Université de Technologie de Compiègne

Ingénieur UTC - Branche Informatique
Ingénierie des systèmes informatiques

Projet de fin d'études TN10

Rapport de stage

Stage Ingénieur Développement
Automatisation des tests logiciels

Stagiaire :
Yue LI

Enseignant référent UTC :
Mehdi SERAIRI
Responsable pédagogique

Tuteur de stage :
Nadezda VUKMIROVIC
Chef de Projet Informatique

Entreprise :
Crédit Agricole Corporate
And Investment Bank

Adresse :
Place des États-Unis
92120 Montrouge

Service :
IOS

Durée :
Du 21/05/2025 au 21/11/2025

15 octobre 2025

Résumé

Ce rapport présente le travail réalisé lors de mon stage de fin d'études au sein de Crédit Agricole Corporate and Investment Bank (CA-CIB), dans le service IOS (Infrastructure and Operations Services). Ce stage de six mois, effectué du 21 mai au 21 novembre 2025, s'inscrit dans le cadre de ma formation d'ingénieur en informatique, spécialité Ingénierie des systèmes informatiques, à l'Université de Technologie de Compiègne.

Le stage s'est articulé autour de deux missions complémentaires sur l'application CJUST, outil de justification comptable. La première mission a consisté à développer un framework de tests automatisés end-to-end avec Cypress et Cucumber, en appliquant l'approche BDD et le pattern Page Object Model. Cette solution couvre l'ensemble des workflows métier et garantit la non-régression de l'application lors des évolutions futures.

La seconde mission a porté sur le développement full stack d'un module batch Spring Batch pour automatiser la reprojection des suspens comptables entre périodes. L'architecture en couches assure la maintenabilité du code, et le déploiement est orchestré sur Kubernetes via un CronJob quotidien avec pipeline CI/CD complet.

Table des matières

Chapitre 1

Présentation de l'entreprise

1.1 Le Groupe Crédit Agricole

Le Crédit Agricole est l'une des plus grandes institutions financières en France et en Europe. Créé en 1885, il est aujourd'hui composé d'un réseau de banques coopératives et d'entités spécialisées dans les services financiers.

Son périmètre rassemble le Crédit Agricole S.A., l'ensemble des Caisses régionales et des Caisses locales, ainsi que leurs filiales, toutes liées au groupe par des accords de coopération et de partage des ressources.

Implantée dans 46 pays à travers le monde, la gamme d'offres proposée par le Crédit Agricole couvre le financement et l'investissement, le crédit à la consommation, l'assurance, la gestion d'actifs et bien plus encore.

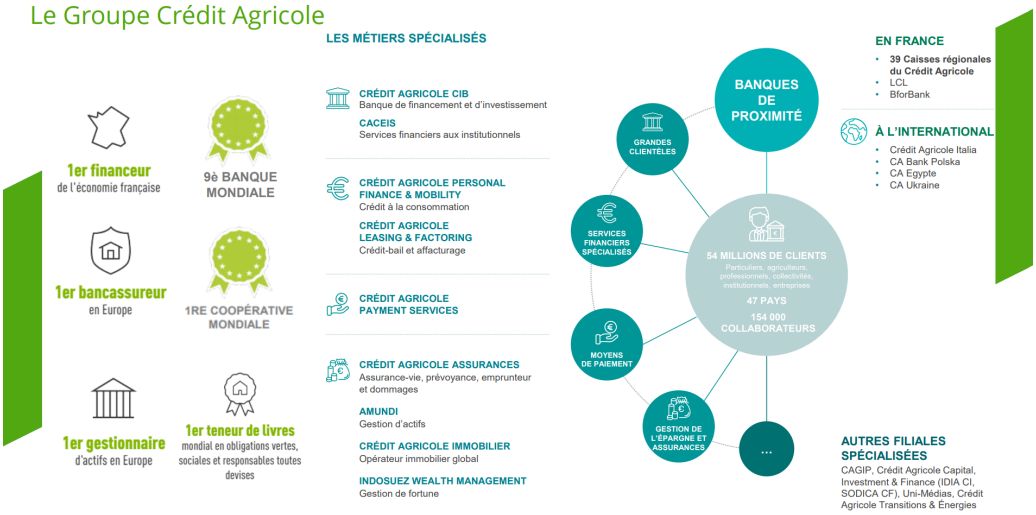

Fig. 1.1 : Le groupe Crédit Agricole

1.2 Crédit Agricole CIB

Crédit Agricole CIB (CACIB) constitue la branche de financement et d'investissement du groupe Crédit Agricole. Ses activités s'organisent autour de 6 axes stratégiques majeurs :

- Banque commerciale et Trade
- Banque d'investissement
- Financements structurés
- Banque de marchés
- Distribution & Asset Rotation
- Finance corporate et à effet de levier

Son segment de clientèle comprend les entreprises (grandes entreprises, ETI), les institutions financières et les fonds de capital-investissement et d'infrastructure. Elle leur propose une approche spécialisée sur 8 secteurs différenciants dans lesquels son expertise est reconnue, notamment l'immobilier, l'automobile ainsi que l'agri-agro.

CACIB occupe une position de leader mondial dans les activités de Finance Durable :

- Dans le Top 5 mondial des banques en Finance Durable
- Co-auteur des Green Bond Principles
- Une offre de services dédiée au reporting d'impact

Une part importante des revenus commerciaux de CACIB est réalisée à l'international : 39% à l'étranger, 35% en France et 26% dans le reste de l'Europe (données au 31 décembre 2022).

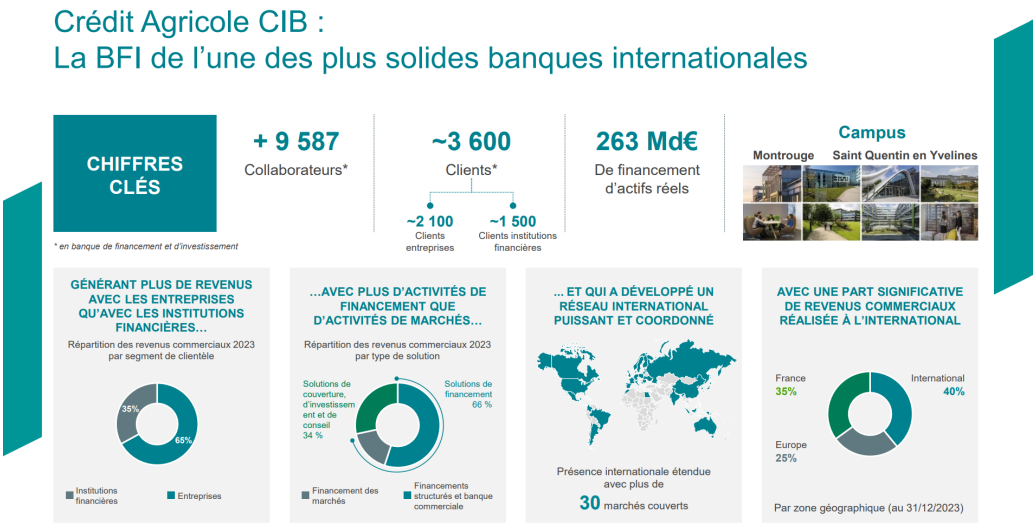

Fig. 1.2 : Crédit Agricole CIB

1.3 Le département IOS du Crédit Agricole CIB

1.3.1 Présentation générale

IOS est une division performante, digitale et durable au service du développement et de la transformation de CACIB. Cette direction développe des solutions intégrées sur mesure, tant pour la clientèle externe que pour les métiers internes de la banque.

Fig. 1.3 : Les directions IOS

1.3.2 Focus sur la direction Casa ES et IT Communautaires (ICA)

ICA est la direction responsable du système d'information CASA et de l'IT communautaire. Cette entité a été créée suite à une réorganisation au sein d'IOS, effective depuis mars 2024.

ICA rassemble diverses équipes IT chargées de gérer le système d'information de l'entité CASA ES, de trois filiales et de l'IT Communautaire. Les directions métier de CASA ES agissent en tant que donneurs d'ordre pour ces activités.

1.3.3 Périmètre de responsabilité

La direction ICA a dans son périmètre de responsabilité :

- La prise en charge de la feuille de route, de l'architecture, de la conception, de la mise en œuvre et de l'exploitation du SI CASA ES et de l'IT Communautaire, en collaboration avec CAGIP, CISO CASA et IOS/RMC pour la gestion des risques et la sécurité informatique

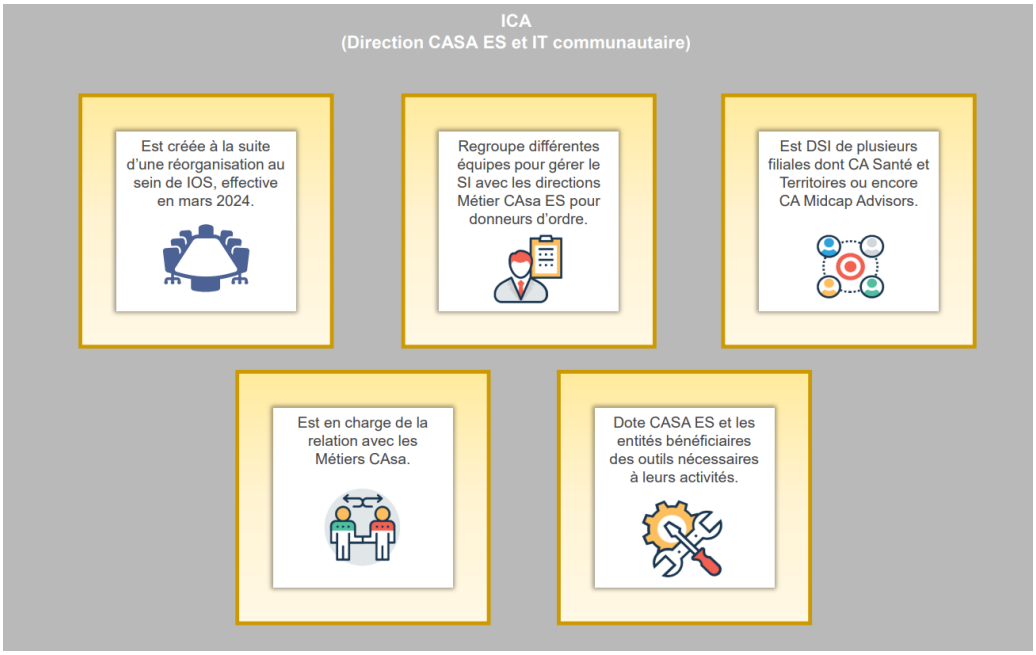

Fig. 1.4 : Présentation ICA

- La définition de la politique informatique et la coordination de sa mise en œuvre pour doter les Métiers CASA ES et les entités bénéficiaires des outils nécessaires, alignés sur leur activité, tout en assurant l'alignement avec les objectifs et l'optimisation des dépenses

Chapitre 2

Présentation du projet

2.1 Description fonctionnelle

À la suite de la mission de l'Inspection Générale, un chantier a été initié afin de renforcer le dispositif de contrôle comptable. Dans le cadre de ce chantier, il a été demandé de mettre en place un applicatif unique pour les travaux de justification des comptes et suivi des suspens anormaux.

Cette interface de travail unique permettra de regrouper dans un même outil les différents travaux réalisés par les CRC et TCC afin de sécuriser, de simplifier et d'améliorer le processus de révision des comptes et justifications de l'ensemble des soldes dont les comptes sensibles et les déclarations des suspens anormaux.

