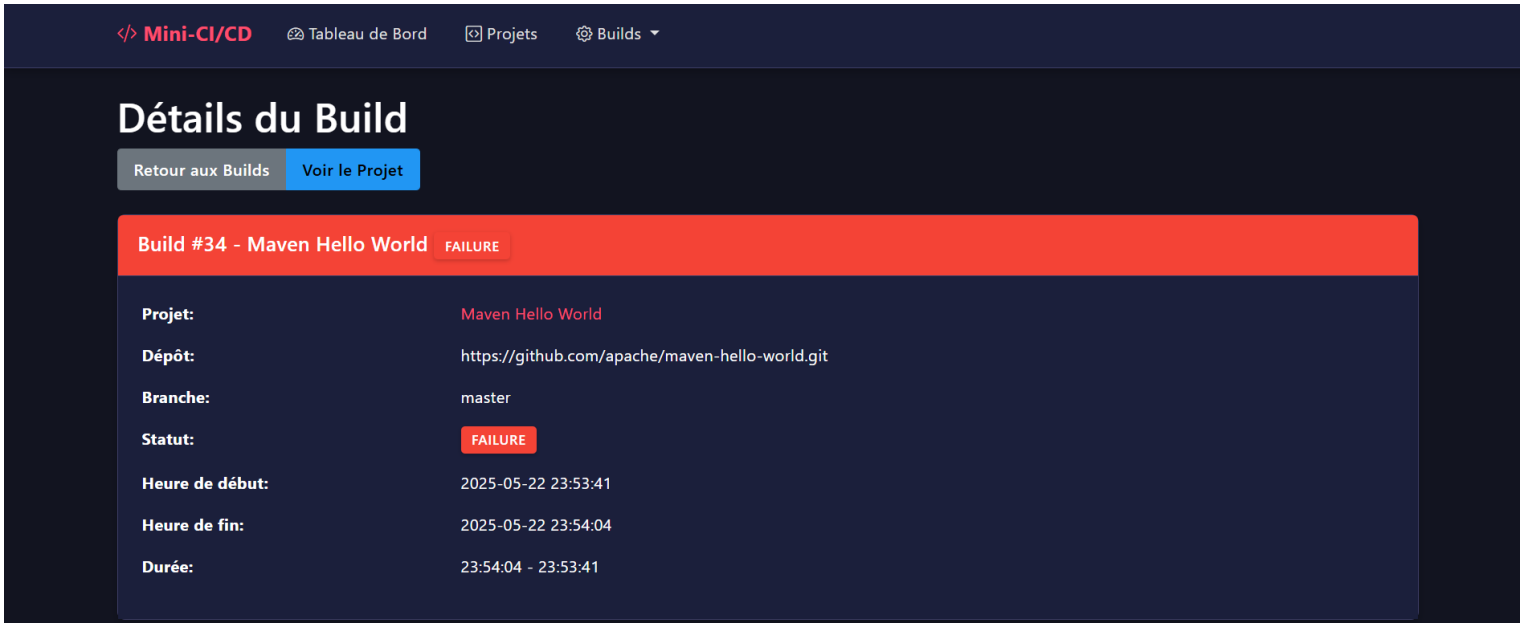

Mini-CI/CD
Tableau de Bord
Projets
Builds
Projects
Add New Project
Project Name
Repository URL (GitLab)
https://gitlab.com/username/repository.git
Branch
main
Add Project

