

## PROJET DE FIN D'ÉTUDES

### Développement d'une application de gestion de quiz pour l'évaluation des compétences en agilité

Filière : Ingénierie Informatique et Réseaux (5<sup>e</sup> année)



Organisme d'accueil : CAPGEMINI

Réalisé par : ANIBA SOUFIANE  
Encadrant professionnel : Mme ASSALI FATIMA  
Encadrant pédagogique : Pr OUATIQ AMINA

Année universitaire : 2024/2025

## Dédicaces

Je dédie ce travail, fruit de plusieurs mois d'efforts, de recherches et d'engagement,

**À mes chers parents,**

pour leur amour infini, leurs sacrifices, leurs prières et leur soutien indéfectible dans chaque étape de ma vie. Vous êtes ma plus grande source d'inspiration et de force.

**À mes frères et sœurs,**

pour leur affection, leur complicité et leurs encouragements constants, qui m'ont soutenu dans les moments les plus exigeants.

**À mes grands-parents et à toute ma famille,**

pour leur bienveillance, leurs prières et leur confiance en moi.

**À mes amis,**

pour leur présence précieuse, leur joie et leurs encouragements qui ont rendu ce parcours plus humain et plus agréable.

**Enfin,**

à toutes celles et ceux qui, de près ou de loin, ont cru en moi, m'ont soutenu et motivé tout au long de ce chemin.

*Merci à vous tous.*

---

## Remerciements

J'exprime ma profonde reconnaissance à mon encadrant professionnel chez **Capgemini**, pour la confiance qu'il m'a accordée, ses conseils avisés et son accompagnement tout au long de cette expérience.

Mes sincères remerciements vont également à **toute l'équipe** pour son aide précieuse, sa disponibilité et son soutien constant dans la réalisation de ce projet.

Je tiens à remercier chaleureusement **toute l'équipe technique de Capgemini**, pour leur accueil, leur disponibilité et leur bienveillance, qui ont largement contribué à rendre ce stage formateur et agréable.

Je souhaite également remercier mon encadrant pédagogique pour son suivi rigoureux, ses conseils précieux et son accompagnement tout au long de ces six mois.

Enfin, j'exprime ma gratitude à l'**École Marocaine des Sciences de l'Ingénieur (EMSI)** pour la qualité de l'enseignement reçu durant ces cinq années de formation, qui m'ont permis de mener à bien ce projet.

*À toutes celles et ceux qui, de près ou de loin, ont contribué à cette réussite :  
merci infiniment.*

# Résumé

Le présent rapport est l'aboutissement du travail effectué dans le cadre du projet de fin d'études au sein de **Capgemini** pour l'obtention du diplôme d'Ingénieur en Informatique. L'objectif principal du projet consiste en la conception et le développement d'une application de gestion de quiz pour l'évaluation des compétences en agilité des collaborateurs.

Dans un monde où l'apprentissage continu est essentiel, **QUIZ AGILE** permet aux entreprises d'évaluer et de suivre les compétences de leurs collaborateurs. Intégrée à la plateforme de formation **NEXT**, cette solution facilite l'auto-évaluation, personnalise les parcours de formation et offre des analyses précises aux managers.

L'évaluation des compétences en agilité représente un défi majeur pour les entreprises modernes. Les méthodes traditionnelles sont souvent chronophages, subjectives et difficiles à standardiser. **QUIZ AGILE** répond à cette problématique en proposant une plateforme flexible et évolutive, permettant la création et la gestion de questions de type QCM, la génération de quiz personnalisés, et l'analyse détaillée des résultats.

Durant le déroulement du projet, nous avons commencé par une analyse fonctionnelle et technique des spécifications afin de bien saisir la problématique, puis nous avons entamé l'étape de conception pour finir avec la mise en œuvre. L'application a été développée avec une stack technique moderne comprenant React pour le frontend et Spring Boot pour le backend, avec une approche DevOps intégrant GitLab, Jenkins et Docker.

Pour la mise en place du projet **QUIZ AGILE**, et pour assurer l'agilité, le choix est tombé sur Scrum comme méthodologie de gestion de projet, avec un cycle de développement itératif et incrémental. Cette approche nous a permis de livrer régulièrement des fonctionnalités à valeur ajoutée et d'adapter le développement aux retours des utilisateurs.

**Mots clés :** Gestion de quiz, évaluation des compétences, agilité, formation continue, Spring Boot, React, analyse fonctionnelle, conception, mise en œuvre, Scrum.

# Abstract

This report is the result of the work carried out as part of the end-of-study project at Capgemini, aiming to obtain the Engineering degree in Computer Science. The main objective of the project is the design and development of a quiz management application for assessing employees' agility-related skills.

In a world where continuous learning is essential, **QUIZ AGILE** enables companies to evaluate and monitor the competencies of their employees. Integrated with the **NEXT** training platform, this solution facilitates self-assessment, personalizes learning paths, and provides managers with accurate performance analytics.

Assessing agility skills is a major challenge for modern companies. Traditional methods are often time-consuming, subjective, and difficult to standardize. **QUIZ AGILE** addresses this issue by offering a flexible and scalable platform that allows the creation and management of multiple-choice questions (MCQs), the generation of personalized quizzes, and the detailed analysis of results.

Throughout the project, we began with a functional and technical analysis of the specifications to fully understand the problem. Then, we moved on to the design phase and finally to the implementation. The application was developed using a modern technical stack including **React** for the frontend and **Spring Boot** for the backend, following a **DevOps** approach integrating **GitLab**, **Jenkins**, and **Docker**.

To ensure agility in the development process, the **Scrum** methodology was adopted, enabling an iterative and incremental development cycle. This approach allowed us to deliver value-added features regularly and to adapt development based on user feedback.

**Keywords** : Quiz management, skills assessment, agility, continuous training, Spring Boot, React, functional analysis, design, implementation, Scrum

# Liste des Abréviations

**Table 1 – Liste des principales abréviations utilisées dans ce rapport**

Abréviation	Signification
AJAX	Asynchronous JavaScript and XML (JavaScript et XML Asynchrones)
API	Application Programming Interface (Interface de Programmation d'Application)
CI/CD	Continuous Integration/Continuous Deployment (Intégration/Déploiement Continu)
CRUD	Create, Read, Update et Delete (Créer, Lire, Modifier, Supprimer)
CSS	Cascading Style Sheets (Feuilles de Style en Cascade)
DAO	Data Access Object (Objet d'Accès aux Données)
DOM	Document Object Model
DTO	Data Transfer Object (Objet de Transfert de Données)
HTML	HyperText Markup Language (Langage de Balisage HyperTexte)
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment (Environnement de Développement Intégré)
Java	Plateforme de développement orientée objet
JDBC	Java Database Connectivity (Connectivité Base de Données Java)
JPA	Java Persistence API (API de Persistance Java)
JSON	JavaScript Object Notation (Notation d'Objet JavaScript)
JWT	JSON Web Token
JVM	Java Virtual Machine
MVC	Model View Controller (Modèle Vue Contrôleur)
NPM	Node Package Manager
ORM	Object-Relational Mapping (Mappage Objet-Relationnel)
QCM	Questionnaire à Choix Multiples
RBAC	Role-Based Access Control

---

**Table 1 – Liste des principales abréviations utilisées dans ce rapport**

Abréviation	Signification
REST	Representational State Transfer (Transfert d'État Représentationnel)
RGPD	Règlement Général sur la Protection des Données
SCRUM	Méthodologie de gestion de projet agile
SDLC	Software Development Life Cycle
SGBDR	Système de gestion de bases de données relationnelles
SOLID	Principes de conception orientée objet
SPA	Single Page Application
SQL	Structured Query Language (Langage de Requête Structuré)
UI/UX	User Interface/User Experience (Interface/Expérience Utilisateur)
UML	Unified Modeling Language (Langage de Modélisation Unifié)
URL	Uniform Resource Locator

**Tableau 1 :** Liste des principales abréviations utilisées dans ce rapport

# Table des matières

<b>Remerciements</b>	<b>i</b>
<b>Résumé</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Liste des Abréviations</b>	<b>iv</b>
<b>Liste des figures</b>	<b>xi</b>
<b>Liste des tableaux</b>	<b>xiii</b>
<b>Introduction générale</b>	<b>1</b>
<b>Chapitre 1 : Contexte Général du projet</b>	<b>4</b>
1.1 Introduction . . . . .	4
1.2 Présentation de l'organisme d'accueil . . . . .	4
1.2.1 Groupe Capgemini . . . . .	4
1.2.2 Historique du groupe Capgemini . . . . .	5
1.2.3 Fiche d'identité du groupe Capgemini . . . . .	5
1.2.4 Métiers du groupe Capgemini . . . . .	5
1.2.5 Capgemini TS Maroc . . . . .	6
1.2.6 Fiche d'identité de Capgemini TS Maroc . . . . .	6
1.2.7 Organigramme de Capgemini TS Maroc . . . . .	6
1.2.8 Présentation du DigiCamp . . . . .	8
1.3 Présentation du projet Quiz Agile . . . . .	8
1.3.1 Contexte du projet . . . . .	9
1.3.2 Problématique . . . . .	9
1.3.3 Objectifs du projet . . . . .	10
1.3.4 Population concernée . . . . .	11
1.3.5 Solution proposée . . . . .	11
1.3.6 Conduite et planification du projet . . . . .	12
1.3.7 Méthodologie du travail : Agile & Scrum . . . . .	13



1.4	Conclusion . . . . .	14
<b>Chapitre 2 : Analyse et Spécification des besoins</b>		<b>16</b>
2.1	Introduction . . . . .	16
2.2	Exigences fonctionnelles . . . . .	16
2.2.1	Gestion des utilisateurs . . . . .	16
2.2.2	Gestion des quiz . . . . .	16
2.2.3	Gestion des questions . . . . .	16
2.2.4	Passage des quiz . . . . .	17
2.2.5	Évaluation et résultats . . . . .	17
2.2.6	Analyse et reporting . . . . .	17
2.2.7	Intégration avec la plateforme NEXT . . . . .	17
2.3	Spécifications Non Fonctionnelles . . . . .	17
2.3.1	Performance . . . . .	17
2.3.2	Sécurité . . . . .	18
2.3.3	Disponibilité et fiabilité . . . . .	18
2.3.4	Maintenabilité . . . . .	18
2.3.5	Scalabilité . . . . .	18
2.3.6	Utilisabilité . . . . .	18
2.3.7	Interopérabilité . . . . .	19
2.4	Analyse des Acteurs et des Cas d'Utilisation . . . . .	19
2.4.1	Identification des acteurs . . . . .	19
2.4.2	Diagramme de Cas d'Utilisation . . . . .	20
2.5	Conclusion . . . . .	20
<b>Chapitre 3 : Conception et Mise en place du système</b>		<b>24</b>
3.1	Introduction . . . . .	24
3.2	Architecture du système . . . . .	24
3.2.1	Architecture logicielle . . . . .	25
3.2.2	Architecture de déploiement . . . . .	25
3.3	Modélisation UML . . . . .	25
3.3.1	Diagramme de classes . . . . .	25
3.3.2	Diagramme de paquetages . . . . .	27
3.3.3	Diagramme de séquence . . . . .	28
3.3.4	Diagramme d'activité . . . . .	30
3.3.5	Diagrammes d'état . . . . .	30
3.3.6	Cartographie fonctionnelle . . . . .	30
3.4	Modèle de données . . . . .	33
3.4.1	Modèle conceptuel - Diagramme entité-relation . . . . .	33
3.4.2	Mécanismes d'import/export . . . . .	35