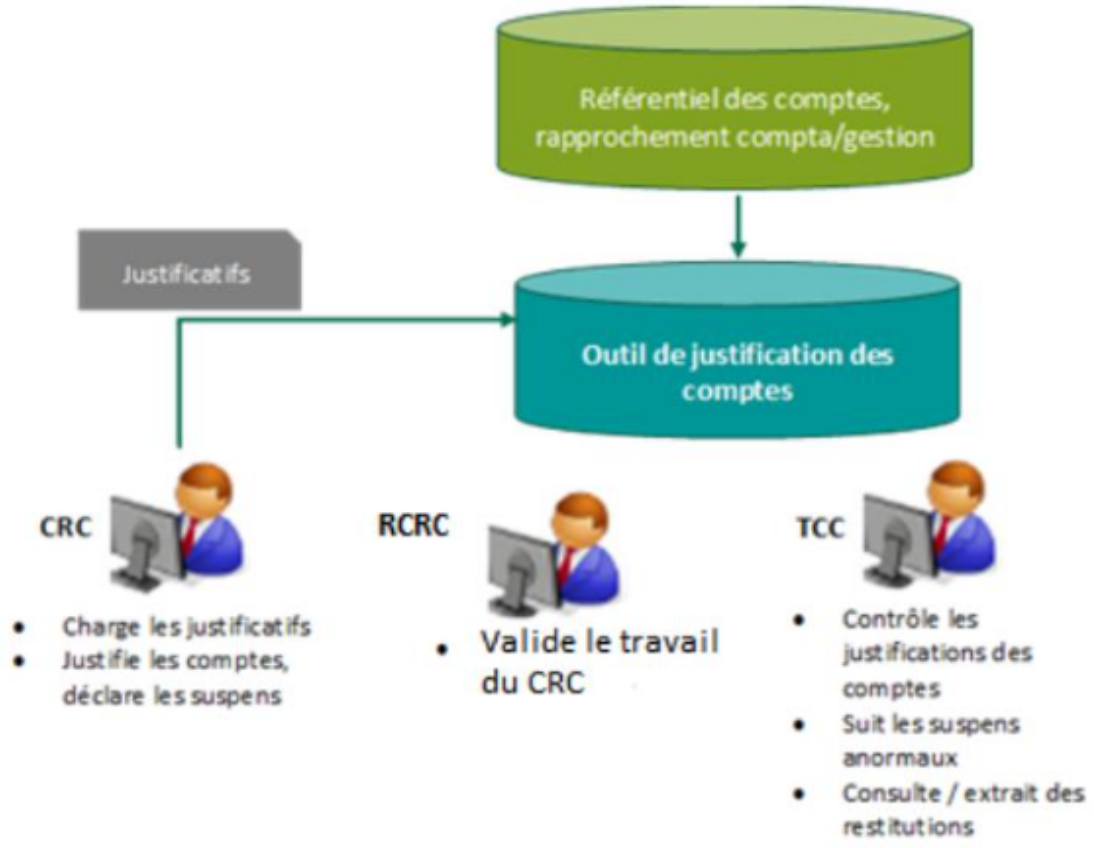

Fig. 2.1 : Explication des périmètres CRC, RCRC et TCC dans l'organisation comptable

La figure 2.1 présente les concepts fondamentaux des périmètres CRC, RCRC et TCC utilisés dans l'écosystème bancaire de Crédit Agricole CIB. Cette structuration organisationnelle est au cœur du fonctionnement de l'application CJUST et détermine les workflows de validation et les contrôles d'accès.

2.2 Objectifs du projet

L'objectif du projet est de mettre en place une interface de travail unique qui permettra de regrouper dans un même outil les différents travaux réalisés par les CRC et TCC afin de sécuriser, de simplifier et d'améliorer le processus de révision des comptes et le suivi des suspens anormaux.

L'outil doit permettre la centralisation et l'harmonisation des travaux de justification des comptes tout en automatisant un certain nombre de processus avec une traçabilité accrue des contrôles et des pistes d'audits.

2.2.1 Fonctionnalités principales

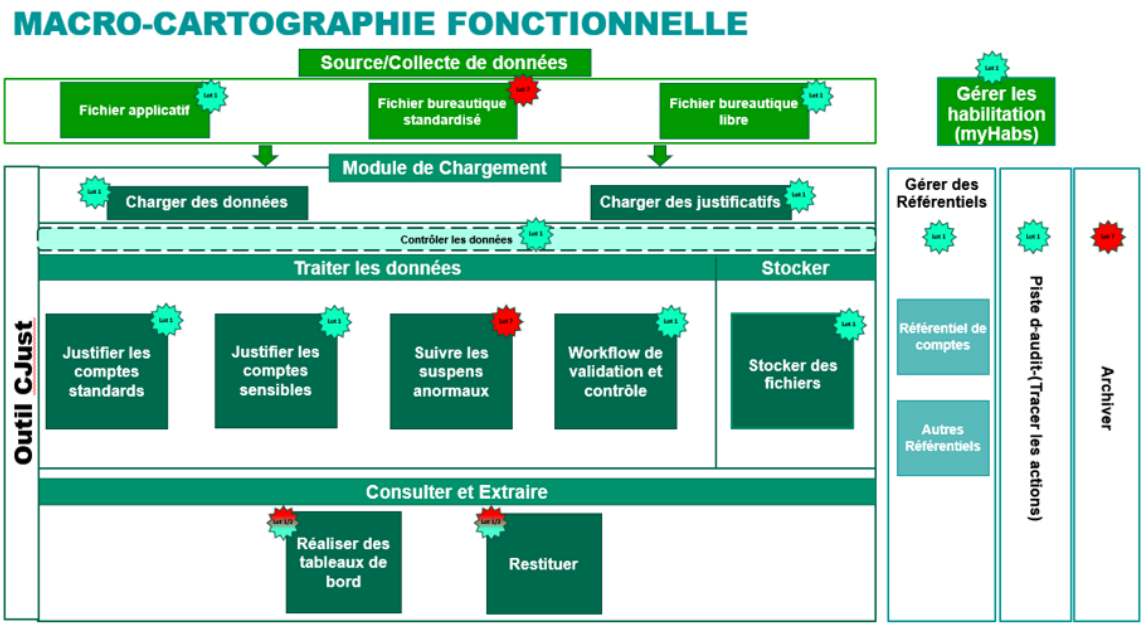

Fig. 2.2 : Workflow et processus de l'application CJUST

La figure 2.2 présente le workflow complet de l'application CJUST, illustrant les différentes étapes du processus de justification des comptes et de suivi des suspens anormaux, depuis la collecte des données jusqu'à la validation finale.

L'application CJUST doit intégrer les fonctionnalités suivantes :

- Gérer les droits et les habilitations
- Collecter les données
- Gérer les référentiels
- Justifier les comptes
- Valider les comptes
- Contrôler les comptes
- Workflow de validation
- Tracer les actions

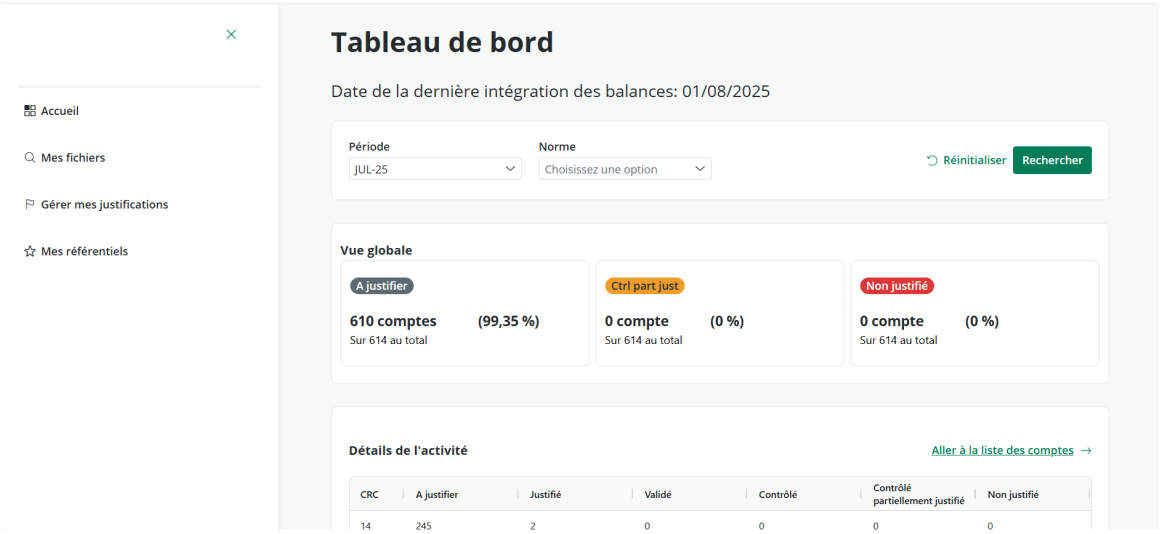

Fig. 2.3 : Interface d'accueil de l'application CJUST

La figure 2.3 présente l'interface d'accueil de l'application CJUST, montrant la page principale que les utilisateurs rencontrent lors de leur connexion. Cette interface intuitive permet d'accéder aux différentes fonctionnalités de l'application selon les droits et profils des utilisateurs.

2.3 Architecture technique

2.3.1 Infrastructure d'hébergement

L'application CJUST est hébergée on-premise. L'infrastructure d'hébergement de l'application est composée de :

- **Clusters Kubernetes** : Intranet HPROD et PROD sur les Datacenters pour l'hébergement des services de l'application CJUST
- **Une BDD PostgreSQL** : Instance unique contenant les tables de données de la BDD de l'application
- **Un espace de stockage de fichiers CJUST** sur S3 ECS

2.3.2 Composants de l'application

L'application CJUST est une application web intranet, elle est hébergée on-premise dans le réseau de CASA et est constituée d'un front-end et d'un back-end qui permettent aux utilisateurs du groupe CASA de réaliser les opérations détaillées dans la section précédente.

L'application CJUST est composée de :

- **UI – Front-end (Angular)** : WebUI de l'application
- **API – Backend (Spring Boot)** : API Rest qui permet aux utilisateurs selon leurs autorisations d'interroger et de mettre à jour les données qui sont stockées en BDD
- **Batch (Spring Batch)** : Batch d'ingestion des données qui sont issues du référentiel START et des données suivantes qui sont issues de l'application UG - Comptabilité générale CASA :
  - Balance comptable BAL 13
  - Référentiel des comptes PCI et PCCA

2.4 Sécurité et gestion des accès

Les droits d'accès des utilisateurs sont soumis à une gestion d'habilitation qui est administrée par des gestionnaires CASA. Cette gestion d'habilitation permet de :

- Contrôler l'accès aux différentes fonctionnalités selon le profil utilisateur
- Assurer la traçabilité des actions effectuées
- Maintenir la sécurité des données sensibles

2.5 Bénéfices attendus

La mise en place de cette interface de travail unique apportera les bénéfices suivants :

- **Centralisation** : Regroupement de tous les travaux dans un seul outil
- **Harmonisation** : Standardisation des processus de contrôle
- **Automatisation** : Réduction des tâches manuelles et des erreurs
- **Traçabilité** : Suivi complet des actions et contrôles effectués
- **Sécurisation** : Renforcement de la sécurité des processus
- **Simplification** : Réduction de la complexité des procédures

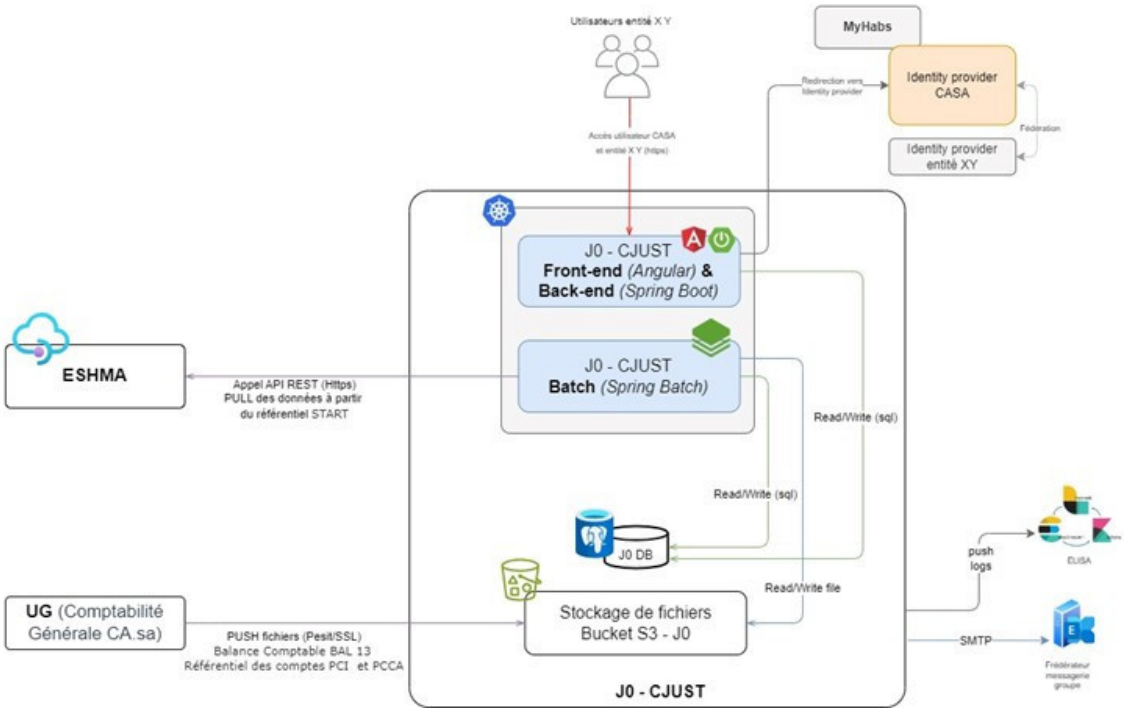

Fig. 2.4 : Architecture logique et technique de l'application CJUST

La figure 2.4 illustre l'architecture technique complète de l'application CJUST, montrant l'organisation des composants front-end, back-end et base de données, ainsi que leur interaction avec l'infrastructure Kubernetes et les services de stockage.