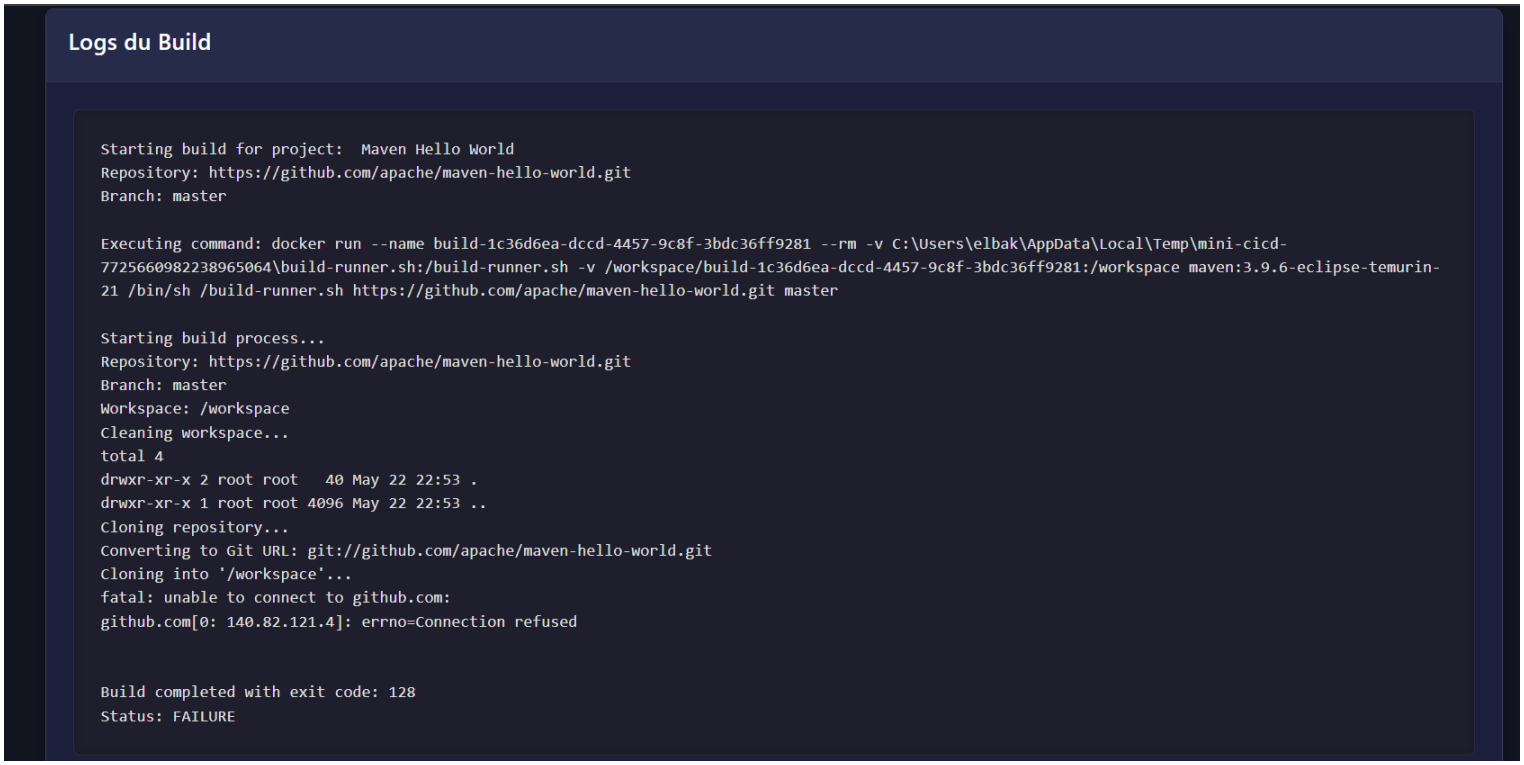

Project List
ID Name Repository Branch Created Actions
1 projet https://gitlab.com/diomemilk/projet.git main 2025-05-21 15:47 View Builds Build Delete
2 simple-java-maven-app https://github.com/jenkins-docs/simple-java-maven-app.git master 2025-05-21 16:01 View Builds Build Delete
3 Hello world https://github.com/jenkins-docs/simple-java-maven-app.git master 2025-05-21 18:24 View Builds Build Delete
4 Spring Boot Hello World https://github.com/spring-guides/gs-spring-boot.git main 2025-05-21 18:36 View Builds Build Delete
5 Simple Java Maven App https://github.com/jenkins-docs/simple-java-maven-app.git master 2025-05-21 18:38 View Builds Build Delete
6 Maven Hello World https://github.com/apache/maven-hello-world.git master 2025-05-21 18:41 View Builds Build Delete
Mini-CI/CD
Tableau de Bord
Projets
Builds
Project Details
Back to Projects
projet
Trigger Build
ID: 1
Name: projet
Repository URL: https://gitlab.com/diomemilk/projet.git
Branch: main
Created At: 2025-05-21 15:47
View Builds
Edit
Delete
Recent Builds
ID Status Start Time End Time Actions
1 FAILURE 2025-05-21 15:47:24 2025-05-21 15:52:07 View
2 FAILURE 2025-05-21 15:54:05 2025-05-21 15:54:06 View
3 FAILURE 2025-05-21 15:57:01 2025-05-21 15:57:03 View
4 FAILURE 2025-05-21 15:57:12 2025-05-21 15:57:14 View
5 FAILURE 2025-05-21 16:01:16 2025-05-21 16:01:18 View
7 FAILURE 2025-05-21 16:04:23 2025-05-21 16:21:45 View
12 FAILURE 2025-05-21 18:27:38 2025-05-21 18:27:40 View