3.4.3	Diagramme de planification du projet . . . . .	36
3.5	Modèle de données . . . . .	36
3.5.1	Modèle conceptuel . . . . .	36
3.5.2	Modèle relationnel . . . . .	36
3.6	Technologies et outils utilisés . . . . .	37
3.6.1	Backend - Spring Boot . . . . .	37
3.6.2	Frontend - React . . . . .	37
3.6.3	Base de données - MySQL . . . . .	37
3.7	Architecture monolithique . . . . .	37
3.8	Conclusion . . . . .	38
<b>Chapitre 4 : Réalisation, Tests et Étude Technique</b>		<b>40</b>
4.1	Introduction . . . . .	40
4.2	Architecture technique du système . . . . .	40
4.2.1	Vue d'ensemble de l'architecture . . . . .	40
4.2.2	Justification des choix architecturaux . . . . .	40
4.3	Environnement de développement . . . . .	41
4.3.1	Configuration de l'environnement . . . . .	41
4.3.2	Structure du projet . . . . .	41
4.4	Implémentation du backend . . . . .	41
4.4.1	Configuration Spring Boot . . . . .	41
4.4.2	Modèle de données . . . . .	42
4.4.3	Services métier . . . . .	42
4.4.4	API REST . . . . .	42
4.5	Implémentation du frontend . . . . .	42
4.5.1	Architecture React . . . . .	42
4.5.2	Gestion d'état . . . . .	42
4.5.3	Composants principaux . . . . .	43
4.6	Tests et validation . . . . .	43
4.6.1	Tests unitaires . . . . .	43
4.6.2	Tests d'intégration . . . . .	43
4.6.3	Tests de performance . . . . .	43
4.7	Déploiement . . . . .	43
4.7.1	Conteneurisation . . . . .	43
4.7.2	Pipeline CI/CD . . . . .	44
4.8	Conclusion . . . . .	44
4.9	Technologies et outils utilisés . . . . .	44
4.9.1	Backend - Spring Boot . . . . .	44
4.9.2	Frontend - React . . . . .	44

4.9.3	Authentification et Sécurité . . . . .	45
4.9.4	Documentation et API . . . . .	46
4.9.5	Base de données . . . . .	47
4.9.6	Système de gestion de dépendances . . . . .	48
4.10	Outils de développement et de test . . . . .	49
4.10.1	Outils de test . . . . .	49
4.10.2	Outils de documentation et modélisation . . . . .	50
4.10.3	Outils de test API et intégration . . . . .	51
4.10.4	Outils DevOps et gestion de version . . . . .	52
4.10.5	Intégration continue et déploiement . . . . .	54
4.10.6	Outils d'analyse de qualité de code . . . . .	55
4.11	Architecture technique globale . . . . .	57
4.11.1	Vue d'ensemble de la stack technologique . . . . .	57
4.11.2	Architecture technique détaillée . . . . .	58
4.12	Architecture générale du système . . . . .	58
4.12.1	Vue d'ensemble architecturale . . . . .	58
4.12.2	Architecture Backend détaillée . . . . .	59
4.12.3	Structure et implémentation du projet Backend . . . . .	59
4.12.4	Structure et implémentation du projet Frontend . . . . .	62
4.12.5	Tests et validation . . . . .	65
4.13	Conclusion . . . . .	66
<b>Chapitre 5 : Présentation des interfaces et résultats</b>		<b>68</b>
5.1	Introduction . . . . .	68
5.2	Interface d'authentification . . . . .	68
5.2.1	Page de connexion . . . . .	68
5.3	Interface du tableau de bord . . . . .	68
5.3.1	Tableau de bord principal . . . . .	69
5.4	Interface administrateur . . . . .	69
5.4.1	Gestion des utilisateurs . . . . .	69
5.4.2	Création d'un utilisateur . . . . .	70
5.5	Interface coach . . . . .	71
5.5.1	Création de questions . . . . .	71
5.5.2	Création d'un quiz . . . . .	72
5.5.3	Paramétrage et publication . . . . .	74
5.6	Interface participant . . . . .	75
5.6.1	Passage d'un quiz . . . . .	75
5.6.2	Résultats d'un quiz . . . . .	76
5.7	Tableaux de bord et analyses . . . . .	77

## TABLE DES MATIÈRES

---

5.7.1	Analyse des performances . . . . .	77
5.7.2	Rapports détaillés . . . . .	77
5.7.3	Gestion des questions . . . . .	78
5.7.4	Analyse des résultats . . . . .	78
5.8	Interface collaborateur . . . . .	79
5.8.1	Accueil collaborateur . . . . .	79
5.8.2	Passage de quiz . . . . .	79
5.8.3	Résultats personnels . . . . .	79
5.9	Fonctionnalités transversales . . . . .	79
5.9.1	Responsivité . . . . .	79
5.9.2	Accessibilité . . . . .	79
5.10	Résultats et performances . . . . .	80
5.10.1	Métriques de performance . . . . .	80
5.10.2	Adoption utilisateur . . . . .	80
5.11	Conclusion . . . . .	80
	<b>Conclusion générale</b>	<b>81</b>
	<b>Annexes</b>	<b>83</b>
	<b>Webographie</b>	<b>85</b>

# Table des figures

1.1	Logo Capgemini . . . . .	4
1.2	Fiche d'identité de Capgemini Technology Services Maroc . . . . .	7
1.3	Organigramme de Capgemini TS Maroc . . . . .	7
1.4	Plateforme Coding Challenge de Capgemini . . . . .	8
1.5	Optimisation des parcours de formation selon les évaluations . . . . .	10
1.6	Organisation des réunions quotidiennes (Daily Scrum) . . . . .	13
2.1	Acteurs principaux du système Quiz Agile . . . . .	19
2.2	Diagramme de cas d'utilisation du système Quiz Agile . . . . .	21
3.1	Architecture globale du système Quiz Agile . . . . .	24
3.2	Diagramme de classes du système Quiz Agile . . . . .	26
3.3	Diagramme de paquetages de l'architecture système . . . . .	27
3.4	Diagramme de séquence du système Quiz Agile . . . . .	28
3.5	Diagramme de séquence - Flux de création de quiz et export PDF . . . . .	29
3.6	Diagramme d'activité : Processus de passage d'un quiz . . . . .	31
3.7	Diagramme d'état : Cycle de vie d'un quiz . . . . .	32
3.8	Diagramme d'état : États d'une tentative de quiz . . . . .	32
3.9	Cartographie fonctionnelle du système Quiz Agile . . . . .	33
3.10	Diagramme entité-relation du système Quiz Agile . . . . .	34
3.11	Mécanismes d'import/export de données . . . . .	35
3.12	Diagramme de GANTT du projet Quiz Agile . . . . .	36
3.13	Comparaison entre architecture monolithique et microservices . . . . .	38
4.1	Architecture technique du système Quiz Agile . . . . .	40
4.2	Logo React . . . . .	45
4.3	Logo JWT (JSON Web Token) . . . . .	46
4.4	Logo PostgreSQL . . . . .	48
4.5	Logo et architecture Maven . . . . .	49
4.6	Logo JUnit 5 - Framework de test Java . . . . .	49
4.7	Logo Mockito - Framework de mocking Java . . . . .	50
4.8	Jest - Framework de test JavaScript pour React . . . . .	51
4.9	Logo PlantUML - Outil de génération de diagrammes UML . . . . .	51
4.10	Logo Postman - Plateforme de test d'API . . . . .	52

4.11	Logo Git - Système de contrôle de version distribué . . . . .	52
4.12	Logo GitLab - Plateforme DevOps complète . . . . .	53
4.13	Logo npm - Gestionnaire de paquets Node.js . . . . .	53
4.14	Logo Docker - Plateforme de conteneurisation . . . . .	54
4.15	Logo Jenkins - Serveur d'intégration continue . . . . .	55
4.16	Logo SonarQube - Plateforme d'analyse de qualité de code . . . . .	56
4.17	Analyse de code avec SonarQube - Dashboard de qualité du projet Quiz Agile	56
4.18	Stack technologique complète du projet Quiz Agile . . . . .	57
4.19	Architecture technique de l'application Quiz Agile . . . . .	58
4.20	Architecture globale de l'application Quiz Agile . . . . .	59
4.21	Structure physique du projet Backend - Arborescence des packages Java . .	62
4.22	Structure physique du projet Frontend - Arborescence React avec SCSS modulaire . . . . .	63
5.1	Interface de connexion de l'application Quiz Agile . . . . .	68
5.2	Interface du tableau de bord principal . . . . .	69
5.3	Interface de gestion des utilisateurs . . . . .	70
5.4	Formulaire de création d'un nouvel utilisateur . . . . .	71
5.5	Interface de création d'une nouvelle question . . . . .	72
5.6	Interface de création d'un nouveau quiz - Vue d'ensemble . . . . .	73
5.7	Interface de sélection des questions pour le quiz . . . . .	74
5.8	Interface de paramétrage et publication de quiz . . . . .	75
5.9	Interface de passage d'un quiz . . . . .	76
5.10	Interface de présentation des résultats . . . . .	76
5.11	Tableau de bord d'analyse des performances . . . . .	77
5.12	Interface de génération de rapports détaillés . . . . .	78
5.13	Architecture technique détaillée du système Quiz Agile . . . . .	83

# Liste des tableaux

1	Liste des principales abréviations utilisées dans ce rapport . . . . .	iv
1	Liste des principales abréviations utilisées dans ce rapport . . . . .	v
1.1	Fiche d'identité du groupe Capgemini . . . . .	5

# Introduction générale

Dans un contexte professionnel en constante évolution, les entreprises font face à un défi majeur : assurer le développement continu des compétences de leurs collaborateurs. L'agilité, en particulier, s'est imposée comme une compétence fondamentale dans le secteur informatique, transformant profondément les méthodes de travail et la gestion de projets. Cette approche, qui privilégie l'adaptabilité, la collaboration et l'amélioration continue, nécessite une maîtrise précise de concepts et de pratiques spécifiques.

Cependant, l'évaluation des compétences en agilité demeure un exercice complexe. Les méthodes traditionnelles d'évaluation présentent plusieurs limitations : elles sont souvent chronophages, subjectives, difficiles à standardiser à l'échelle d'une organisation et ne permettent pas toujours d'identifier avec précision les lacunes à combler. De plus, la collecte et l'analyse des résultats peuvent s'avérer fastidieuses, limitant ainsi la capacité des entreprises à adapter rapidement leurs parcours de formation aux besoins réels de leurs collaborateurs.

Face à ces enjeux, **Capgemini** a identifié le besoin de développer une solution innovante permettant d'évaluer et de suivre efficacement les compétences en agilité de ses collaborateurs. C'est dans ce cadre que s'inscrit le projet **QUIZ AGILE**, une application de gestion de quiz conçue pour standardiser l'évaluation des compétences, faciliter l'analyse des résultats et optimiser les parcours de formation.

**QUIZ AGILE** vise à répondre à plusieurs objectifs stratégiques :

- Standardiser l'évaluation des compétences en agilité au sein de l'entreprise
- Adapter les formations selon les besoins spécifiques identifiés
- Favoriser l'auto-apprentissage des collaborateurs
- Fournir aux managers des tableaux de bord détaillés sur les compétences de leurs équipes
- Créer une base de connaissances évolutive pouvant s'étendre à d'autres domaines de compétences

Pour atteindre ces objectifs, l'application propose un ensemble de fonctionnalités clés : création et gestion de questions de type QCM, génération de quiz personnalisés, diffusion via des URL uniques, analyse détaillée des résultats, et intégration avec la plateforme de formation NEXT existante. Cette intégration est particulièrement importante car



elle permet de créer un écosystème complet allant de l'évaluation des compétences à la formation ciblée.

Ce mémoire retrace l'ensemble des étapes de ce projet. Il est structuré de la manière suivante :

- **Chapitre I** : Présente le contexte général du projet, l'organisation d'accueil Capgemini ainsi que les objectifs poursuivis.
- **Chapitre II** : Détaille l'étude préliminaire, l'analyse des besoins fonctionnels et non-fonctionnels.
- **Chapitre III** : Expose l'analyse fonctionnelle et conceptuelle, avec les cas d'utilisation et la conception du système.
- **Chapitre IV** : Présente l'étude technique, les choix d'architecture, l'environnement de développement et les outils utilisés.
- **Chapitre V** : Illustre la mise en œuvre concrète du système avec les interfaces clés de l'application.

# Chapitre 1

## Contexte Général du projet

## 1.1 Introduction

Ce chapitre a pour objectif de présenter le cadre dans lequel notre projet de fin d'études a été effectué. Nous y trouverons des détails concernant l'entreprise d'accueil, son organisation ainsi que ses activités.

Dans la deuxième partie, nous introduirons le projet **Quiz Agile** en décrivant brièvement sa raison d'être et ses objectifs. Cette mise en contexte permettra de mieux comprendre les enjeux et les défis auxquels le projet répond, ainsi que son importance stratégique pour **Capgemini**.