Chapitre 3

Analyse de l'existant

Cette section présente l'analyse de l'existant de l'application CJUST, en détaillant les différentes couches techniques qui composent l'architecture de l'application.

3.1 Couche de présentation (IHM)

L'interface homme-machine (IHM) est développée en Angular et elle permet aux utilisateurs selon leurs droits et profils d'accéder aux fonctionnalités suivantes :

- Gérer les droits et les habilitations
- Collecter les données
- Gérer les référentiels
- Justifier les comptes
- Valider les comptes
- Contrôler les comptes
- Workflow de validation
- Tracer les actions (piste d'audit)

3.1.1 Fonction

La couche de présentation (IHM) constitue l'interface utilisateur de l'application CJUST. Elle permet aux utilisateurs d'interagir avec le système selon leurs droits d'accès et leurs profils, en proposant une interface intuitive et ergonomique conforme aux standards du groupe Crédit Agricole.

3.1.2 Framework

Le frontend de l'application est développé en **Angular**.

La charte graphique utilisée est celle du groupe Crédit Agricole.

Angular est un framework open source, principalement utilisé pour créer des applications web et mobiles. Voici quelques caractéristiques clés du framework Angular :

- **Typescript** : Angular est écrit en TypeScript, un langage de programmation basé sur JavaScript. Cela permet d'ajouter des fonctionnalités de typage statique au JavaScript, ce qui rend le code plus fiable et plus facile à maintenir.
- **Architecture basée sur les composants** : Angular encourage la création d'applications modulaires en utilisant des composants réutilisables. Chaque composant encapsule une partie de l'interface utilisateur et de la logique.
- **Routing** : Angular propose un système de routage intégré pour gérer la navigation dans l'application en fonction des URL.
- **Injection de dépendances** : Angular facilite l'injection de dépendances, ce qui rend le code plus testable et modulaire.
- **Directives personnalisées** : Possibilité de créer des directives spécifiques pour étendre la syntaxe HTML et ajouter des fonctionnalités personnalisées aux modèles.
- **Services** : Les services sont des classes réutilisables qui permettent de partager de la logique métier entre différents composants.

3.1.3 Principes généraux

Description de l'architecture du framework Angular :

1. **Module (Module)** :
    - Les modules sont des conteneurs logiques de l'application Angular.
    - Ils regroupent des fonctionnalités similaires et définissent ce qui peut être utilisé dans une partie spécifique de l'application.
    - Chaque application Angular possède au moins un module racine (généralement appelé AppModule), qui est le point de départ de l'application.
2. **Component (Composant)** :
    - Les composants sont les blocs de construction fondamentaux d'Angular.
    - Ils sont responsables de la gestion de la logique et de l'interface utilisateur d'une partie de l'application.
    - Chaque composant est associé à un template qui définit son apparence et son comportement.
3. **Template (Modèle)** :
    - Le modèle est la partie de l'interface utilisateur d'un composant défini à l'aide d'un langage de modèle Angular.
    - Il contient le HTML, les balises Angular et les directives pour afficher les données et interagir avec l'utilisateur.
    - Le modèle est rendu dynamiquement pour afficher les données à l'utilisateur.
4. **Router (Routeur)** :
    - Le routeur Angular gère la navigation dans l'application en fonction des URL.
    - Il permet de définir des routes qui correspondent à des composants spécifiques à afficher lorsque l'URL change.
    - Le routeur permet également de passer des paramètres et de gérer la navigation entre les différentes vues de l'application.
5. **Metadata (Métadonnées)** :
    - Les métadonnées sont des informations spécifiques fournies en tant que décorateurs dans Angular.
    - Les décorateurs, tels que `@Component` et `@NgModule`, sont utilisés pour ajouter des métadonnées aux classes.

3.2 Couche de service

3.2.1 Description générale

La couche de service dans une architecture microservices joue le rôle de gestionnaire de la logique métier de l'application. Elle agit comme une couche intermédiaire entre les microservices individuels (qui se concentrent sur des fonctionnalités spécifiques) et l'ensemble de l'application.

Cette couche implémente le modèle CRUD (Création, Lecture, Mise à jour, Suppression) pour gérer les opérations sur les données métier.

3.2.2 Fonction

La couche service constitue la couche backend de CJUST et permet de traiter l'ensemble des requêtes utilisateurs provenant du frontend WebUI.

3.2.3 Framework

Le framework utilisé pour développer les fonctions du backend est Spring Boot.

Spring Boot est un outil qui accélère et simplifie le développement d'applications Web et de microservices avec le Spring Framework grâce à trois fonctionnalités principales :

1. Configuration automatique
2. Approche orientée configuration (convention over configuration)
3. Possibilité de créer des applications autonomes

Ces fonctionnalités fonctionnent ensemble pour fournir un outil qui permet de configurer une application Spring avec une configuration et une installation minimale.

Le framework utilisé pour développer le batch d'ingestion des données est Spring Batch.

Spring Batch est un framework conçu pour le traitement par lots (batch processing). Il est utilisé pour traiter de grands volumes de données, généralement de façon automatique et planifiée, en exécutant des tâches telles que la lecture de fichiers, la transformation de données, et l'enregistrement des résultats. Il gère des tâches courantes des traitements par lots, comme la gestion des transactions, le redémarrage après une erreur, le suivi de l'état des exécutions, etc.

Spring Boot fournit une intégration simplifiée avec Spring Batch. Grâce aux dépendances et aux configurations automatiques de Spring Boot, il est facile d'inclure Spring Batch dans une application Spring Boot. En combinant Spring Boot avec Spring Batch, il est possible de créer une application de traitement par lots entièrement autonome. Spring Boot peut encapsuler l'application batch dans un fichier JAR ou WAR exécutable, ce qui rend le déploiement et l'exécution faciles dans différents environnements.

3.3 Couche d'accès aux données

3.3.1 Description générale

L'accès aux données se fait soit via les API backend de CJUST, soit à partir du traitement batch d'intégration des données qui ingère les données issues de l'API ESHMA (référentiels START) ou des fichiers d'interface provenant de l'application UG.

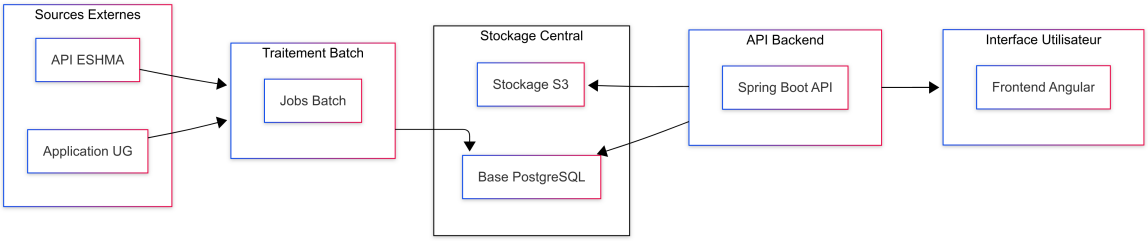

Fig. 3.1 : Flux de données et intégrations - CJUST

La figure 3.1 illustre les flux de données complets dans l'écosystème CJUST, depuis les sources externes (API ESHMA et application UG) jusqu'aux utilisateurs finaux, en passant par les traitements batch et les API backend.

3.3.2 Fonction

La couche d'accès aux données (ou couche de persistance) a pour rôle de gérer l'interaction avec la BDD de CJUST. Elle assure la communication entre les services backend Java développés en Spring Boot et les données stockées en BDD.

3.3.3 Framework

Le rôle principal de la couche d'accès aux données dans Spring Boot comprend les éléments suivants :

1. **Mapping Objet-Relationnel (ORM)** : Dans le cas d'une base de données relationnelle, la couche d'accès aux données utilise Spring Data JPA pour faciliter la correspondance entre les objets Java de l'application et les tables de la base de données. Elle permet de manipuler des objets Java au lieu de SQL brut.

2. **Définition des modèles de données** : Cette couche définit les entités ou les classes Java qui représentent les données stockées. Chaque classe correspond généralement à une table de base de données.

3. **Gestion des requêtes** : Elle gère la création, l'exécution et la gestion des requêtes pour récupérer, insérer, mettre à jour ou supprimer des données de la base de données. Cela peut se faire en utilisant le langage de requête adapté à la source de données, par exemple SQL pour les bases de données relationnelles.

4. **Transactions** : La couche d'accès aux données gère les transactions pour garantir l'intégrité des données. Elle assure que plusieurs opérations sur la base de données sont atomiques, c'est-à-dire qu'elles sont soit toutes effectuées, soit aucune d'entre elles.

5. **Optimisation des requêtes** : Elle peut également inclure des mécanismes d'optimisation des requêtes pour améliorer les performances en minimisant le nombre de requêtes à la base de données.

6. **Sécurité** : Elle peut contribuer à la sécurité en implémentant des mécanismes d'accès aux données sécurisés, tels que la gestion des autorisations et la prévention des attaques SQL injection.

7. **Gestion des erreurs** : En cas d'erreurs lors de l'accès aux données, cette couche gère les exceptions et les erreurs de manière appropriée, par exemple en les transformant en exceptions spécifiques à l'application.

En résumé, la couche d'accès aux données dans Spring Boot facilite l'interaction de l'application avec la base de données ou d'autres systèmes de stockage de données, en abstrayant les détails techniques et en fournissant une interface de programmation Java cohérente pour la manipulation des données. Cela simplifie le développement d'applications robustes et évolutives en permettant aux développeurs de se concentrer sur la logique métier plutôt que sur les détails de la persistance des données.

3.4 Bilan de l'architecture technique

L'architecture technique de CJUST s'articule autour de trois couches principales :

- **Couche de présentation** : Interface utilisateur développée avec Angular, offrant une expérience utilisateur moderne et réactive
- **Couche de service** : Backend développé avec Spring Boot pour la logique métier et Spring Batch pour le traitement par lots
- **Couche d'accès aux données** : Gestion de la persistance avec Spring Data JPA et intégration avec les systèmes externes (API ESHMA, application UG)

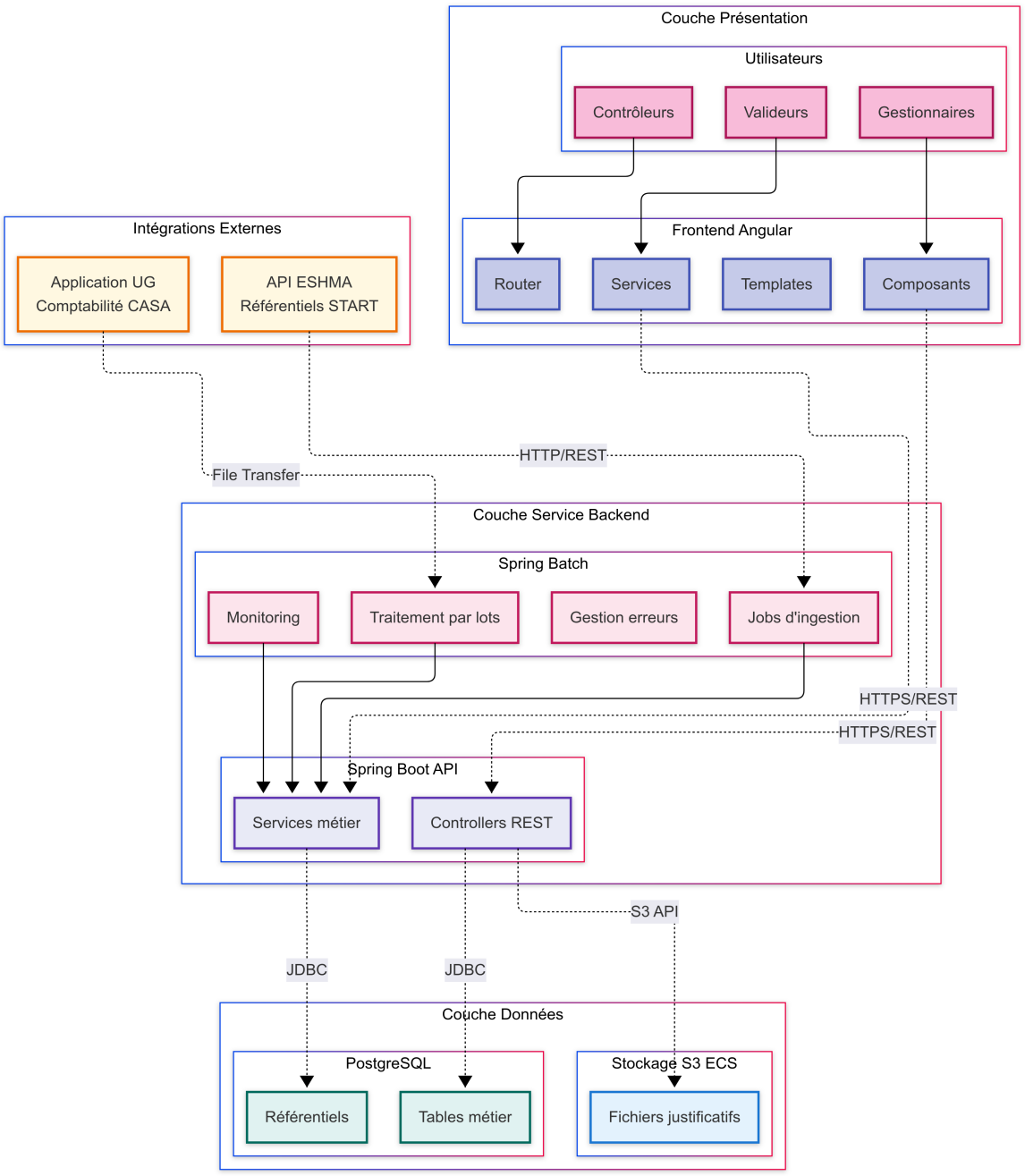

Fig. 3.2 : Architecture 3-tiers de CJUST - Vue d'ensemble technique

La figure 3.2 présente une vue d'ensemble de l'architecture 3-tiers de l'application CJUST. Elle montre les interactions entre la couche de présentation Angular, les services backend Spring Boot/Batch, et les couches de persistance avec leurs intégrations externes.