Mini-CI/CD
Tableau de Bord
Projets
Builds
Edit Project
Back to Projects
Edit Project Details
Project Name
projet
Repository URL (GitLab)
https://gitlab.com/diomemilk/projet.git
Branch
main
Update Project
Cancel


Mini-CI/CD
Tableau de Bord
Projets
Builds
Builds Réussis
Retour aux Projets
Voir les Builds Échoués
Historique des Builds Réussis
ID Projet Statut Heure de début Heure de fin Durée Actions
33 simple-java-maven-app SUCCESS 2025-05-21 19:00:30 2025-05-21 19:00:54 19:00:54 - 19:00:30 Voir
31 simple-java-maven-app SUCCESS 2025-05-21 18:58:38 2025-05-21 18:59:03 18:59:03 - 18:58:38 Voir
24 Hello world SUCCESS 2025-05-21 18:45:39 2025-05-21 18:46:05 18:46:05 - 18:45:39 Voir
22 Simple Java Maven App SUCCESS 2025-05-21 18:44:23 2025-05-21 18:44:46 18:44:46 - 18:44:23 Voir
10 simple-java-maven-app SUCCESS 2025-05-21 16:55:41 2025-05-21 16:57:09 16:57:09 - 16:55:41 Voir


Mini-CI/CD
Tableau de Bord
Projets
Builds
Builds Échoués
Retour aux Projets
Voir les Builds Réussis
Historique des Builds Échoués
ID Projet Statut Heure de début Heure de fin Durée Actions
34 Maven Hello World FAILURE 2025-05-22 23:53:41 2025-05-22 23:54:04 23:54:04 - 23:53:41 Voir
32 Spring Boot Hello World FAILURE 2025-05-21 18:59:51 2025-05-21 19:00:13 19:00:13 - 18:59:51 Voir
30 Spring Boot Hello World FAILURE 2025-05-21 18:57:18 2025-05-21 18:57:40 18:57:40 - 18:57:18 Voir
29 Simple Java Maven App FAILURE 2025-05-21 18:56:43 2025-05-21 18:57:05 18:57:05 - 18:56:43 Voir
28 Maven Hello World FAILURE 2025-05-21 18:55:52 2025-05-21 18:56:14 18:56:14 - 18:55:52 Voir
27 Maven Hello World FAILURE 2025-05-21 18:51:23 2025-05-21 18:51:45 18:51:45 - 18:51:23 Voir


Mini-CI/CD
Tableau de Bord
Projets
Builds
Détails du Build
Retour aux Builds
Voir le Projet
Build #34 - Maven Hello World FAILURE
Projet: Maven Hello World
Dépôt: https://github.com/apache/maven-hello-world.git
Branche: master
Statut: FAILURE
Heure de début: 2025-05-22 23:53:41
Heure de fin: 2025-05-22 23:54:04
Durée: 23:54:04 - 23:53:41



4.Conclusion

Dans ce chapitre, nous avons présenté les principales interfaces graphiques de Mini-CI/CD, en mettant l'accent sur les fonctionnalités clés qui facilitent l'interaction avec le système. Du tableau de bord offrant une vue d'ensemble des builds récents à la page détaillée des logs de build, en passant par le formulaire de création de projet et les pages dédiées aux builds réussis et échoués, chaque interface a été conçue pour offrir une expérience développeur simple et efficace.

Ces interfaces permettent non seulement de configurer et gérer les projets, déclencher des builds et visualiser leurs résultats, mais aussi d'analyser rapidement les problèmes grâce à une séparation claire des builds par statut. Cela met en évidence l'importance d'une interface intuitive dans la gestion des processus d'intégration continue, tout en assurant une continuité logique entre la configuration des projets, l'exécution des builds, et l'analyse des résultats dans un environnement web moderne.

Les améliorations apportées à l'interface utilisateur, notamment la séparation des builds par statut (SUCCESS/FAILURE) et l'affichage détaillé des logs, facilitent considérablement l'identification et la résolution des problèmes. Cette approche centrée sur l'utilisateur permet aux développeurs de se concentrer sur la qualité de leur code plutôt que sur la gestion complexe des outils d'intégration continue.

La conception modulaire et responsive de l'interface assure également une expérience cohérente sur différents appareils, permettant aux développeurs de surveiller leurs builds même en situation de mobilité. Cette flexibilité, combinée à la simplicité d'utilisation, fait de Mini-CI/CD un outil précieux pour les équipes de développement cherchant à automatiser efficacement leurs processus de build sans la complexité des solutions d'entreprise plus lourdes.