## 1.2 Présentation de l'organisme d'accueil

### 1.2.1 Groupe Capgemini



Figure 1.1 – Logo Capgemini

**Capgemini** est un leader mondial des services de conseil, de transformation numérique, de technologie et d'ingénierie. Le Groupe est à la pointe de l'innovation pour répondre à l'ensemble des opportunités des clients dans le monde en évolution du cloud, du numérique et des plateformes. Le premier en France et le 6ème à l'échelle mondiale conduisant une stratégie de développement et de diversification qui a donné naissance à une croissance interne et externe. **Capgemini** est responsable de 325 000 personnes dans près de 50 pays. Partenaire stratégique des entreprises pour la transformation de leurs activités en tirant profit de toute la puissance de la technologie, le Groupe est guidé au quotidien par sa raison d'être ; libérer les énergies humaines par la technologie pour un avenir inclusif et durable. Fort de plus de 55 ans d'expérience et d'une grande expertise des différents secteurs d'activité, **Capgemini** est reconnu par ses clients pour répondre à l'ensemble de leurs besoins, de la stratégie et du design jusqu'au management des opérations, en tirant parti des innovations dans les domaines en perpétuelle évolution du cloud, de la data, de l'Intelligence Artificielle, de la connectivité, des logiciels, de l'ingénierie digitale et des plateformes. Le Groupe a réalisé un chiffre d'affaires de 17 milliards d'euros en 2022.

## 1.2.2 Historique du groupe Capgemini

L'entreprise **Capgemini** a été créée en 1967 à Grenoble sous le nom de Sogeti par Serge Kampf. L'entreprise devient un des leaders européens en 1975 à la suite de l'acquisition des entreprises de services informatiques CAP et Gemini Computer Systems. Sogeti devient alors **Capgemini** Sogeti (CGS). **Capgemini** connaît une phase d'expansion entre 1975 et 1979. La restructuration interne de 1989, l'arrivée de l'entreprise sur le marché américain et l'expansion européenne de l'entreprise lui permettent de devenir un des leaders mondiaux de son secteur d'activité.

## 1.2.3 Fiche d'identité du groupe Capgemini

**Table 1.1 – Fiche d'identité du groupe Capgemini**

Groupe Capgemini
<b>Année de création</b> : 1967
<b>Fondateur</b> : Serge KAMPFF
<b>Forme juridique</b> : Société anonyme
<b>PDG</b> : Aiman EZZAT
<b>Effectif</b> : Plus de 340 000 (2022)
<b>Chiffre d'affaires</b> : 18 Milliards d'Euros (2021)

**Tableau 2 : Fiche d'identité du groupe Capgemini**

## 1.2.4 Métiers du groupe Capgemini

**Capgemini** est un leader mondial dans le domaine des services informatiques, offrant une gamme complète de solutions pour répondre aux besoins diversifiés de ses clients. La société propose une expertise approfondie dans plusieurs domaines clés, permettant aux entreprises de maximiser leur performance et de s'adapter aux évolutions technologiques.

Voici un aperçu des principaux services offerts par **Capgemini** :

<b>Consulting Services</b>	Aide à améliorer la performance des organisations, en s'appuyant sur une connaissance approfondie des industries et des processus clients.
<b>Application Services</b>	Conçoit, développe, met en œuvre et maintient des applications informatiques couvrant les activités d'intégration système et de maintenance des applications du Groupe.

<b>Services de technologie et d'ingénierie</b> (Sogeti)	Fournissent une assistance et un soutien aux équipes informatiques internes des entreprises clientes.
<b>L'intégration de systèmes</b> (Technology Services)	Le pôle comprend l'architecture des systèmes d'information (conception/design), l'intégration de systèmes, les développements applicatifs, le pilotage et l'optimisation des systèmes, des réseaux et des données.
<b>L'Infogérance</b> (Outsourcing Services)	Il s'agit de la prise en charge totale ou partielle du système d'information d'un client (ou d'un regroupement de plusieurs clients) et des activités métiers s'y rattachant pour une durée moyenne de cinq ans, mais qui peut aller jusqu'à dix ans, voire davantage.

**Tableau 3 :** Les différents métiers de Capgemini

### 1.2.5 Capgemini TS Maroc

**Capgemini Technology Services** Maroc a été créée en 2007 avec une première agence, pour déménager en janvier 2010 vers les nouveaux locaux de **Casanearshore** avant de lancer en juillet de la même année son activité Infrastructure Management (activité Sogeti). Le choix est justifié par la proximité géographique du Maroc, la qualité reconnue de ses compétences et pour accompagner le pays dans sa stratégie de développement dans le domaine des Nouvelles Technologies de l'Information et de la Communication (TIC). **Capgemini Maroc** est une filiale **Nearshore** ayant le statut de Société Anonyme (SA) est dont le capital s'élève à hauteur de 33 000 000 MAD. Elle est située dans le parc technologique **CasaNearshore** qui a été créé par le gouvernement marocain dans le but d'accueillir des entreprises étrangères et d'y faire travailler la main d'œuvre marocaine issue des meilleures écoles et instituts supérieurs marocains.

### 1.2.6 Fiche d'identité de Capgemini TS Maroc

**Capgemini Technology Services** Maroc est une entité spécialisée dans la fourniture de services technologiques et d'ingénierie. Depuis sa création, l'entreprise s'est imposée comme un partenaire incontournable pour de nombreuses organisations, grâce à son expertise et à son engagement envers l'innovation et la qualité des services.

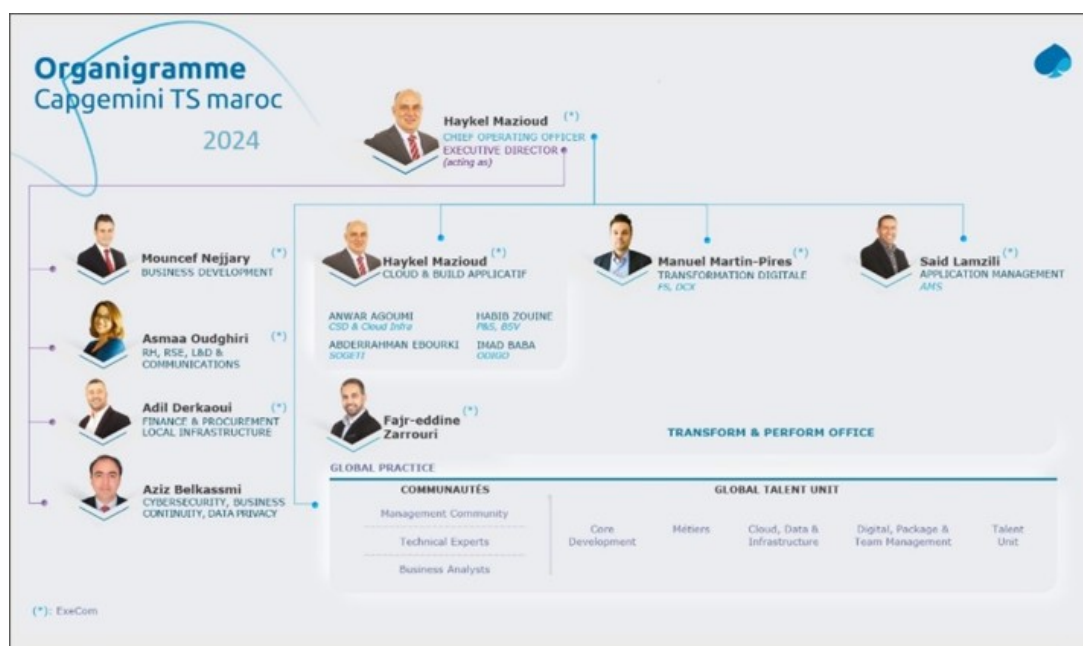
### 1.2.7 Organigramme de Capgemini TS Maroc

Cette fiche présente l'organigramme de **Capgemini TS Maroc**, mettant en avant les rôles clés et les responsabilités au sein de notre structure. Elle illustre la hiérarchie et

Capgemini Technology Services Maroc	
<b>Année de création</b>	2007
<b>Forme juridique</b>	Société Anonyme
<b>PDG</b>	Moncef BENABDESLAM
<b>Effectif</b>	Plus 2700 (2019)

**Figure 1.2 – Fiche d'identité de Capgemini Technology Services Maroc**

les interactions entre les différents départements, tout en soulignant l'importance de la collaboration pour atteindre nos objectifs.



**Figure 1.3 – Organigramme de Capgemini TS Maroc**

Chaque membre joue un rôle crucial dans le succès de nos projets, et cette représentation visuelle facilite la compréhension de notre organisation.

## 1.2.8 Présentation du DigiCamp

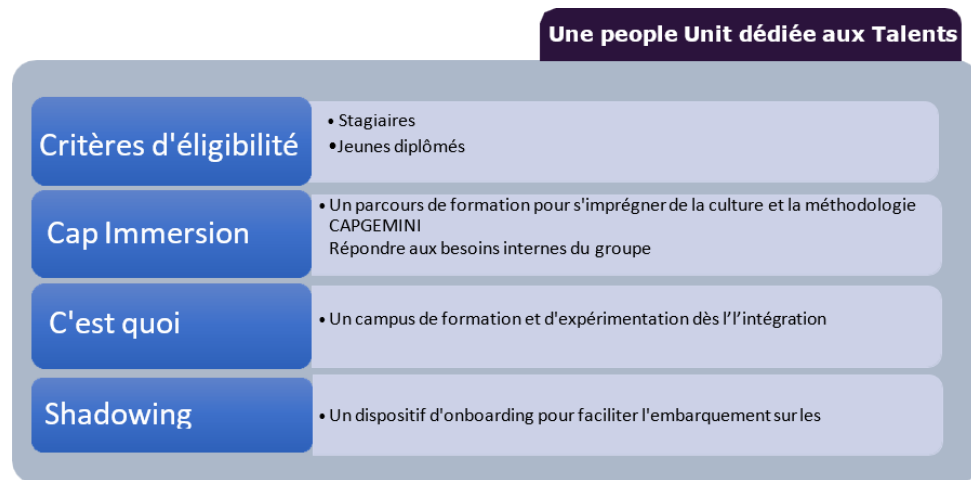


Figure 1.4 – Plateforme Coding Challenge de Capgemini

Dans un contexte où l'évaluation précise des compétences techniques des candidats est cruciale pour le recrutement efficace de talents, la plateforme Coding Challenge de **Capgemini** représente une réponse stratégique aux défis actuels. Cette plateforme innovante offre aux recruteurs un outil puissant pour évaluer objectivement les capacités des candidats à travers des tests techniques rigoureux. En automatisant une partie du processus de présélection, Coding Challenge permet non seulement de réduire les biais subjectifs associés aux méthodes traditionnelles, mais également d'accélérer significativement le temps nécessaire pour identifier les meilleurs talents. Cet outil s'inscrit dans une stratégie plus large visant à renforcer l'attrait de **Capgemini** en tant qu'employeur de choix, capable d'attirer et de retenir les profils les plus qualifiés dans le domaine de l'informatique.

DigiCamp a démarré en 2019 avec 39 collaborateurs qui travaillent sur 13 projets en utilisant diverses technologies. Le but est de permettre au talent de prendre son vol vers le staffing dans les projets en passant par les 3 principaux étapes :

- Cap Immersion.
- Conditionnement en mode projet.
- Formation L&D et accompagnement dans les soft skills, technologies et méthodologie de travail.

Un portfolio ambitieux avec des technologies alignées aux besoins et contenant plus de 23 projets dont 15 applications en développement en 2020.

## 1.3 Présentation du projet Quiz Agile

### 1.3.1 Contexte du projet

Dans un environnement professionnel où l'agilité est devenue un facteur clé de succès, **Capgemini** a identifié le besoin de standardiser et d'optimiser l'évaluation des compétences de ses collaborateurs dans ce domaine. Ce besoin s'inscrit dans une stratégie plus large visant à renforcer la culture agile au sein de l'entreprise et à assurer l'excellence opérationnelle des équipes.

L'agilité, avec ses différentes méthodologies (Scrum, Kanban, SAFe, etc.), ses rôles spécifiques (Scrum Master, Product Owner) et ses pratiques (daily stand-up, sprint planning, retrospectives), représente un corpus de connaissances complexe dont la maîtrise est essentielle pour la réussite des projets informatiques modernes.