Chapitre 4

Implémentation et développement

Ce chapitre présente l'implémentation des deux missions principales réalisées durant ce stage au sein du groupe Crédit Agricole CIB. Ces missions s'inscrivent dans le cadre du développement et de la maintenance de l'application **CJUST**.

4.1 Mission 1 : Automatisation des tests de non-régression

4.1.1 Contexte et objectifs de la première mission

Cette première mission s'est concentrée sur l'automatisation des tests end-to-end pour garantir la qualité et la fiabilité de l'application CJUST. Les objectifs principaux étaient de :

- **Automatiser les tests end-to-end** pour garantir la qualité et la fiabilité de l'application CJUST avec une couverture complète des workflows métier ;
- **Implémenter une approche BDD** (Behavior Driven Development) avec Cucumber afin de créer des tests de non-régression ;
- **Assurer une couverture complète** des fonctionnalités critiques incluant la gestion des justifications, workflows de validation, et etc ;
- **Mettre en place un framework de test robuste** avec architecture Page Object Model et système de tags pour les évolutions futures ;

4.1.2 Système de gestion des justifications comptables

L'application CJUST gère un système critique de **gestion des justifications comptables** avec les fonctionnalités suivantes :

- **Import/Export de fichiers**
- **Workflows de validation** : Gestionnaire → Valideur → Contrôleur permanent
- **Contrôles d'accès** basés sur les rôles et périmètres CRC
- **Gestion des périodes** ouvertes/fermées avec calendrier

4.1.3 Architecture et approche technique

4.1.4 Choix technologiques

Le projet a été développé en utilisant **Cypress 14.4.1** comme framework de test principal, couplé à **Cucumber** pour l'approche BDD. Cette combinaison offre plusieurs avantages :

- **Cypress** : Framework moderne et performant pour les tests E2E ;
- **Cucumber** : Permet d'écrire les scénarios de test en langage naturel (Gherkin), facilitant la compréhension ;
- **Page Object Model** : Architecture modulaire facilitant la maintenance et la réutilisabilité du code.

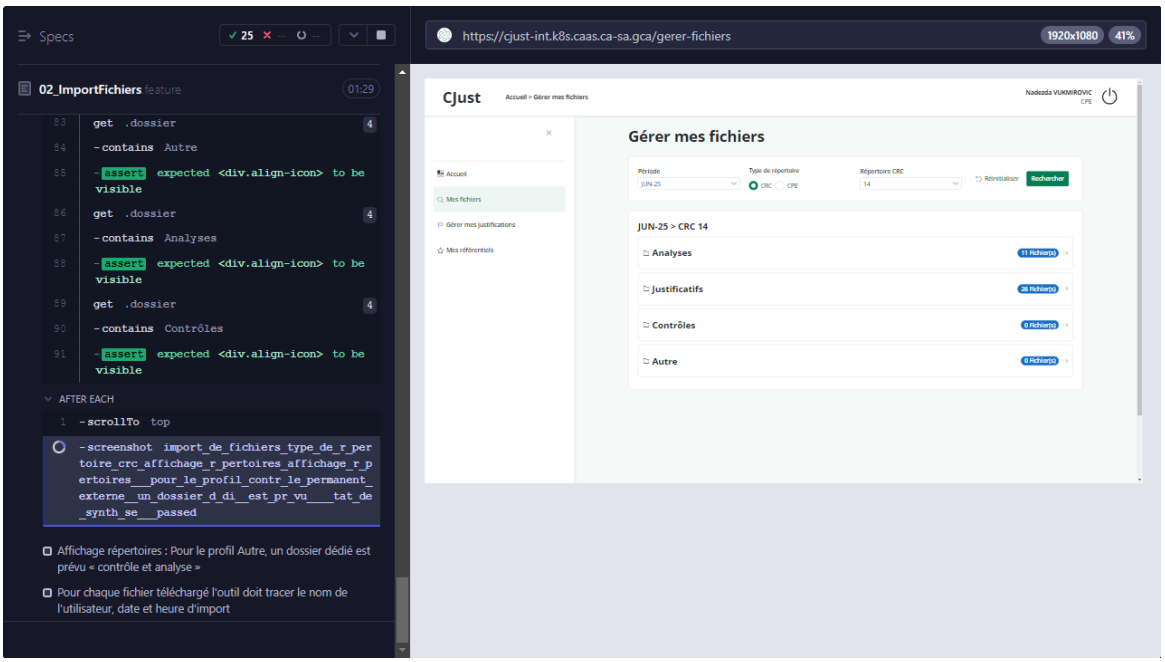

Fig. 4.1 : Interface de développement Cypress utilisée par l'équipe

La figure 4.1 présente l'interface de développement Cypress, l'outil principal utilisé par l'équipe pour l'automatisation des tests end-to-end. Cette interface moderne et intuitive permet aux développeurs de créer, exécuter et déboguer efficacement les tests automatisés, offrant une expérience de développement optimale pour la validation des fonctionnalités de l'application CJUST.

4.1.5 Structure du projet

La structure du projet suit une organisation claire et modulaire avec une architecture BDD complète :

```
cypress_TNR/
├── cypress.config.js            # Configuration Cypress avec plugins avancés
├── e2e/                         # Tests end-to-end (Architecture BDD)
│   ├── features/                # 11 fichiers de spécifications Cucumber
│   │   ├── 01_PerimetreCRC.feature          # Tests de périmètre par rôle
│   │   ├── 02_ImportFichiers.feature        # Tests d'import fichiers
│   │   └── ...
│   ├── pages/                   # Page Objects (POM Pattern)
│   │   ├── BasePage.js                      # Classe de base
│   │   ├── DashboardPage.js                 # Tableau de bord
│   │   └── ...
│   ├── step_definitions/        # Implémentations Cucumber
│   │   ├── dashboard.steps.js               # Étapes tableau de bord
│   │   └── ...
│   └── utils/                   # Utilitaires
│       └── TestDataManager.js               # Gestionnaire données
├── fixtures/                    # Données de test et sessions
│   ├── dev/cookies/                         # Sessions environnement DEV
│   ├── int/cookies/                         # Sessions environnement INT
│   ├── dataFile.json                        # Données de test centralisées
│   └── userInfo.json                        # Configuration 13 rôles
├── support/                     # Configuration globale Cypress
│   ├── commands.js                          # Commandes personnalisées
│   └── e2e.js                               # Configuration globale
└── test_fiches/                 # Fichiers de test pour import
    └── test-toto1.docx                      # Fichier test
```

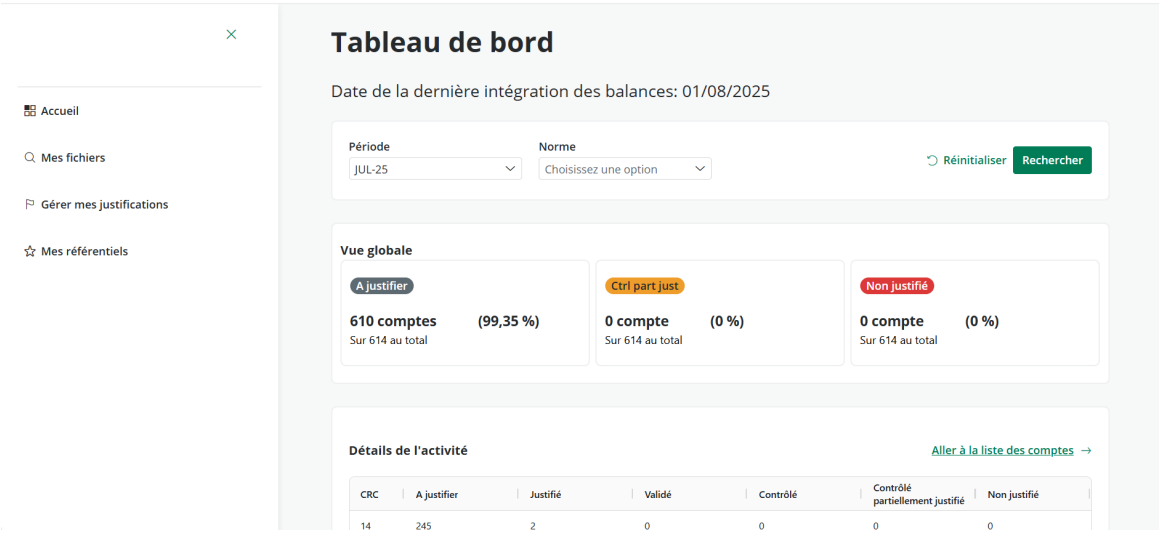

Fig. 4.2 : Interface d'accueil de l'application CJUST

La figure 4.2 présente l'interface d'accueil de l'application CJUST, montrant la page principale que les utilisateurs rencontrent lors de leur connexion. Cette interface intuitive permet d'accéder aux différentes fonctionnalités de l'application selon les droits et profils des utilisateurs.

4.1.6 Architecture Page Object Model avancée

Le projet implémente une architecture **Page Object Model étendue** avec **classes spécialisées** :
Cette architecture permet une séparation claire entre :

- **Les Page Objects** : Encapsulent les interactions avec l'interface utilisateur
- **Les Step Definitions** : Implémentent la logique métier des scénarios Cucumber
- **Les Features** : Définissent les comportements attendus en langage naturel

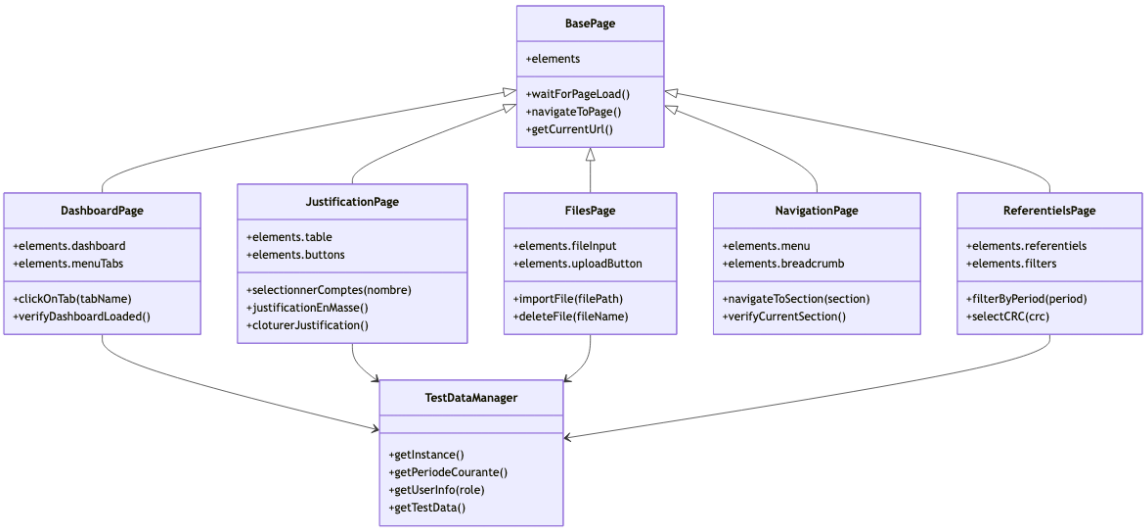

Fig. 4.3 : Architecture des classes Page Object Model avec héritage et dépendances

La figure 4.3 présente l'architecture des classes Page Object Model implémentée. La classe BasePage centralise les fonctionnalités communes, tandis que les classes spécialisées héritent de cette base et utilisent le TestDataManager pour la gestion centralisée des données de test.

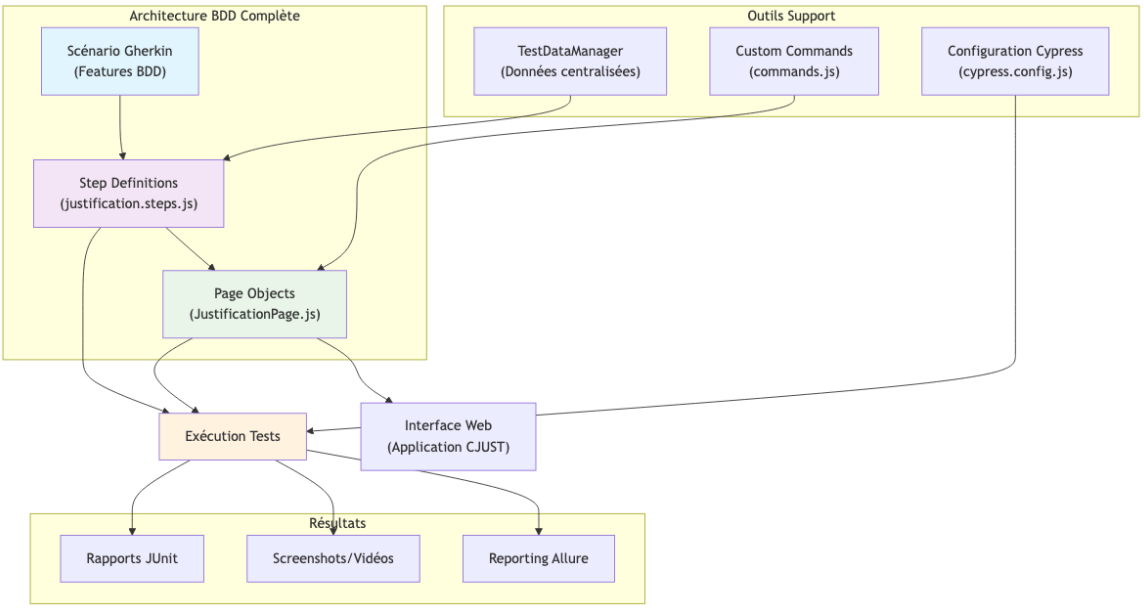

Fig. 4.4 : Architecture BDD complète : de la spécification Gherkin à l'exécution

La figure 4.4 illustre l'architecture complète de l'approche BDD implémentée. Le flux montre comment les spécifications métier en langage naturel (Gherkin) sont traduites en étapes techniques (Step Definitions) puis en interactions avec l'interface utilisateur via les Page Objects, avec le support des utilitaires centralisés.

4.1.6.1 Logique en chaîne complète : Scénario → Step Definitions → Page Objects

Voici la logique complète illustrée avec l'exemple de justification en masse :

4.1.6.1.1 1. Scénario Cucumber (Feature) - Justification en masse

```
@gestcompta @justification
Scénario: [FINLQPRO] Justification en masse
Quand je visite le tableau de bord
Et je clique sur l'onglet Gérer mes justifications
Et je sélectionne l'option Période "courante" Statut "A justifier" CRC "choice_CRC"
Et je clique sur le bouton <Rechercher> dans la page <Liste des comptes>
Et je sélectionne 4 comptes dans la page <Liste des comptes>
Et je clique le bouton <Justification en masse> dans la page <Liste des comptes>
Et je saisis le nom du lot dans la page <Justification en masse>
Et je clique <Clôturer la justification> dans la page <Justification en masse>
```

4.1.6.1.2 2. Step Definitions (justification.steps.js) - Implémentation une étape de Cucumber

```
// Sélection de comptes
When("je sélectionne {int} comptes dans la page <Liste des comptes>",
  function (nombreLignes) {
    JustificationPage.selectionnerComptes(nombreLignes);
  }
);
```

4.1.6.1.3 3. Page Objects (JustificationPage.js) - Méthode Page Object pour la sélection

```
class JustificationPage extends BasePage {
  elements = {
    // Table des comptes
    table: () ⇒ cy.get("cjust-liste-comptes ag-grid-angular"),
  };

  // Sélectionner des comptes dans la liste
  selectionnerComptes(nombreLignes) {
```

```
    this.elements.table().find(".ag-center-cols-container .ag-row").then(($rows) => {
      const rowCount = Math.min($rows.length, nombreLignes);
      for (let i = 0; i < rowCount; i++) {
        cy.wrap($rows[i]).find(".ag-cell").eq(0).click();
      }
    });
  }
}
```

4.1.6.1.4 Flux de données et logique métier

Cette chaîne illustre le **workflow complet** :

1. **Scénario Gherkin** : Définit le comportement métier en langage naturel
2. **Step Definitions** : Traduit chaque étape en appels de méthodes Page Object
3. **Page Objects** : Implémente les interactions UI avec gestion des intercepts API
4. **TestDataManager** : Gère les données de test et les périodes courantes

4.1.7 Système de tags Cucumber complet

Le projet utilise un **système de tags Cucumber complet** avec **tags organisés** en 4 catégories :

4.1.7.1 Tags par fonctionnalité métier

- `@import-suppression` - Tests d'importation et suppression de fichiers justificatifs
- `@justification` - Processus de justification des comptes (en masse et unitaire)
- ... etc

4.1.7.2 Tags par rôle utilisateur

- `@gestcompta` - Tests pour gestionnaires comptables
- `@valideur` - Tests pour valideurs
- ... etc

4.1.7.3 Tags par type de test

- `@smoke` - Tests critiques prioritaires
- `@negative` - Tests de cas de restrictions d'accès
- ... etc

4.1.7.4 Stratégie d'Exécution

1. **Tests de regression** : `@smoke` (tests critiques rapides)
2. **Tests fonctionnels** : Par domaine (`@import-suppression`, `@justification`)
3. **Tests par rôle** : Validation des permissions par profil utilisateur
4. **Tests négatifs** : `@negative` pour validation des restrictions
5. **Tests complets** : Exécution sans filtre pour couverture totale

4.1.8 Déploiement et intégration continue

4.1.9 Configuration GitLab CI/CD

Le projet est configuré pour s'intégrer dans un pipeline CI/CD avec :

- **Exécution automatique** des tests sur push/merge request
- **Environnements isolés** (dev, int)
- **Reporting automatisé** avec Allure

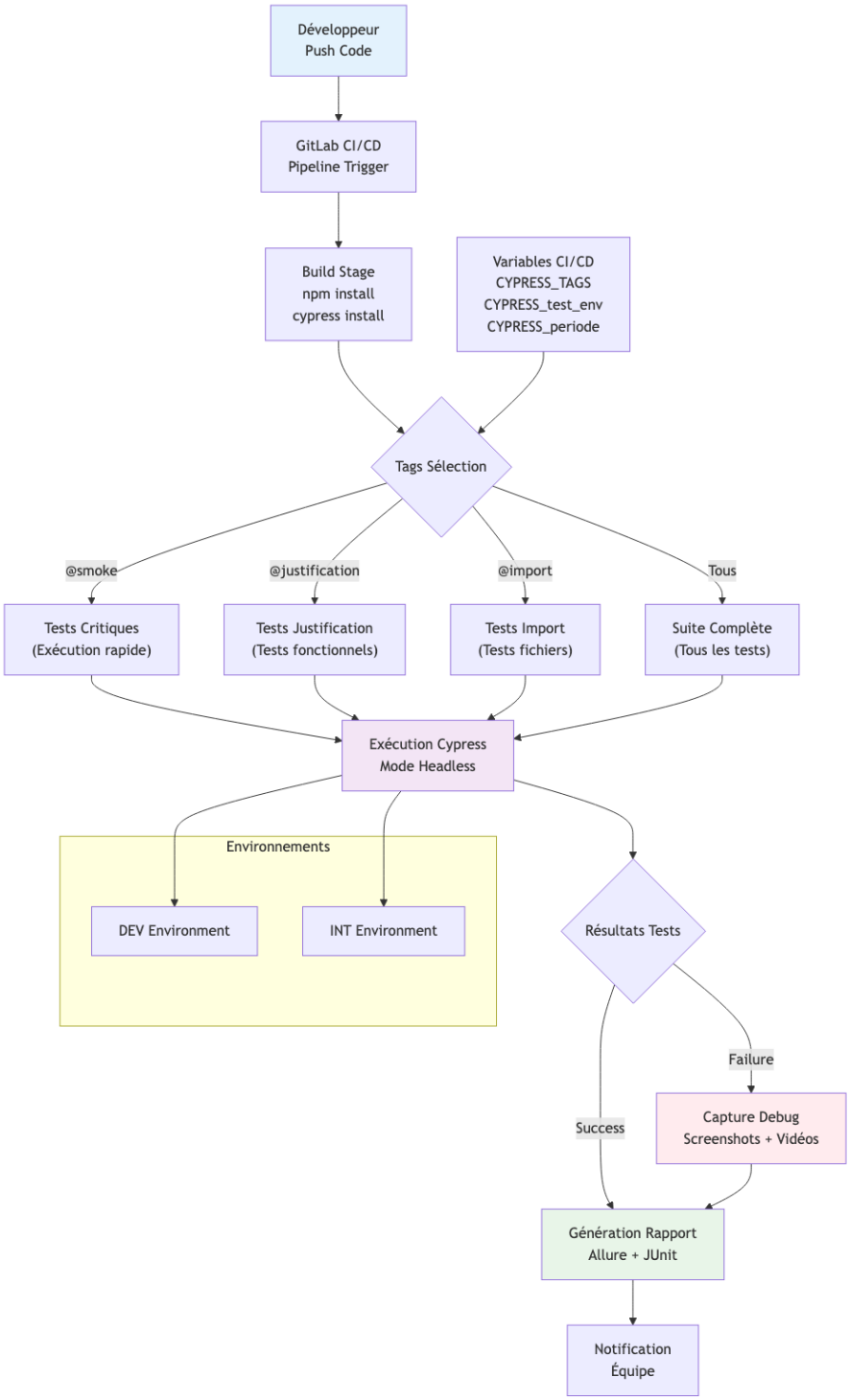

Fig. 4.5 : Workflow complet du pipeline CI/CD avec exécution sélective par tags

La figure 4.5 détaille le processus d'intégration continue mis en place. Le pipeline permet une exécution sélective des tests via le système de tags Cucumber, avec génération automatique de rapports et notifications en cas d'échec.

4.1.10 Outils de reporting

Plusieurs outils de reporting ont été intégrés :

- **Allure** : Rapports détaillés avec métriques et visualisations
- **JUnit** : Rapports XML pour l'intégration avec les outils CI/CD

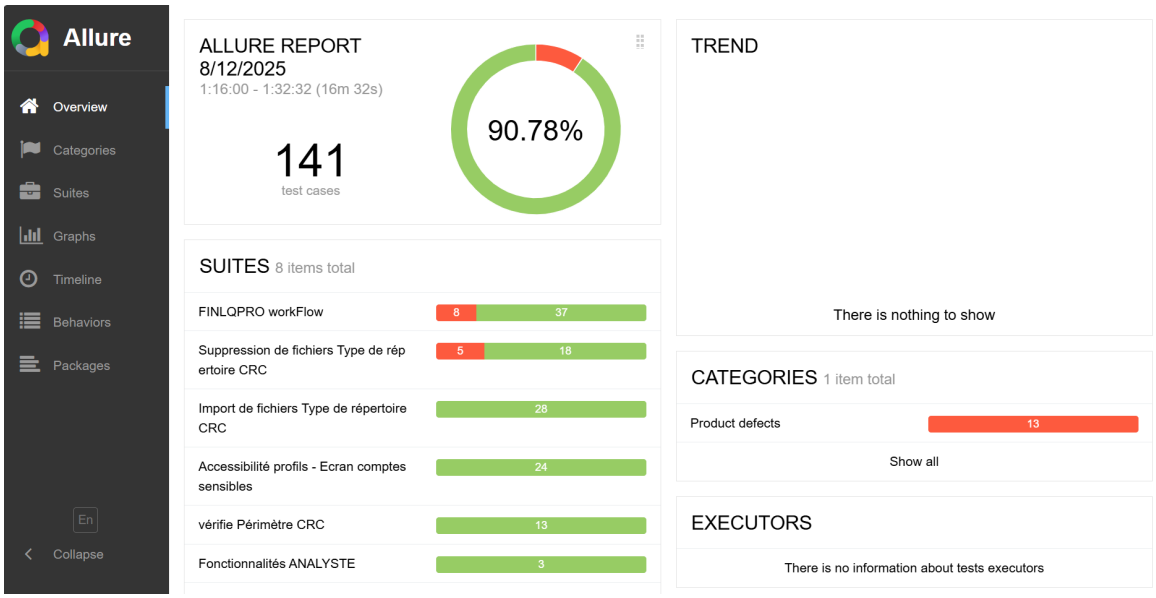

Fig. 4.6 : Dashboard principal d'Allure pour le reporting des tests

La figure 4.6 présente le dashboard principal d'Allure, l'outil de reporting utilisé pour visualiser les résultats des tests automatisés. Ce dashboard offre une vue d'ensemble claire des exécutions de tests avec des métriques détaillées et des visualisations graphiques.