Jusqu'à présent, l'évaluation des compétences en agilité au sein de **Capgemini** reposait sur des approches variées et non standardisées : entretiens individuels, observations sur le terrain, ou questionnaires adhoc. Cette diversité d'approches rendait difficile la comparaison des niveaux de compétence entre équipes et limitait la capacité à identifier précisément les besoins de formation.

Par ailleurs, l'entreprise dispose déjà d'une plateforme de formation nommée NEXT, qui propose divers contenus pédagogiques mais ne dispose pas d'un module d'évaluation des compétences intégré. L'articulation entre évaluation et formation représente donc un enjeu majeur pour optimiser les parcours d'apprentissage.

### 1.3.2 Problématique

Dans le contexte spécifique de l'évaluation des compétences en agilité, le projet **QUIZ AGILE** vise à répondre à plusieurs problématiques interconnectées :

- **Comment standardiser l'évaluation des compétences en agilité à l'échelle de l'entreprise ?** L'absence d'un référentiel commun et d'outils standardisés rend difficile la comparaison des niveaux de compétence entre collaborateurs et entre équipes.
- **Comment identifier avec précision les lacunes à combler ?** Les méthodes actuelles ne permettent pas toujours d'analyser finement les domaines spécifiques dans lesquels les collaborateurs ont besoin de renforcer leurs connaissances.
- **Comment optimiser les parcours de formation en fonction des résultats d'évaluation ?** Sans lien direct entre évaluation et formation, il est difficile de proposer des parcours d'apprentissage véritablement personnalisés.





Figure 1.5 – Optimisation des parcours de formation selon les évaluations

- **Comment faciliter le suivi de l'évolution des compétences dans le temps ?**  
L'absence d'historisation des résultats limite la capacité à mesurer les progrès réalisés par les collaborateurs.

### 1.3.3 Objectifs du projet

Le projet **QUIZ AGILE** poursuit plusieurs objectifs stratégiques :

- **Standardiser l'évaluation des compétences en agilité** en proposant un référentiel commun et des outils d'évaluation uniformisés.
- **Adapter la formation selon les besoins identifiés** grâce à une analyse précise des résultats et une intégration avec la plateforme de formation NEXT.
- **Favoriser l'auto-apprentissage des collaborateurs** en leur permettant d'identifier leurs propres lacunes et de suivre leur progression.
- **Fournir aux managers des tableaux de bord détaillés** sur les compétences de leurs équipes pour faciliter la prise de décision.
- **Créer une base de connaissances évolutive** pouvant s'étendre à d'autres domaines de compétences au-delà de l'agilité.
- **Préparer l'infrastructure pour une future intégration avec des fonctionnalités d'IA** qui permettront d'adapter dynamiquement les quiz en fonction des réponses précédentes.

### 1.3.4 Population concernée

Dans le contexte de la transformation digitale, le projet **QUIZ AGILE** s'adresse à un écosystème professionnel diversifié au sein de **Capgemini**, articulé autour de plusieurs catégories de professionnels stratégiques. Les collaborateurs techniques constituent le premier cercle de la population cible, comprenant :

- Les ingénieurs et développeurs impliqués dans les processus de développement logiciel
- Les architectes systèmes et les experts en technologies émergentes
- Les consultants spécialisés en transformation numérique

Les professionnels en management représentent un second périmètre crucial, intégrant :

- Les responsables d'unités opérationnelles
- Les directeurs de projets technologiques
- Les leaders en charge du développement organisationnel
- Les responsables de pratiques métiers

L'écosystème de formation et de développement des compétences constitue un troisième groupe stratégique :

- Les formateurs spécialisés en méthodologies agiles
- Les responsables du learning & development
- Les coaches certifiés en agilité
- Les experts en gestion des compétences

### 1.3.5 Solution proposée

Face aux défis de l'évolution continue des compétences professionnelles, Quiz Agile émerge comme une solution technologique innovante d'évaluation et de développement des compétences en agilité.

La plateforme se caractérise par une approche méthodologique rigoureuse, articulée autour de plusieurs axes stratégiques :

#### Évaluation standardisée des compétences

Quiz Agile propose un dispositif d'évaluation scientifique et objectif, reposant sur :

- Des questionnaires à choix multiples hautement paramétrables
- Un référentiel de compétences normalisé
- Une approche d'évaluation multicritères

#### Architecture fonctionnelle innovante

La solution intègre des fonctionnalités technologiquement avancées :

- Création de quiz par des experts métiers

- Passation de tests en environnement numérique sécurisé
- Analyse approfondie et contextualisation des résultats
- Recommandations de parcours de formation personnalisés
- Tableaux de bord décisionnels pour les managers

### Intégration technologique et écosystémique

Quiz Agile se distingue par sa capacité d'intégration :

- Interconnexion native avec la plateforme de formation NEXT
- Mécanisme d'authentification unique sécurisé
- Synchronisation dynamique des données
- Compatibilité multiplateforme et multiappareils

### Bénéfices organisationnels stratégiques

Quiz Agile vise à générer une valeur ajoutée significative :

- Identification précise et objective des compétences
- Personnalisation des trajectoires de développement professionnel
- Optimisation des investissements en formation
- Accélération de la transformation culturelle et méthodologique

### 1.3.6 Conduite et planification du projet

La planification est cruciale pour le succès du projet. Elle commence par la conception, où les objectifs et les besoins des utilisateurs sont identifiés. Ensuite, le cadrage fonctionnel et technique établit les fonctionnalités requises et les technologies à utiliser. La phase de codage suit, durant laquelle les développeurs créent le produit selon les spécifications. Ensuite, des tests internes sont effectués pour détecter et corriger les erreurs. Une fois ces tests complétés, le produit est soumis à une validation avec le client pour s'assurer qu'il répond aux attentes. Cette approche structurée garantit un développement efficace et réduit les risques de dérives.

### Intégration et formation continue

Lors de mon intégration chez **Capgemini**, j'ai rejoint l'équipe de développement du projet Quiz Agile en tant que Software Engineer Intern. Cette expérience m'a permis de participer activement aux différentes phases de développement de l'application, qui repose sur un écosystème technologique moderne.

Parallèlement, j'ai suivi des formations ciblées sur ces technologies via :

- La plateforme NEXT de **Capgemini**
- La plateforme Udemey

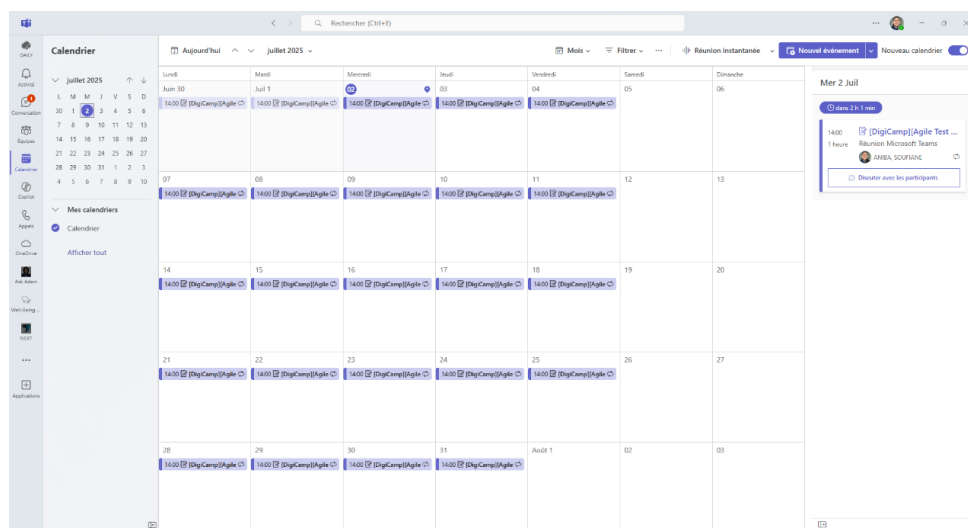
- Des sessions de formation internes
- J'ai également complété des formations obligatoires sur :
- L'éthique professionnelle
  - Le code de conduite de **Capgemini**
  - La sécurité informatique
  - Les politiques de propriété intellectuelle

### 1.3.7 Méthodologie du travail : Agile & Scrum

Le projet **Quiz Agile** a été développé en suivant rigoureusement la méthodologie Agile Scrum, permettant :

- Un développement itératif et incrémental
- Une adaptation continue aux besoins
- Une livraison rapide de valeur métier

Les réunions quotidiennes (Daily Scrum) jouent un rôle central dans cette méthodologie, favorisant la communication et la coordination de l'équipe.



**Figure 1.6 – Organisation des réunions quotidiennes (Daily Scrum)**

Ces réunions quotidiennes de synchronisation permettent à l'équipe de partager les avancées, identifier les blocages et coordonner les efforts pour atteindre les objectifs du sprint.

## 1.4 Conclusion

Ce premier chapitre a permis de dresser un panorama de **Capgemini**, de ses activités et de ses engagements, tout en soulignant le rôle central de l'innovation dans ses projets de transformation numérique.

L'analyse du contexte a mis en évidence les limites des processus actuels d'évaluation des compétences en agilité : lenteur, subjectivité, difficulté de standardisation et manque d'outils d'analyse. Ces constats révèlent la nécessité de disposer d'une solution plus rapide, objective et accessible.

Le projet **QUIZ AGILE** s'inscrit dans cette perspective en proposant une application web intégrée utilisant des technologies modernes. Cette solution vise à améliorer l'évaluation des compétences, faciliter l'adaptation des formations et fournir des outils d'analyse performants, tout en offrant une alternative innovante et efficace aux processus traditionnels.

# Chapitre 2

## Analyse et Spécification des besoins

## 2.1 Introduction

La phase de conception et de modélisation constitue une étape cruciale dans le cycle de développement du projet Quiz Agile. Elle permet de traduire les besoins identifiés lors de l'analyse préliminaire en une architecture technique et fonctionnelle cohérente. Ce chapitre présente de manière détaillée l'ensemble des travaux de conception réalisés, depuis l'analyse des besoins jusqu'aux choix d'architecture technique. Nous y aborderons successivement l'analyse des exigences fonctionnelles et non fonctionnelles, l'identification des acteurs du système, la modélisation conceptuelle à travers différents diagrammes UML, et enfin les choix techniques qui ont guidé l'implémentation du projet. Cette démarche méthodique vise à garantir que le système final réponde parfaitement aux attentes des utilisateurs tout en s'inscrivant dans une architecture évolutive et maintenable.

## 2.2 Exigences fonctionnelles

Les exigences fonctionnelles du projet Quiz Agile ont été établies à partir des besoins exprimés par les parties prenantes et des objectifs stratégiques de l'entreprise. Ces exigences définissent les fonctionnalités que le système doit offrir pour répondre aux attentes des utilisateurs.

### 2.2.1 Gestion des utilisateurs

- Authentification des utilisateurs avec différents rôles (Administrateur, Coach, Collaborateur)
- Gestion des profils utilisateurs (création, modification, suppression)
- Attribution et gestion des droits d'accès selon les rôles

### 2.2.2 Gestion des quiz

- Création de quiz par les coachs avec titre, description et paramètres
- Configuration des quiz (durée, nombre de tentatives autorisées, seuil de réussite)
- Importation et exportation de questions au format standardisé
- Catégorisation des quiz par thématique et niveau de difficulté
- Génération d'URL de partage pour diffusion aux collaborateurs

### 2.2.3 Gestion des questions

- Création de questions de différents types (QCM, vrai/faux, réponses multiples)
- Association d'options de réponse avec indication des réponses correctes
- Attribution d'un niveau de difficulté et de mots-clés aux questions

- Réutilisation des questions dans différents quiz
- Suggestion intelligente de questions basée sur l'historique et les thématiques

#### **2.2.4 Passage des quiz**

- Interface intuitive pour répondre aux questions
- Chronomètre pour respecter la durée impartie
- Navigation entre les questions
- Possibilité de marquer des questions pour y revenir plus tard

#### **2.2.5 Évaluation et résultats**

- Calcul automatique des scores
- Affichage des réponses correctes et des explications après soumission
- Génération de rapports détaillés pour les participants
- Attribution de badges et récompenses selon les performances
- Historique des tentatives et progression dans le temps

#### **2.2.6 Analyse et reporting**

- Tableau de bord pour les coachs avec statistiques globales
- Analyse des performances par équipe, thématique ou question
- Identification des points forts et des axes d'amélioration
- Exportation des résultats pour analyse externe
- Recommandations personnalisées de formation

#### **2.2.7 Intégration avec la plateforme NEXT**

- Authentification unique (SSO) entre Quiz Agile et NEXT
- Synchronisation des données utilisateurs
- Intégration des quiz dans les parcours de formation NEXT
- Partage des résultats avec la plateforme NEXT

### **2.3 Spécifications Non Fonctionnelles**

Les exigences non fonctionnelles définissent les critères de qualité et les contraintes techniques que le système doit respecter.