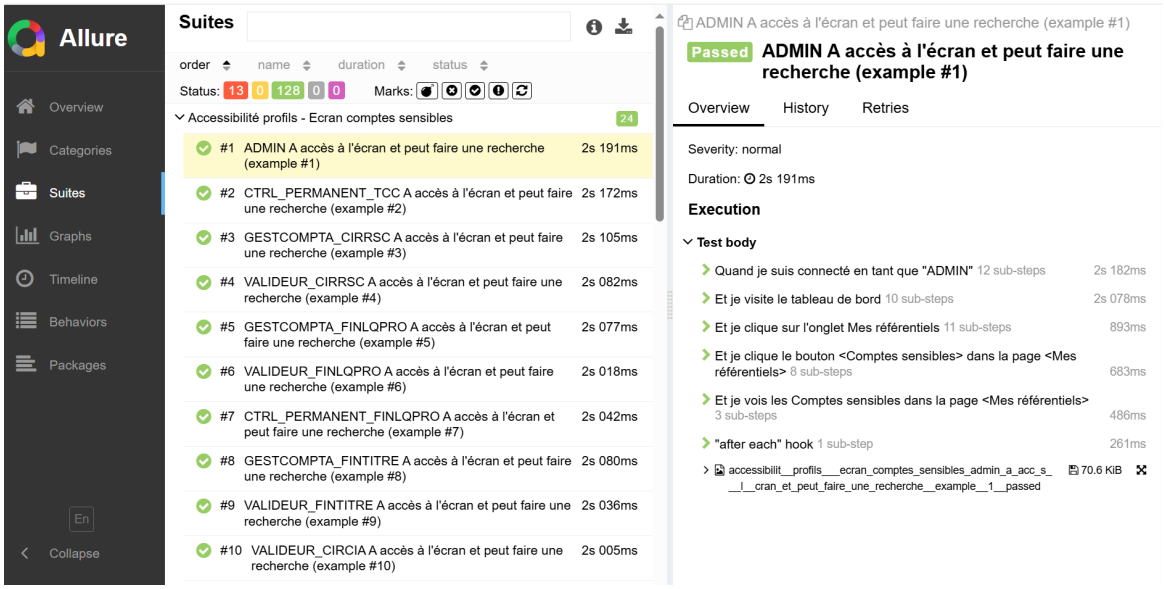

Fig. 4.7 : Rapport détaillé Allure avec analyse des étapes de test

La figure 4.7 présente un rapport détaillé d'Allure, montrant l'analyse approfondie des étapes de test avec des informations sur les temps d'exécution, les captures d'écran et les logs détaillés. Cette vue permet aux développeurs d'identifier précisément les points de défaillance et d'optimiser les performances des tests.

4.1.11 Exécution des tests

4.1.11.1 Exécution locale

Les tests peuvent être exécutés localement avec différentes commandes :

```
# Mode interactif
npm cypress open

# Mode ligne de commande
npm cypress run

# Tests spécifiques par fonctionnalité
npx cypress run --env TAGS="@import @smoke"
npx cypress run --spec "cypress/e2e/features/02_ImportFichiers.feature"

# Générer un rapport HTML simple
npx allure generate ./allure-results --single-file --clean -o ./allure-report
```

Listing 4.1: Commandes d'exécution locale

4.1.11.2 Exécution CI/CD

Le projet est configuré pour s'exécuter automatiquement dans un pipeline GitLab CI/CD :

4.1.11.2.1 Configuration des variables d'environnement CI/CD Le pipeline utilise plusieurs variables configurables pour adapter l'exécution :

- **CYPRESS_TAGS** : Filtrage des tests par tags Cucumber (ex : `@justification`, `@smoke`)
- **CYPRESS_test_env** : Paramètre pour l'environnement où sont lancés les tests (dev, int)
- **CYPRESS_periode** : Configuration periode current comptable pour les tests
- **CYPRESS_periode_fin_controle** : Configuration periode fin de controle pour les tests
- **CYPRESS_periode_fermee** : Configuration periode fermee pour les tests

4.1.11.2.2 Déclenchement et monitoring La pipeline peut être déclenchée manuellement via l'interface GitLab avec possibilité de :

- Configurer les variables d'environnement avant exécution
- Sélectionner les tags de test spécifiques
- Monitorer l'exécution en temps réel
- Consulter les rapports générés automatiquement

4.1.12 Conclusion Mission 1 : Tests de non-régression

Cette première mission d'automatisation des tests end-to-end pour l'application CJUST représente une approche de renforcement de la qualité logicielle au sein de l'équipe. L'implémentation d'une approche BDD complète avec Cypress et Cucumber a permis de créer un framework de test robuste et évolutif, adapté aux exigences spécifiques à partir du cahier des charges de la recette.

4.1.12.1 Couverture fonctionnelle

Les résultats obtenus montrent une couverture complète :

- **100% des rôles** utilisateurs testés
- **Fonctionnalités** principales couvertes

Composant	Fichiers	Complexité
Features BDD	11	Scénarios métier
Page Objects	6	Architecture POM
Step Definitions	4	Implémentations Cucumber
Custom Commands	1	Commandes personnalisées
Utilitaires	1	Gestionnaire données centralisé
Configuration	3	Configuration Cypress

Tab. 4.1 : Métriques techniques détaillées du projet

4.1.12.2 Réussites techniques accomplies

La mission a atteint ses objectifs :

- Couverture fonctionnelle : fonctionnalités métiers critiques testées, couvrant l'ensemble des workflows de justification comptable.
- Architecture technique solide : implémentation du pattern Page Object Model avec classes spécialisées, architecture BDD complète avec 4 fichiers de step definitions, et gestion centralisée des données via le pattern Singleton.
- Couverture utilisateur complète : validation de 13 rôles métiers avec leurs périmètres CRC spécifiques, garantissant la conformité aux exigences de sécurité et d'habilitation du répertoire.
- Intégration DevOps : pipeline CI/CD GitLab opérationnelle avec reporting Allure, exécution sélective par tags Cucumber.

4.1.12.3 Impact organisationnel et métier

L'approche BDD adoptée a transformé la collaboration entre équipes MOE et MOA :

- Documentation vivante : les 11 fichiers de spécifications Cucumber en français constituent une documentation technique et métier maintenue automatiquement, facilitant la compréhension des processus complexes de justification comptable.

4.1.12.4 Perspectives d'évolution et réutilisabilité

Le framework développé présente des caractéristiques stratégiques pour l'organisation :

- Évolutivité : architecture modulaire permettant l'ajout facile de nouveaux tests lors des évolutions de l'application CJUST.
- Réutilisabilité : patterns de conception et structure du projet transposables à d'autres applications de l'équipe.
- Maintenabilité : séparation claire des responsabilités et utilisation de patterns éprouvés facilitant la maintenance et l'évolution du code.

4.2 Mission 2 : Développement full stack

4.2.1 Contexte et objectifs de la deuxième mission

Cette deuxième mission a porté sur le développement full stack du lot 2 de l'application **CJUST**. L'objectif principal était d'introduire une nouvelle fonctionnalité de **gestion des comptes anormaux** via la notion de suspens. Les objectifs principaux étaient de :

- **Développer un traitement batch automatisé** pour la reprojection des suspens non apurés entre périodes comptables ;
- **Implémenter les règles métier complexes** de sélection et de filtrage des fiches de suivi selon les statuts d'apurement ;
- **Garantir la traçabilité et l'observabilité** du traitement via logging structuré et notifications e-mail ;
- **Assurer l'intégration dans le pipeline CI/CD** avec déploiement automatisé sur Kubernetes via GitLab ;

4.2.2 Problématique métier et traitement batch

Pour chaque période de gestion, la base de données contient une table `fich_suivi_suspens` (fiches de suivi) et les enregistrements de suspens associés. Le traitement **batch** à concevoir devait alimenter la nouvelle période en reprojetant de manière **sélective** les fiches et suspens **non apurés** de la période précédente.

Il ne s'agit pas d'une simple copie technique : un ensemble de règles métier (statuts d'apurement, cohérence des dates de période, etc.) détermine l'éligibilité des éléments à reporter.

Le batch exécute une chaîne structurée :

- Sélection des fiches et suspens non apurés de la période N-1 ;
- Application des règles de filtrage ;
- Création contrôlée des fiches de suivi et rattachement des suspens dans la période N ;
- Envoi de notifications e-mail aux utilisateurs concernés selon le résultat (succès complet, erreurs bloquantes).

Ce traitement automatisé garantit la continuité du suivi des suspens entre périodes, fiabilise le démarrage de la nouvelle période de justification et réduit les interventions manuelles tout en respectant strictement la logique métier.

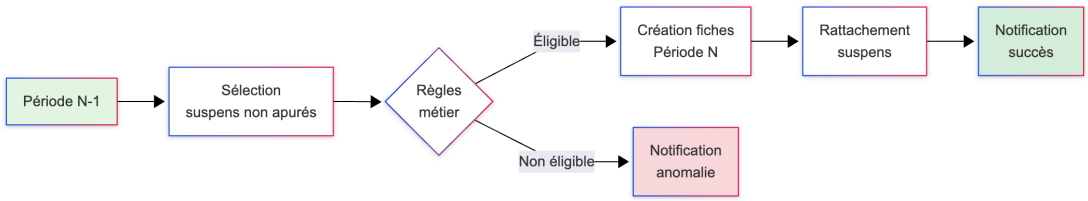

Fig. 4.8 : Workflow simplifié du traitement batch de reprojection des suspens

4.2.3 Choix technologiques

Le projet a été développé en utilisant **Java 21** et l'écosystème **Spring Boot / Spring Batch** comme socle technologique principal. Ce choix s'appuie sur plusieurs critères techniques et opérationnels :

- **Spring Batch** : Framework robuste pour le traitement batch avec gestion native des états d'exécution (tables `BATCH_*`), permettant une reprise contrôlée en cas d'arrêt et un redémarrage idempotent ;
- **Spring Data JPA** : Simplification de l'accès aux données avec repositories typés et requêtes optimisées, réduisant le temps de développement et minimisant les risques d'erreurs SQL ;
- **PostgreSQL** : Base de données relationnelle garantissant l'intégrité et la cohérence des données comptables avec transactions contrôlées ;
- **JUnit** : Tests unitaires et d'intégration garantissant la non-régression des règles de filtrage (statuts d'apurement, bornes de dates) et la stabilité du mapping ;
- **Lombok** : Réduction du code standard (getters/setters, builders) pour améliorer la lisibilité des règles métiers ;
- **Kubernetes + GitLab CI/CD** : Déploiement automatisé et orchestration des jobs via CronJob avec pipeline d'intégration continue.

Le tableau suivant détaille l'ensemble des technologies utilisées et leur rôle dans le projet :

Domaine	Technologie	Rôle / Usage principal
Langage	Java 21	Nouvelles fonctionnalités du JDK (records, pattern matching) et meilleures performances pour un code pérenne.
Framework batch	Spring Batch	Définition du *Job*, d'un Step unique (lecture / filtrage / écriture), gestion transactionnelle, redémarrage idempotent, métadonnées d'exécution.
Accès aux données	Spring Data JPA	Repositories typés, requêtes dérivées et `@Query` ciblées pour la sélection filtrée des fiches et suspens non apurés.
Mapping persistance	Entités JPA (PostgreSQL)	Modèle de données aligné sur les tables et entités associées ; gestion des relations et contraintes d'intégrité.
Configuration build	Maven (`pom.xml`)	Gestion centralisée des dépendances, packaging exécutable.
Tests	JUnit	Tests unitaires (règles de filtrage) et tests d'intégration (Step complet avec base embarquée et dataset contrôlé).
Utilitaires	Lombok	Réduction du code standard (getters/setters, builders) pour se concentrer sur la logique métier.
Base de données	PostgreSQL	Stockage persistant des fiches et suspens, avec transactions contrôlées pendant le batch.
Infra d'exécution	Spring Boot	Démarrage rapide, injection de dépendances et configuration externalisée.
Orchestration	Kubernetes	Déploiement et exécution du batch via CronJob, gestion des ressources (CPU, mémoire), secrets et ConfigMaps.
CI/CD	GitLab CI/CD	Pipeline automatisé : build, tests, analyse qualité (Sonar), construction d'images Docker, promotion multi-environnements et déploiement via Kustomize.
Déploiement continu	Argo CD	GitOps pour la synchronisation déclarative des manifests Kubernetes et le déploiement automatisé.