#### **2.3.1 Performance**

- Temps de réponse inférieur à 2 secondes pour les opérations courantes



- Capacité à gérer simultanément jusqu'à 200 utilisateurs actifs
- Optimisation des requêtes de base de données pour minimiser la latence
- Mise en cache des données fréquemment accédées

### **2.3.2 Sécurité**

- Protection des données personnelles conformément au RGPD
- Authentification sécurisée avec support de l'authentification à deux facteurs
- Chiffrement des données sensibles en transit et au repos
- Journalisation des actions critiques pour audit
- Protection contre les attaques courantes (injection SQL, XSS, CSRF)

### **2.3.3 Disponibilité et fiabilité**

- Disponibilité du système 99,9% du temps (hors maintenance planifiée)
- Sauvegarde quotidienne des données avec rétention de 30 jours
- Plan de reprise d'activité avec un RTO (Recovery Time Objective) de 4 heures
- Mécanismes de détection et de notification des erreurs

### **2.3.4 Maintenabilité**

- Architecture modulaire facilitant l'évolution du système
- Documentation complète du code et de l'architecture
- Tests automatisés couvrant au moins 80% du code
- Respect des standards de codage et des bonnes pratiques

### **2.3.5 Scalabilité**

- Architecture permettant une montée en charge horizontale
- Conception adaptée à une future migration vers les microservices
- Séparation claire des responsabilités pour faciliter la distribution

### **2.3.6 Utilisabilité**

- Interface utilisateur intuitive et responsive
- Support multilingue (français et anglais dans un premier temps)
- Accessibilité conforme aux normes WCAG 2.1 niveau AA
- Compatibilité avec les principaux navigateurs (Chrome, Firefox, Safari, Edge)
- Support des appareils mobiles et tablettes

### 2.3.7 Interopérabilité

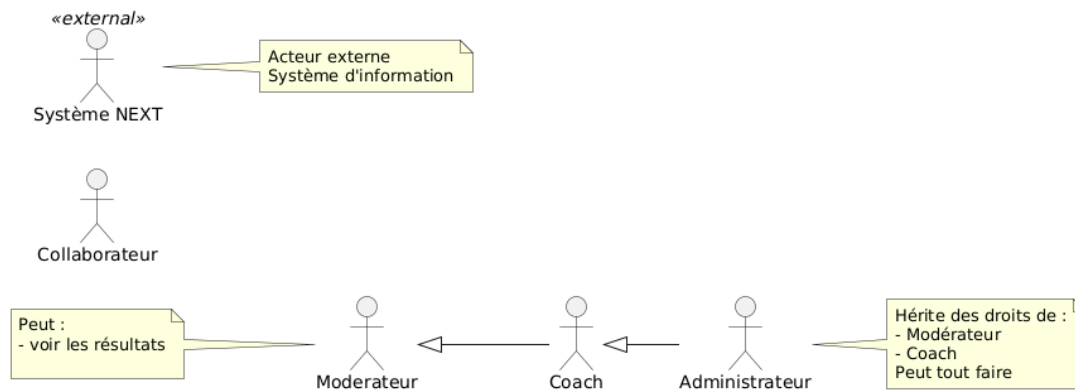
- API RESTful documentée pour l'intégration avec des systèmes tiers
- Support des formats d'échange standards (JSON, XML)

## 2.4 Analyse des Acteurs et des Cas d'Utilisation

L'identification des acteurs et de leurs rôles est une étape cruciale dans la conception d'une application. Elle permet de comprendre les besoins spécifiques de chaque utilisateur et de créer une expérience utilisateur optimale. En définissant clairement les rôles et les fonctionnalités associées à chaque type d'utilisateur, on peut garantir que l'application répond aux attentes de chacun et facilite les interactions entre les différents acteurs.

### 2.4.1 Identification des acteurs

Un acteur est l'idéalisation d'un rôle joué par une personne, un matériel ou un logiciel qui interagit directement avec le système en question. Il peut consulter et / ou modifier directement l'état du système en émettant ou recevant des messages susceptibles d'être porteurs de données.



**Figure 2.1 – Acteurs principaux du système Quiz Agile**

Dans le cadre du projet QUIZ AGILE, plusieurs acteurs clés interagissent avec l'application mobile et la plateforme web. Voici une description de chacun d'eux :

#### Administrateur

- Responsable de la configuration globale du système
- Gère les utilisateurs et leurs droits
- Supervise l'ensemble des activités sur la plateforme
- Accède aux statistiques globales et aux journaux d'audit

### Coach

- Crée et gère les quiz et les questions
- Configure les paramètres des évaluations
- Analyse les résultats des collaborateurs
- Identifie les besoins en formation
- Génère des rapports d'analyse

### Collaborateur

- Passe les quiz d'évaluation
- Consulte ses résultats et son historique
- Suit sa progression dans le temps
- Reçoit des recommandations personnalisées

### Système NEXT

- Échange des données avec Quiz Agile
- Authentifie les utilisateurs via SSO
- Intègre les quiz dans les parcours de formation
- Récupère les résultats des évaluations

## 2.4.2 Diagramme de Cas d'Utilisation

Le diagramme de cas d'utilisation est un outil d'analyse puissant qui permet de visualiser les différents cas d'utilisation de l'application et les fonctionnalités associées aux interactions avec les acteurs, offrant ainsi une compréhension globale des besoins des utilisateurs et des objectifs du système.

Le diagramme de cas d'utilisation ci-dessus illustre les principales interactions entre les acteurs et le système Quiz Agile. Le diagramme met en évidence les actions principales accessibles aux différents acteurs après authentification. Cette modélisation permet de comprendre les interactions et les périmètres de responsabilité au sein de l'application Quiz Agile.

## 2.5 Conclusion

Ce chapitre a présenté l'analyse détaillée des besoins fonctionnels et non-fonctionnels du projet Quiz Agile. L'identification claire des acteurs et de leurs rôles, ainsi que la modélisation des cas d'utilisation, fournissent une base solide pour la conception et le développement du système. Cette approche méthodique garantit que la solution finale

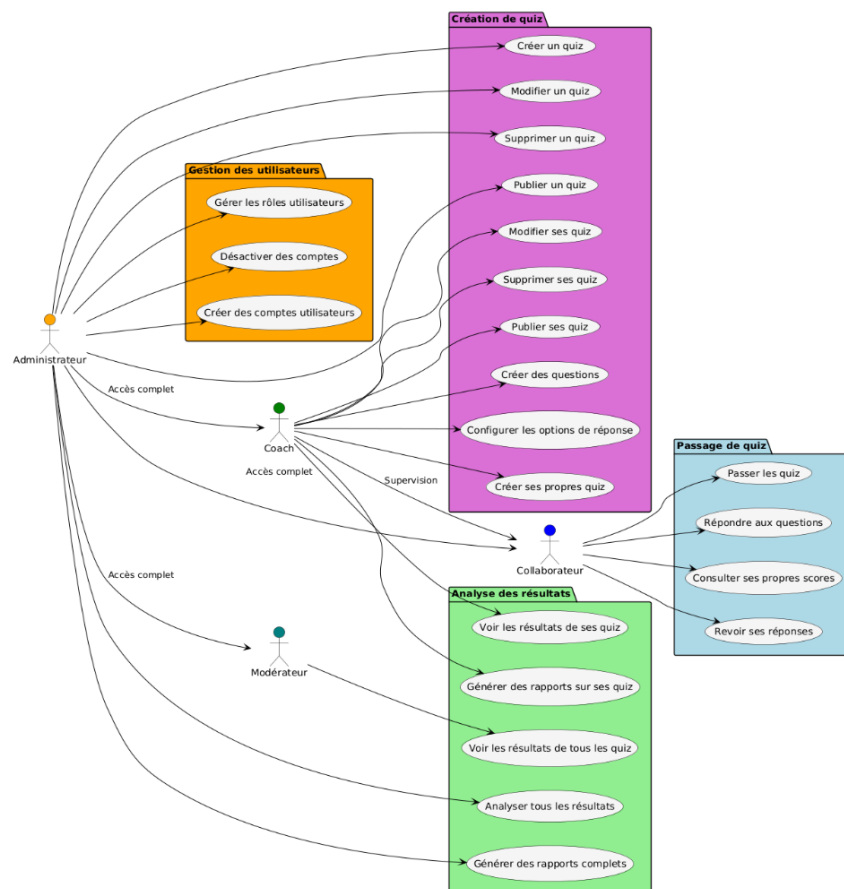


Figure 2.2 – Diagramme de cas d'utilisation du système Quiz Agile

répondra aux attentes de tous les utilisateurs tout en respectant les contraintes techniques et de qualité requises.

# Chapitre 3

## Conception et Mise en place du système

## 3.1 Introduction

La conception constitue une étape clé entre la spécification des besoins et l'implémentation. Elle permet de représenter, sous forme de modèles, la structure et le comportement du système afin de garantir sa cohérence, sa robustesse et sa maintenabilité.

Dans le cadre du projet **QUIZ AGILE**, la conception se décline en deux volets complémentaires :

- la **modélisation dynamique** décrivant les comportements, scénarios et flux de données (diagrammes de séquence et d'activités)
- la **modélisation statique** mettant en avant la structure du système (diagramme de classes et modèle relationnel) ainsi que l'architecture logicielle de déploiement

## 3.2 Architecture du système

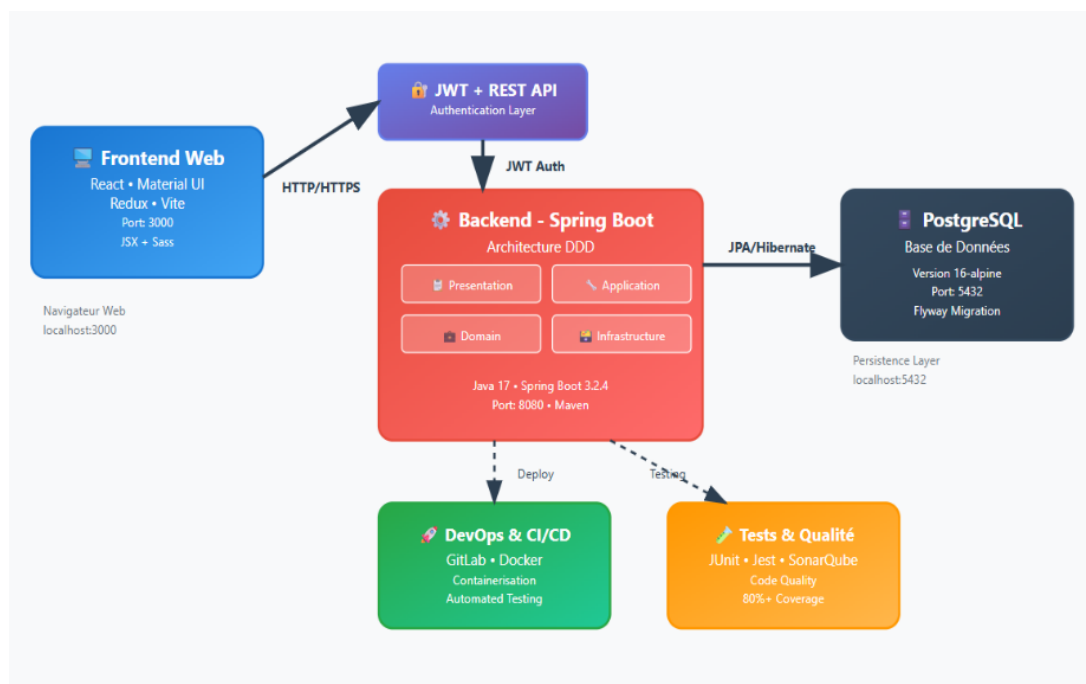


Figure 3.1 – Architecture globale du système Quiz Agile

L'architecture globale du système Quiz Agile illustre l'organisation complète des composants techniques, depuis l'interface utilisateur frontend jusqu'à la base de données, en passant par la couche d'authentification, les services backend et les outils de déploiement et de test.

### 3.2.1 Architecture logicielle

L'architecture logicielle du système Quiz Agile repose sur une approche en couches (layered architecture) qui garantit une séparation claire des responsabilités et facilite la maintenance et l'évolutivité du système.

#### Architecture 3-tiers

Le système adopte une architecture 3-tiers classique :

- **Couche Présentation** : Interface utilisateur développée en React
- **Couche Métier** : Logique applicative implémentée avec Spring Boot
- **Couche Données** : Base de données relationnelle MySQL

#### Patron d'architecture MVC

L'application respecte le patron Modèle-Vue-Contrôleur (MVC) :

- **Modèle** : Entités JPA et services métier
- **Vue** : Composants React pour l'interface utilisateur
- **Contrôleur** : REST Controllers Spring Boot

### 3.2.2 Architecture de déploiement

L'architecture de déploiement prévoit une séparation entre l'environnement de développement et l'environnement de production, avec des mécanismes de CI/CD pour automatiser les déploiements.

## 3.3 Modélisation UML

La modélisation UML (Unified Modeling Language) constitue une étape essentielle de la conception du système Quiz Agile. Elle permet de représenter visuellement la structure, le comportement et les interactions du système à travers différents types de diagrammes.

### 3.3.1 Diagramme de classes

Le diagramme de classes représente la structure statique du système en montrant les classes, leurs attributs, leurs méthodes et les relations entre elles. Cette modélisation UML offre une compréhension approfondie de la structure interne du code et de ses interactions complexes.



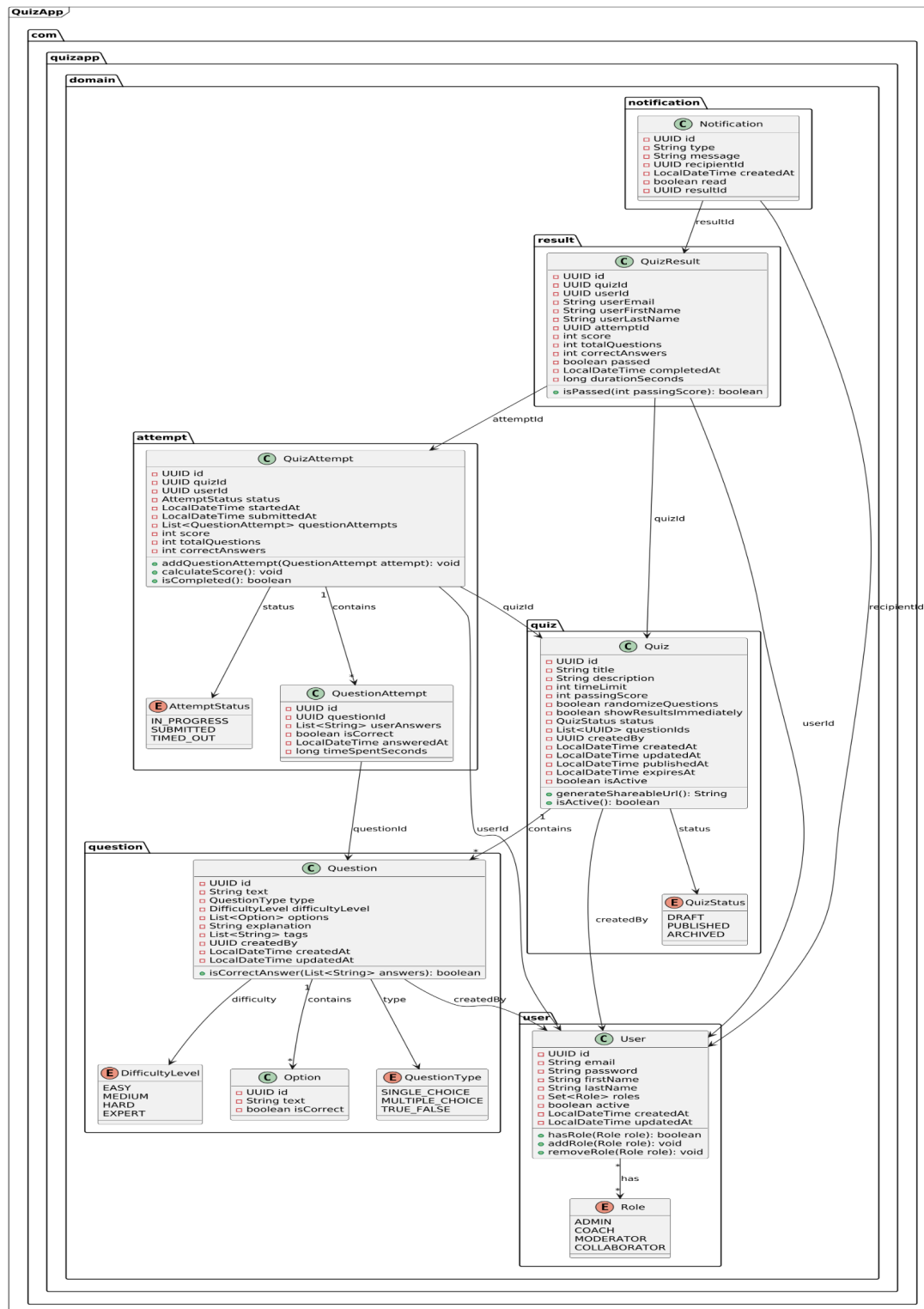


Figure 3.2 – Diagramme de classes du système Quiz Agile

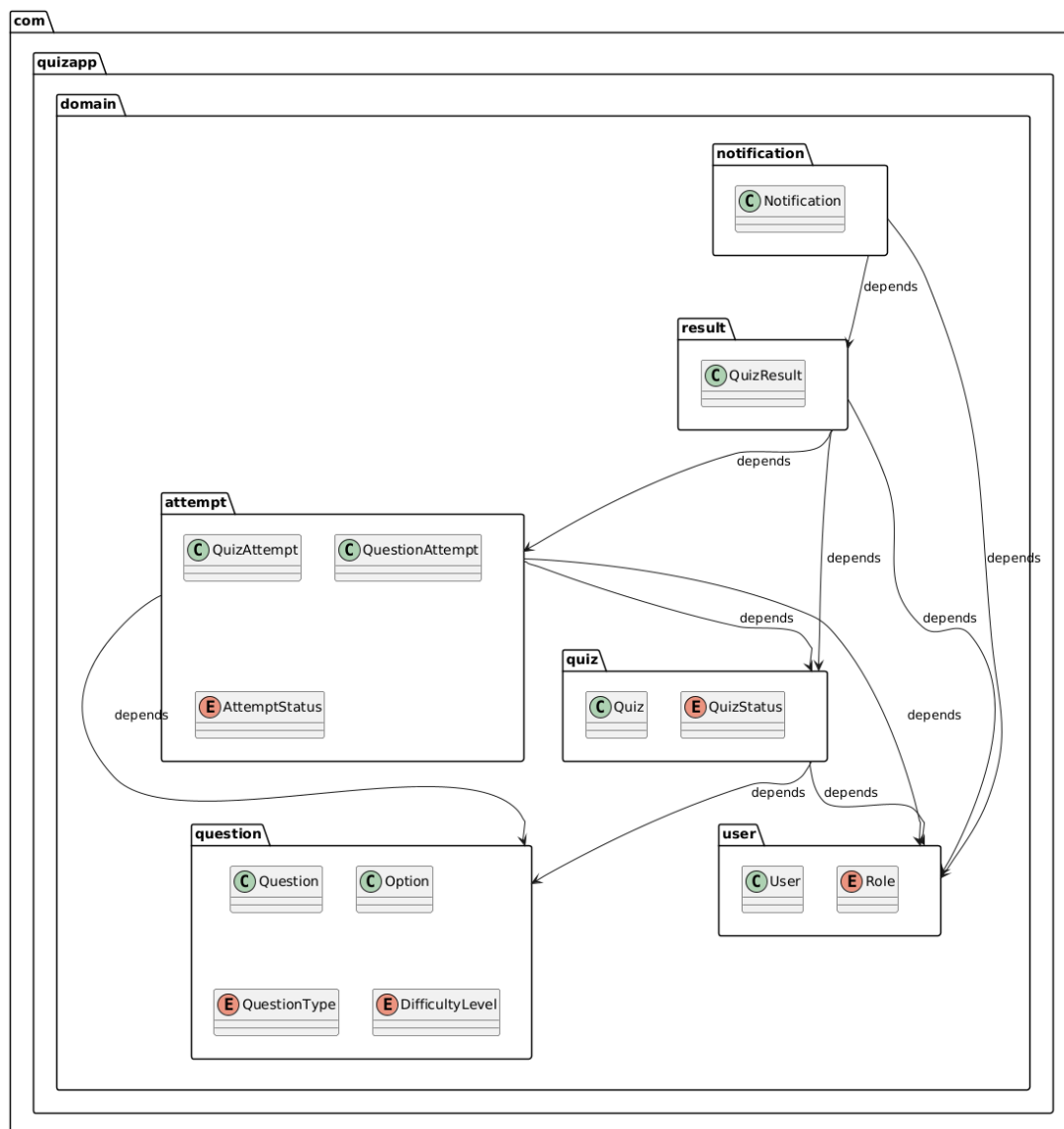
Le diagramme représente un système de quiz en ligne avec 6 packages principaux organisés dans le cadre "Agile Quiz" :

1. **Model** : Contient les entités principales (User, Quiz, Question, QuizAttempt)
2. **Repository** : Interfaces d'accès aux données utilisant Spring Data JPA

3. **Service** : Logique métier et traitements des données
4. **Controller** : Endpoints REST pour l'API
5. **DTO** : Objets de transfert de données pour les échanges API
6. **Config** : Configuration de sécurité et JWT

### 3.3.2 Diagramme de paquetages

Le diagramme de paquetages offre une vue d'ensemble de l'organisation structurale du système Quiz Agile. Il présente les différents modules et leurs relations, facilitant la compréhension de l'architecture globale.

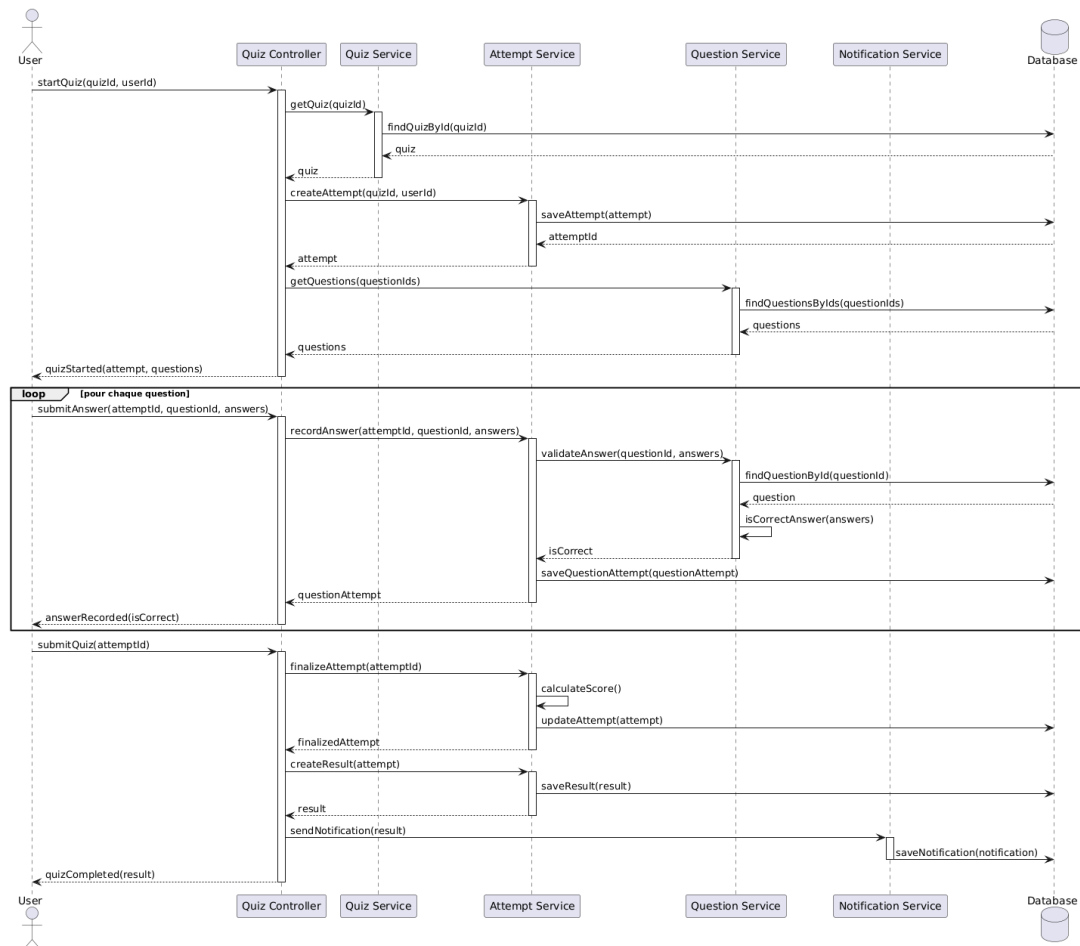


**Figure 3.3** – Diagramme de paquetages de l'architecture système

Cette représentation modulaire favorise une approche de développement claire et maintient une séparation logique entre les différentes responsabilités du système.