Tab. 4.2 : Panorama des technologies utilisées

4.2.3.1 Vue technique de l'architecture

La figure ci-dessous présente l'architecture technique complète du module batch, illustrant les interactions entre les différentes couches (orchestration, métier, données et infrastructure) ainsi que les technologies utilisées à chaque niveau.

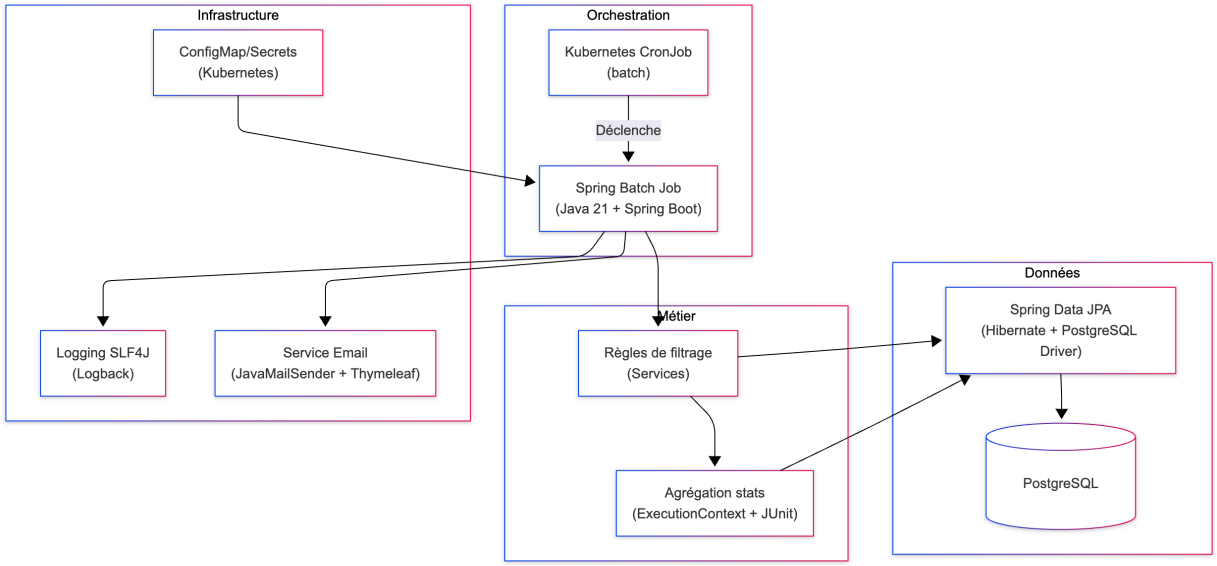

Fig. 4.9 : Architecture technique du module batch avec les technologies impliquées

4.2.4 Architecture et implémentation

Le module batch est structuré selon une architecture en couches spécialisées, chacune encapsulant un périmètre fonctionnel précis. Cette architecture garantit un traitement à la fois **déterministe**, **traçable** et **extensible**, facilitant les évolutions futures (telles que l'ajout de nouvelles règles de sélection ou l'enrichissement des notifications).

```
Couches logicielles du module Batch
-----------------------------------
Batch
 └─ (Spring Batch : Jobs / Steps / Tasklets / Orchestration)
    │
Services métiers
 └─ (Coordination des règles, agrégation, notifications)
    │
Accès aux données
 └─ (Spring Data JPA / Repositories / Requêtes ciblées)
    │
Persistance
 └─ (PostgreSQL / HSQLDB pour les tests / Intégrations S3 pour les échanges)
    │
Transverse
 └─ (Configuration, logging, gestion des exceptions, utilitaires, énumérations)
```

4.2.4.1 Rôle de chaque couche

- **Batch** : Déclare le Job principal, configure le Step selon le segment logique. redémarrage.
- **Services métier** : Encapsulent les règles d'éligibilité (statuts d'apurement, cohérence période source/cible, exclusion des éléments apurés), préparent les agrégats statistiques et déclenchent les notifications e-mail.
- **Accès données** : Fournit des méthodes de sélection optimisées (requêtes dérivées, `@Query`) pour récupérer un sous-ensemble minimal d'entités ; limite la logique au filtrage en base lorsque pertinent.
- **Persistence** : Gestion des entités JPA, base PostgreSQL en production et base embarquée (HSQLDB) en tests pour scénarios reproductibles ; intégration S3 potentielle pour échanger des exports si besoin.
- **Transverses** : Composants communs (configuration applicative, gestion centralisée des exceptions métiers vs techniques, utilitaires de dates/formats, énumérations de statuts) utilisés de façon partagée.

Cette stratification réduit le couplage : aucun service métier n'accède directement aux API bas niveau de persistence ; l'orchestrateur de batch ne code pas les règles, mais délègue aux services ; les exceptions métier remontent avec un contexte enrichi (identifiants de période, compteurs) pour un logging exploitable.

4.2.4.2 Organisation des packages principaux

L'arborescence des packages reflète ces responsabilités :

Package	Rôle consolidé
`com.casa.cjust.batch.*`	Définition des Jobs / Steps, Tasklets, implémentations concrètes des services métier, exposition des métriques de fin de run.
`com.casa.cjust.configuration`	Beans Spring : datasource, transaction manager, configuration Batch, mail, (option S3), propriétés externes (périodes, envoi e-mails).
`com.casa.cjust.persistence.entity`	Entités JPA : mapping des tables (fiches, suspens, calendrier périodes, historiques) et relations ; annotations de contraintes.
`com.casa.cjust.enumeration`	Énumérations métier (statuts d'apurement, codes de notification, types d'anomalie) centralisant les valeurs contrôlées.
`com.casa.cjust.exception`	Exceptions spécialisées (métier vs technique) + stratégie de propagation pour différencier échec partiel / bloquant.
`com.casa.cjust.utils`	Outils transverses (dates période, format, conversions collections, helpers idempotence).

Tab. 4.3 : Organisation des packages et responsabilités associées

4.2.5 Composants principaux du module Suspens

Le module de reprojection des suspens s'articule autour de plusieurs composants clés qui assurent l'orchestration, l'exécution et la traçabilité du traitement batch.

4.2.5.1 Composants d'orchestration

- **SuspensBatchConfiguration.java** : Configuration principale Spring Batch qui déclare le Job *copieSuspensJob* et ses Steps (*step1Suspens*, *step2Suspens*) en utilisant les API Spring Batch (*JobBuilder*/*StepBuilder*).
- **SuspensRepositories.java** : Façade qui agrège tous les repositories requis, réduisant la complexité d'injection et offrant un point unique d'accès à la persistance.

4.2.5.2 Composants de traitement

- **SuspensTasklet.java** : Composant cœur qui applique les règles métier de filtrage, effectue les opérations de copie/-création/suppression et collecte les statistiques d'exécution. Utilise des records internes pour structurer les catégories d'actions et stocke les métriques dans l'*ExecutionContext*.
- **SuspensRapportTasklet.java** : Step 2 qui récupère les statistiques de l'*ExecutionContext* et trace un rapport synthétique (période, créations, copies, suppressions) dans les logs.

4.2.5.3 Composants de notification

- **SuspensJobCompletionNotificationListener.java** : Listener de post-traitement qui lit les métriques et le contexte d'exécution, puis envoie un e-mail HTML récapitulatif via Thymeleaf avec variables dynamiques et messages typés selon le résultat (succès/échec).

Cette architecture sépare clairement les responsabilités : *SuspensTasklet* concentre la logique métier, tandis que la configuration et les Listeners assurent l'orchestration, l'observabilité et la communication.

4.2.6 Flux d'exécution du traitement

Le traitement batch s'exécute selon un enchaînement séquentiel de trois phases principales, chacune ayant un rôle spécifique dans le processus de reprojection des suspens.

4.2.6.1 Phase 1 : Traitement des suspens (Step 1)

Le *SuspensTasklet* effectue le traitement principal :

1. **Chargement des référentiels** : Récupération des données via les repositories agrégés
2. **Application des règles métier** : Filtrage selon les statuts (actif/inactif), balances et autres critères
3. **Exécution des actions** : Création, copie ou suppression des fiches de suivi selon l'éligibilité
4. **Collecte des métriques** : Stockage des statistiques et messages dans l'*ExecutionContext*

4.2.6.2 Phase 2 : Génération du rapport (Step 2)

Le *SuspensRapportTasklet* produit une synthèse du traitement :

- Lecture des métriques stockées dans l'*ExecutionContext*
- Génération d'un rapport détaillé incluant la période traitée, le nombre de créations, copies et suppressions
- Enregistrement du rapport dans les logs structurés

4.2.6.3 Phase 3 : Notification (Listener)

Le *SuspensJobCompletionNotificationListener* assure la communication finale :

- Analyse du résultat global de l'exécution (succès complet, partiel ou échec)
- Composition d'un e-mail HTML récapitulatif via templates Thymeleaf
- Envoi aux utilisateurs concernés avec variables dynamiques et contexte d'exécution

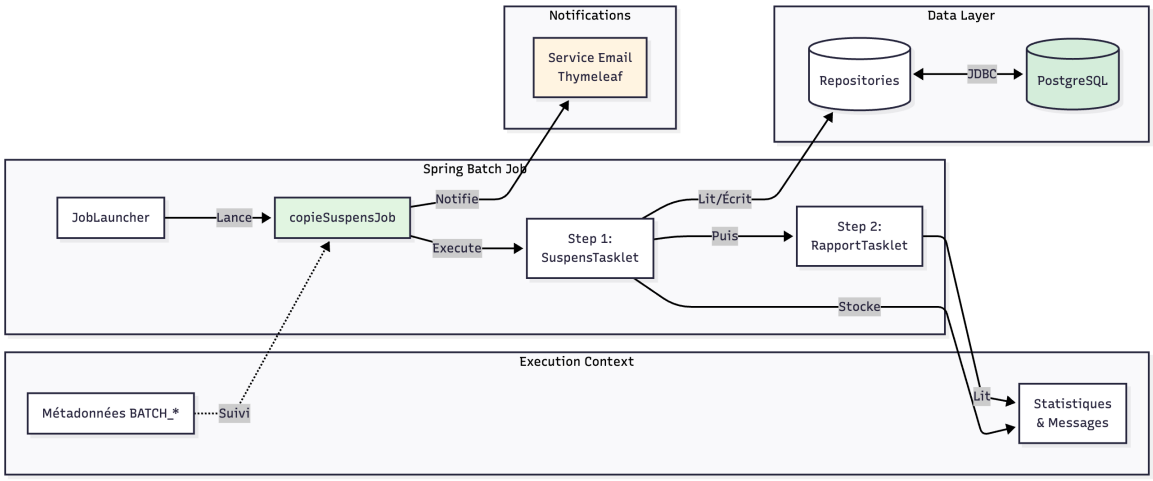

Fig. 4.10 : Flux d'exécution du Job copieSuspensJob avec les 3 phases de traitement

La figure 4.10 illustre l'enchaînement séquentiel des trois phases du traitement batch. Le Job *copieSuspensJob* orchestre l'exécution successive du traitement des suspens (Step 1), de la génération du rapport (Step 2) et de l'envoi des notifications (Listener).

4.2.6.4 Avantages de l'architecture modulaire

Ce découpage en trois phases distinctes (traitement, rapport, notification) offre plusieurs avantages :

- **Isolation des responsabilités** : Chaque composant a un périmètre fonctionnel clair
- **Maintenabilité** : Les modifications d'une phase n'impactent pas les autres
- **Extensibilité** : Possibilité d'ajouter de nouvelles phases sans modifier le cœur métier
- **Testabilité** : Chaque composant peut être testé indépendamment

4.2.7 Déploiement et intégration continue

Le module batch est intégré dans une chaîne d'intégration et de déploiement continus complète, orchestrée par GitLab CI/CD et déployée sur Kubernetes. Cette infrastructure garantit la qualité du code, la traçabilité des versions et le déploiement sécurisé sur les différents environnements.