### 3.3.3 Diagramme de séquence

Le diagramme de séquence est un diagramme d'interaction qui détaille le déroulement temporel des échanges entre les différents objets du système. Il représente la chronologie des appels de méthodes et des interactions pour un scénario d'usage particulier.

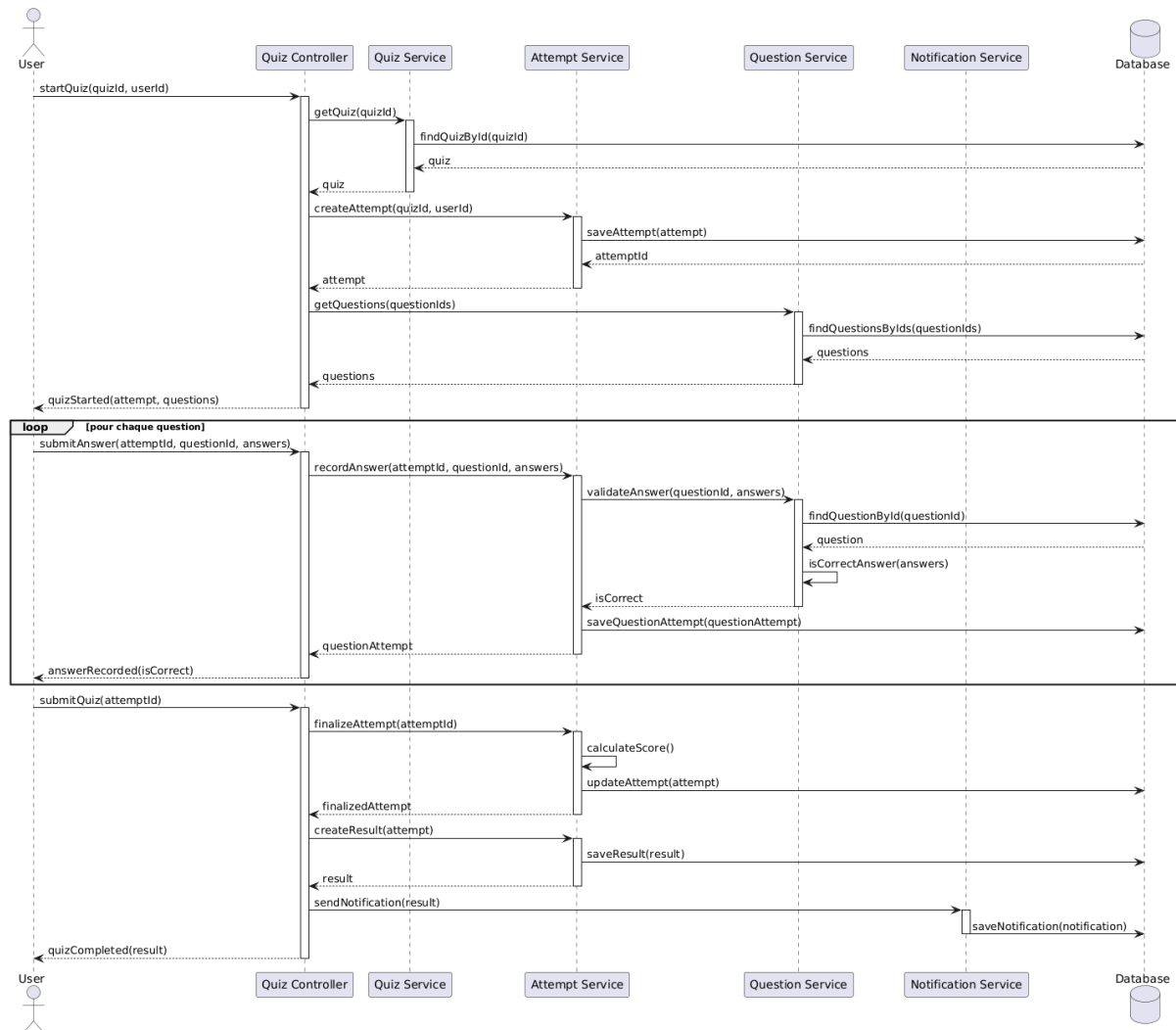


**Figure 3.4 – Diagramme de séquence du système Quiz Agile**

Ce diagramme illustre les interactions principales lors du processus de création et de passage d'un quiz, mettant en évidence la communication entre les couches de l'architecture.

#### Diagramme de séquence - Flux de création et export de quiz

Ce diagramme de séquence détaillé illustre spécifiquement le flux de création d'un quiz et le processus d'export PDF, mettant en évidence les différents composants de notre solution fondée sur l'architecture hexagonale.



**Figure 3.5 – Diagramme de séquence - Flux de création de quiz et export PDF**

Ce diagramme montre les interactions entre :

- **React Frontend** : Interface utilisateur pour la création de quiz
- **QuizController** : Point d'entrée REST pour les requêtes de création
- **QuizService** : Logique métier pour la gestion des quiz
- **QuizMapper** : Conversion entre entités et DTOs
- **Quiz Entity** : Modèle de données principal
- **QuizRepository** : Accès aux données PostgreSQL
- **PdfExportService** : Service d'export en format PDF

Le flux illustre les étapes depuis la soumission d'un nouveau quiz jusqu'à la génération du fichier PDF de résultats, démontrant la séparation claire des responsabilités dans notre architecture hexagonale.

### 3.3.4 Diagramme d'activité

Le diagramme d'activité représente le flux de processus et les décisions dans le système. Il modélise les activités, les transitions et les points de décision pour illustrer le comportement dynamique du système.

Ce diagramme détaille le processus complet de passage d'un quiz, depuis la sélection du quiz par l'utilisateur jusqu'à l'affichage des résultats finaux. Il met en évidence les points de décision critiques, les boucles de traitement des questions, et les différents chemins possibles selon les actions de l'utilisateur.

### 3.3.5 Diagrammes d'état

Les diagrammes d'état modélisent les différents états que peuvent prendre les objets du système et les transitions entre ces états en réponse aux événements.

#### Diagramme d'état : Statut du Quiz

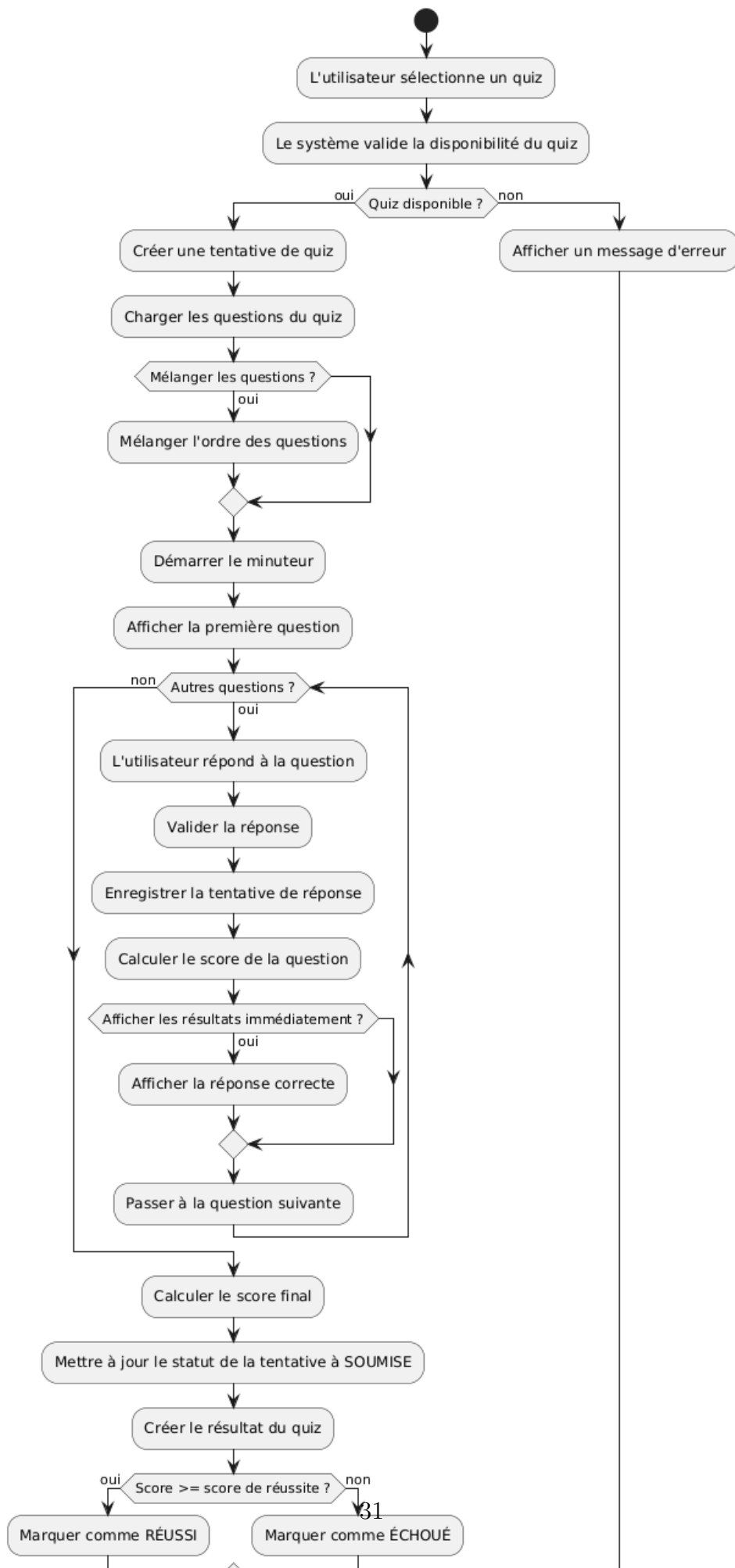
Ce diagramme illustre les différents états par lesquels passe un quiz durant son cycle de vie : depuis sa création en mode "Brouillon", sa publication, jusqu'à son archivage final.

#### Diagramme d'état : Tentative de Quiz

Ce diagramme modélise les états d'une tentative de quiz, de sa création jusqu'à son état final (soumise ou expirée), en passant par l'état "en cours" où l'utilisateur répond aux questions.

### 3.3.6 Cartographie fonctionnelle

La cartographie fonctionnelle présente une vue d'ensemble des principales fonctionnalités du système Quiz Agile et leurs relations dans une perspective de conception.