4.2.7.1 Pipeline CI/CD GitLab

Le pipeline, défini dans le fichier *.gitlab-ci.yml*, automatise l'ensemble du cycle de vie de l'application, de la compilation au déploiement en production. Il est structuré en plusieurs stages qui s'enchaînent de manière séquentielle :

1. **Build** : Compilation du code source avec Maven
2. **Test** : Exécution des tests unitaires et d'intégration (JUnit)
3. **Versionning** : Génération automatique du tag de version
4. **Sonar Check** : Analyse de la qualité du code (bugs, vulnérabilités, dette technique)
5. **Build Docker** : Construction de l'image Docker avec Kaniko
6. **Déploiements** : Déploiement progressif sur les environnements (DEV → INT → UAT → PREPROD → PROD)

4.2.7.2 Stratégie de déploiement multi-environnements

Le pipeline implémente une stratégie de promotion d'images graduelles avec des points de contrôle manuels :

- **Déploiement automatique** : L'environnement de développement (DEV) reçoit automatiquement les nouvelles versions validées par les tests
- **Promotion staging** : Passage manuel de l'image du registre scratch vers staging après validation en DEV
- **Déploiements contrôlés** : Les environnements d'intégration (INT), recette (UAT), pré-production et production nécessitent une validation manuelle
- **Promotion stable** : L'image finale est promue vers le registre stable après validation complète en pré-production

Cette approche par *gates* décisionnels permet de valider fonctionnellement et qualitativement chaque version avant son passage en environnement sensible.

4.2.7.3 Contrôle qualité et traçabilité

Le pipeline intègre plusieurs mécanismes de contrôle qualité :

- **Tests automatisés** : Exécution systématique des tests JUnit avant toute construction d'image
- **Analyse Sonar** : Détection précoce des bugs, vulnérabilités et code smells avec Quality Gate
- **Versioning déterministe** : Chaque image Docker est taguée avec une version unique générée automatiquement
- **Déploiement via Kustomize** : Manifests Kubernetes versionnés et patchés automatiquement selon l'environnement cible

Cette industrialisation CI/CD garantit un chemin de livraison reproductible, auditable et sécurisé du code source jusqu'en production.

4.2.8 Orchestration Kubernetes

Le déploiement du module batch sur Kubernetes utilise un **CronJob** pour automatiser l'exécution quotidienne du traitement de reprojection des suspens. Cette ressource Kubernetes garantit une exécution autonome, récurrente et contrôlée du job Spring Batch.

4.2.8.1 Configuration du CronJob

Le CronJob `cjust-batch-copie-suspens` est configuré avec les caractéristiques suivantes :

- **Planification** : Exécution quotidienne à minuit (expression cron : `0 0 * * *`)
- **Timeout** : Durée maximale d'exécution fixée à 10 minutes (`activeDeadlineSeconds=600`)
- **Tolérance de démarrage** : Fenêtre de 5 minutes pour le lancement (`startingDeadlineSeconds=300`)

4.2.8.2 Gestion des ressources et configuration

Le pod batch est déployé avec une allocation de ressources optimisée :

- **CPU** : 250m en request (avec possibilité de bursting)
- **Mémoire** : 500Mi en request et limit
- **Image Docker** : Version taguée automatiquement par le pipeline CI/CD, substituée via Kustomize lors du déploiement

La configuration du batch est externalisée via deux mécanismes Kubernetes :

- **Secrets** : Stockage sécurisé des credentials de base de données (`SPRING_DATASOURCE_*`) et des clés d'accès S3 (`ACCESS_KEY`, `SECRET_ACCESS_KEY`)
- **ConfigMap** : Paramètres applicatifs (périodes de gestion, seuils, configuration email) modifiables sans reconstruction de l'image

4.2.8.3 Avantages de l'approche

Cette architecture d'orchestration présente plusieurs bénéfices opérationnels :

- **Autonomie** : Exécution automatique sans intervention humaine
- **Sécurité** : Gestion des secrets via les mécanismes natifs Kubernetes
- **Flexibilité** : Configuration externalisée permettant des ajustements sans redéploiement
- **Observabilité** : Logs centralisés et métriques disponibles via l'infrastructure Kubernetes

4.2.9 Conclusion Mission 2 : Développement full stack

Cette deuxième mission de développement full stack pour l'application CJUST a porté sur la création d'un batch dédié à la gestion automatisée des comptes anormaux.

4.2.9.1 Composants développés

Les livrables techniques démontrent une couverture complète du cycle de développement :

Composant	Éléments	Technologie
Module Batch	5 classes principales	Spring Batch
Couches logicielles	4 strates	Architecture modulaire
Tests	JUnit	Unitaires + Intégration

Tab. 4.4 : Composants techniques développés pour la Mission 2

4.2.9.2 Réussites techniques accomplies

La mission a atteint ses objectifs :

- Architecture en couches : séparation claire des responsabilités entre orchestration (Batch), métier (Services), données (JPA) et infrastructure (Configuration), garantissant maintenabilité et évolutivité.
- Automatisation complète : traitement batch autonome s'exécutant quotidiennement via CronJob Kubernetes, avec gestion des règles métier complexes de filtrage et d'éligibilité des suspens.
- Pipeline industrialisé : CI/CD GitLab couvrant l'ensemble du cycle (build, test, quality gate Sonar, dockerisation, promotion multi-environnements), assurant qualité et traçabilité.
- Observabilité : système de notifications e-mail avec templates Thymeleaf, logging structuré SLF4J et métriques stockées dans ExecutionContext pour audit complet.

4.2.9.3 Impact organisationnel et métier

L'automatisation du traitement de reprojection des suspens transforme les processus comptables :

- Continuité de service : garantie de reprojection systématique des suspens non apurés à chaque nouvelle période, fiabilisant le démarrage des cycles comptables.
- Conformité renforcée : application déterministe des règles métier avec traçabilité complète des actions (créations, copies, suppressions).

4.3 Conclusion générale

Les deux missions réalisées illustrent la complémentarité entre validation qualité et développement métier pour l'application CJUST. La première mission a établi un framework de tests automatisés BDD (Cypress + Cucumber) avec architecture Page Object Model, couvrant l'ensemble des workflows et profils utilisateurs. La seconde a consisté en le développement d'un module batch Spring Batch pour l'automatisation de la reprojection des suspens, structuré en couches spécialisées (orchestration, métier, données, transverse).

4.3.1 Convergence technologique

Bien que distinctes dans leurs objectifs, les deux missions partagent des principes architecturaux communs :

Aspect	**Mission 1**	**Mission 2**
Pattern structurant	Page Object Model	Architecture en couches
Séparation des responsabilités	Features / Steps / Pages	Batch / Services / Repositories
Gestion centralisée	TestDataManager (Singleton)	Spring Configuration
Observabilité	Reporting Allure	Logging SLF4J + Métriques
Intégration continue	Pipeline GitLab (tests)	Pipeline GitLab (build/deploy)

Tab. 4.5 : Patterns et principes communs aux deux missions

Chapitre 5

Conclusion

Ce stage de six mois au sein de la direction IOS/ICA du Crédit Agricole CIB a été une expérience professionnelle enrichissante et formatrice. Les deux missions réalisées sur l'application CJUST, outil de justification comptable, m'ont permis d'acquérir des compétences techniques solides tout en découvrant les enjeux métier du secteur bancaire.

Mon intégration au sein de l'équipe s'est déroulée dans d'excellentes conditions. Dès le premier jour, j'ai été accueilli par ma tutrice, Madame Vukmirovic Nadežda, qui m'a présenté l'environnement de travail et les équipes. Le site du Crédit Agricole à SQY Park offre un cadre de travail agréable, avec de nombreux espaces verts. L'équipe dirigée par Madame Houeiss Sabine m'a réservé un accueil chaleureux et m'a permis de m'intégrer rapidement dans les projets en cours. L'ambiance collaborative et bienveillante de l'équipe a largement facilité mon apprentissage et ma montée en compétences.

Madame Houeiss Sabine, en tant que manager de l'équipe IOS/ICA, a toujours été disponible et à l'écoute. Grâce à elle, j'ai pu poser toutes les questions nécessaires pour bien comprendre les enjeux du projet ainsi que les démarches bureautiques. Ma tutrice, Madame Vukmirovic Nadežda, a fait preuve d'une grande patience et pédagogie pour m'expliquer les concepts comptables et les fonctionnalités de l'application CJUST. Nos échanges réguliers sur les aspects métier (justification des comptes, périodes comptables, workflows de validation) et techniques (base de données, architecture, interfaces) ont été essentiels pour mener à bien les deux missions.

Nos Tech Leads, Madame Cohen Noémie et Monsieur Duarte Filipe, m'ont transmis les bonnes pratiques de développement et m'ont initié aux technologies modernes de l'écosystème Java/Spring et aux outils DevOps. Leurs conseils en matière d'architecture logicielle, de clean code et de tests automatisés ont enrichi mes compétences techniques. Ce stage m'a permis de développer une expertise sur un large ensemble de technologies : tests automatisés avec Cypress et Cucumber (approche BDD, Pattern Page Object Model), développement backend avec Java 21, Spring Boot, Spring Batch et Spring Data JPA, ainsi que les outils DevOps (GitLab CI/CD, Docker, Kubernetes, Argo CD, SonarQube) et les bases de données PostgreSQL.

Au-delà de la technique, cette expérience m'a aidé à devenir plus autonome dans la gestion de projets, à mieux communiquer avec les équipes et à mieux comprendre le fonctionnement du monde bancaire et comptable.

Je tiens à exprimer ma profonde gratitude à Madame Vukmirovic Nadežda, ma tutrice, pour son accompagnement quotidien, sa patience et ses précieux conseils techniques et métier. Je remercie également Madame Houeiss Sabine, manager de l'équipe IOS/ICA, pour sa confiance, son encadrement et sa disponibilité, ainsi que Madame Cohen Noémie et Monsieur Duarte Filipe, Tech Leads, pour le partage de leur expertise technique et des bonnes pratiques de développement. Je remercie enfin l'ensemble de l'équipe pour leur accueil, leur bienveillance et leur collaboration.

Ce stage de fin d'études représente une étape déterminante dans mon parcours. Les connaissances et compétences acquises durant ces six mois constituent un socle solide pour aborder ma future carrière d'ingénieur en informatique. L'expérience pratique du développement en environnement bancaire, la maîtrise des technologies modernes et la compréhension des enjeux qualité m'ont préparé à intégrer le marché du travail avec confiance et ambition.

Chapitre 6

Glossaire

Ce glossaire définit les termes techniques et métier utilisés dans ce rapport de stage.

API REST
Interface de programmation d'application utilisant le protocole HTTP et les méthodes GET, POST, PUT, DELETE pour permettre la communication entre différents systèmes.

BDD (Behavior Driven Development)
Approche de développement logiciel qui encourage la collaboration entre développeurs, testeurs et parties prenantes métier en utilisant un langage naturel pour décrire le comportement attendu du système.

Cucumber
Framework de test BDD permettant d'écrire des scénarios de test en langage naturel (Gherkin) et de les exécuter automatiquement.

Cypress
Framework moderne de test end-to-end pour les applications web, offrant une interface de développement intuitive et des fonctionnalités avancées de debugging.

E2E (End-to-End)
Type de test qui valide le fonctionnement complet d'une application en simulant les interactions d'un utilisateur réel depuis l'interface jusqu'à la base de données.

Gherkin
Langage de spécification utilisé par Cucumber pour décrire le comportement des fonctionnalités en langage naturel structuré (Given-When-Then).

Kubernetes
Plateforme d'orchestration de conteneurs utilisée pour déployer, gérer et faire évoluer les applications conteneurisées dans l'infrastructure CJUST.

Page Object Model (POM)
Pattern de conception utilisé dans les tests automatisés pour encapsuler les éléments et actions d'une page web dans des classes dédiées, améliorant la maintenabilité du code de test.

PostgreSQL
Système de gestion de base de données relationnelle open source utilisé pour stocker les données de l'application CJUST.

S3 ECS
Service de stockage d'objets utilisé pour l'hébergement des fichiers de l'application CJUST dans l'infrastructure du groupe Crédit Agricole.

Spring Boot
Framework Java utilisé pour développer l'API REST backend de l'application CJUST, facilitant la création d'applications web autonomes.

Spring Batch
Framework Java utilisé pour implémenter les traitements par lots (batch) d'ingestion des données dans l'application CJUST.

Test de non-régression
: Type de test visant à s'assurer qu'une modification du code n'a pas introduit de nouveaux dysfonctionnements dans les fonctionnalités existantes.

Table des figures

Bibliographie

[1] Angular. Angular documentation. https://angular.io/docs, 2025. [Online ; accessed 2025].

[2] Cucumber. Cucumber documentation. https://cucumber.io/docs/, 2025. [Online ; accessed 2025].

[3] Cypress.io. Cypress documentation. https://docs.cypress.io/, 2025. [Online ; accessed 2025].

[4] GitLab. Gitlab ci/cd documentation. https://docs.gitlab.com/ee/ci/, 2025. [Online ; accessed 2025].

[5] Kubernetes. Kubernetes documentation. https://kubernetes.io/docs/, 2025. [Online ; accessed 2025].

[6] Shelex. Allure cypress plugin. https://www.npmjs.com/package/@shelex/cypress-allure-plugin, 2025. [Online ; accessed 2025].

[7] Spring. Spring boot reference documentation. https://spring.io/projects/spring-boot, 2025. [Online ; accessed 2025].