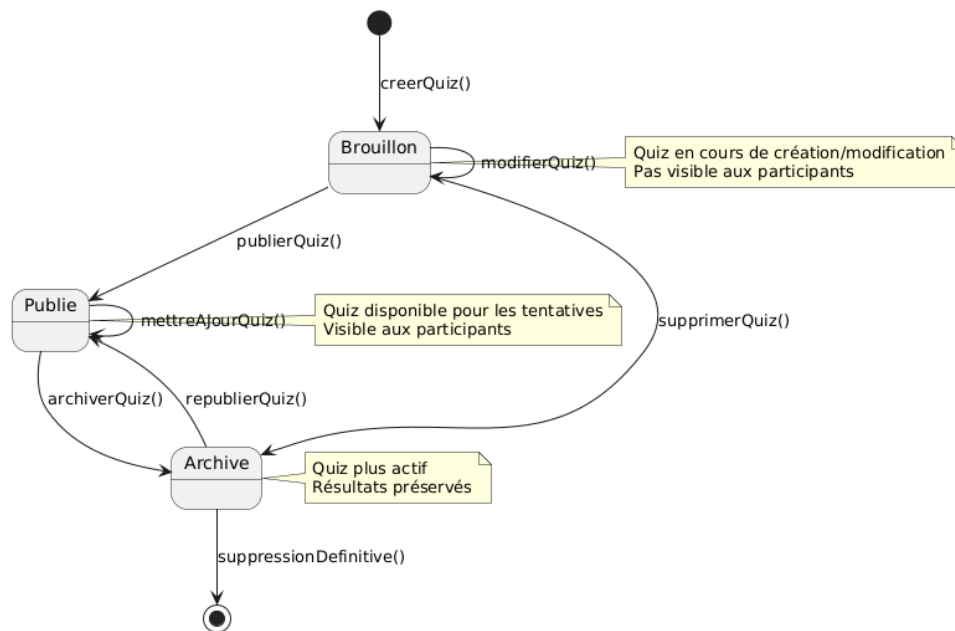


Figure 3.7 – Diagramme d'état : Cycle de vie d'un quiz

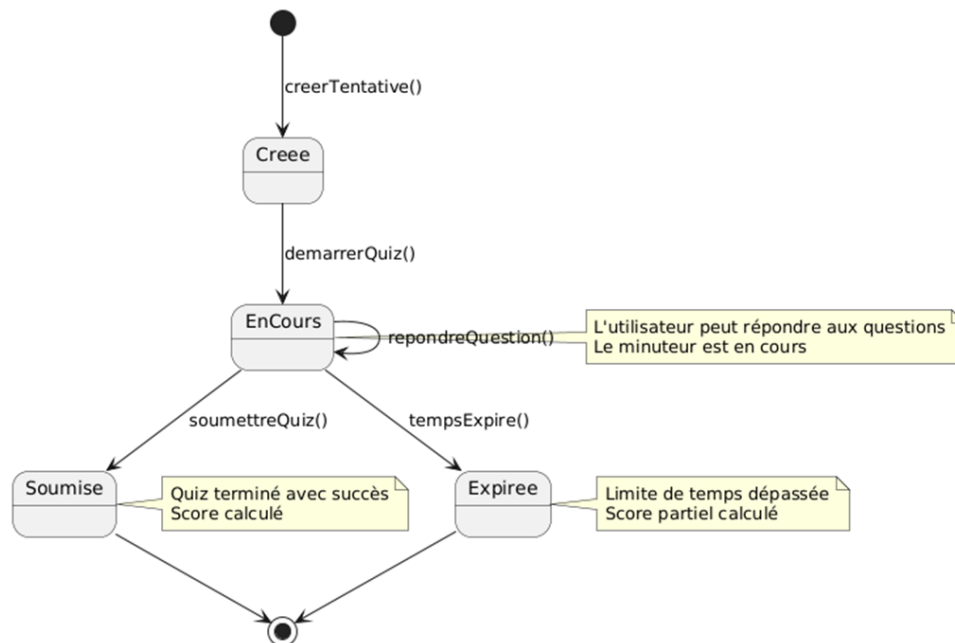
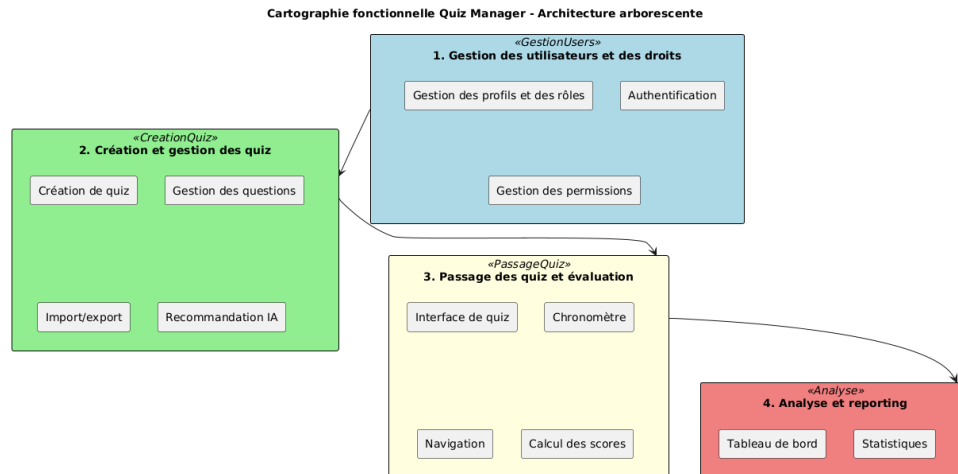


Figure 3.8 – Diagramme d'état : États d'une tentative de quiz



**Figure 3.9 – Cartographie fonctionnelle du système Quiz Agile**

Cette cartographie met en évidence les quatre grands modules fonctionnels du système :

- Module de gestion des utilisateurs et des droits
- Module de création et gestion des quiz
- Module de passage des quiz et évaluation
- Module d’analyse et de reporting

## 3.4 Modèle de données

### 3.4.1 Modèle conceptuel - Diagramme entité-relation

Le diagramme entité-relation (ERD) présente la structure de données de notre système Quiz Agile, illustrant les entités principales, leurs attributs et les relations qui les lient. Cette modélisation conceptuelle constitue le fondement de notre base de données et assure la cohérence des informations stockées.

#### Explications des dépendances

**Quiz dépend de :**

- **User** (pour le créateur du quiz) - Chaque quiz est créé par un utilisateur spécifique
- **Question** (pour les questions incluses) - Un quiz contient une collection de questions

**Attempt dépend de :**

- **Quiz** (quel quiz est tenté) - Chaque tentative est liée à un quiz spécifique
- **User** (qui tente) - Identifie l’utilisateur qui passe le quiz
- **Question** (pour validation) - Permet de valider les réponses données

**Result agrège les données de :**

- **Quiz** (paramètres) - Récupère les paramètres de notation du quiz
- **User** (participant) - Identifie le participant pour lequel le résultat est généré



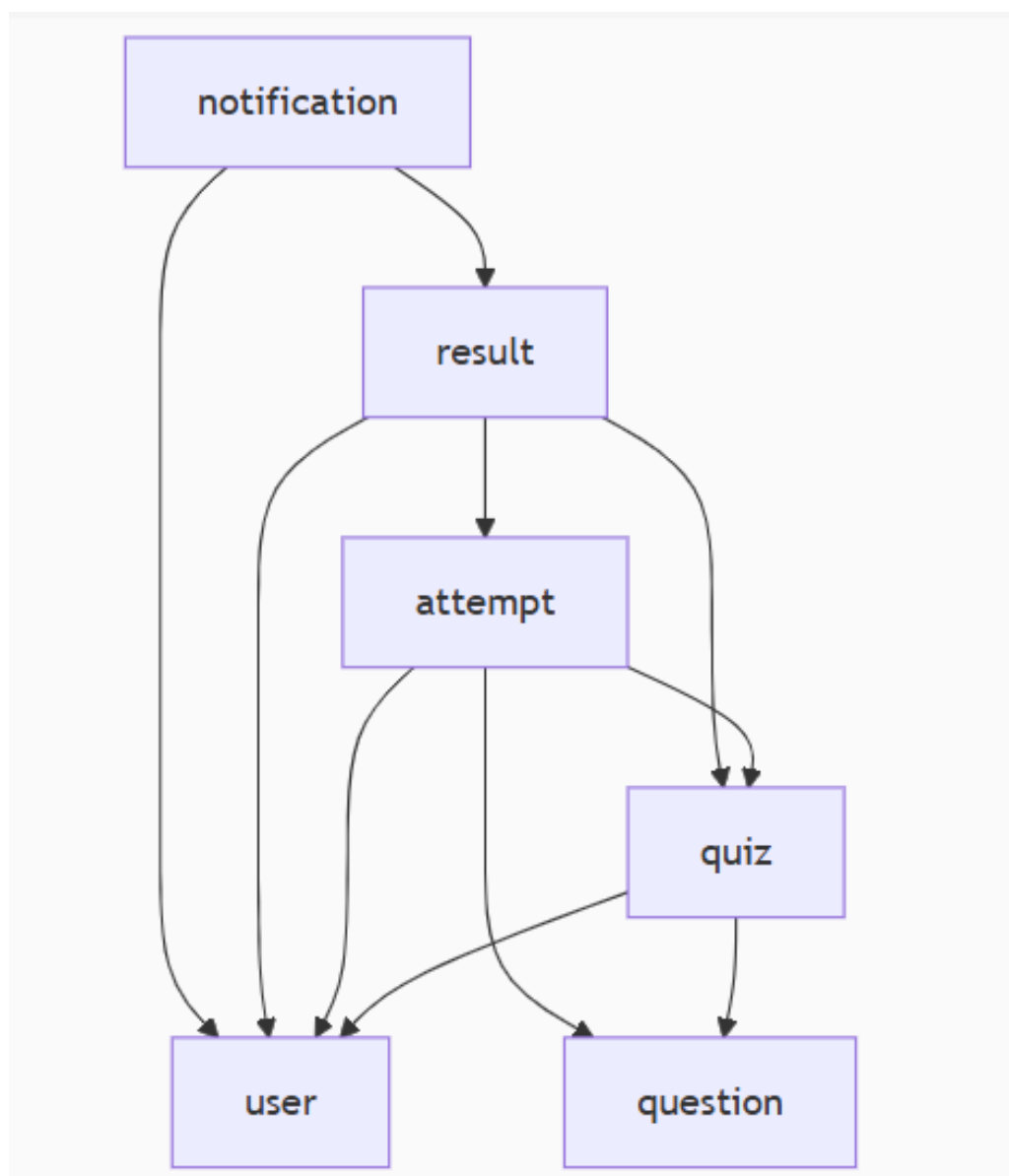


Figure 3.10 – Diagramme entité-relation du système Quiz Agile

- **Attempt** (détails) - Compile les détails de la tentative pour calculer le score

**Notification utilise :**

- **User** (destinataire) - Définit qui recevra la notification
- **Result** (contenu) - Utilise les résultats pour personnaliser le contenu de la notification

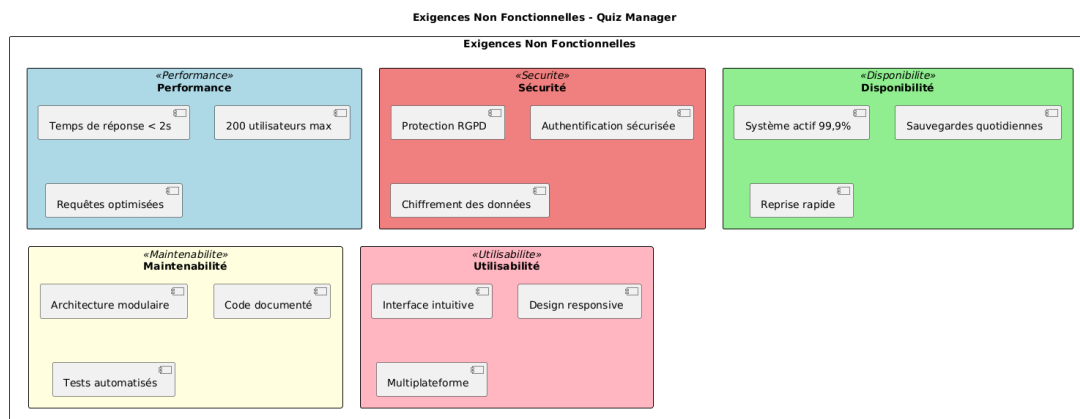
### Analyse Architecturale

- **Architecture** : Hexagonale (Domain-Centric) - Cette approche place le domaine métier au centre, isolant la logique business des détails techniques
- **Niveau de couplage** : Modulaire - Les composants sont faiblement couplés, facilitant la maintenance et l'évolution du système

Cette architecture garantit une séparation claire des responsabilités et facilite l'évolutivité du système, permettant d'ajouter de nouvelles fonctionnalités sans impacter l'architecture existante.

### 3.4.2 Mécanismes d'import/export

La conception du système intègre des mécanismes sophistiqués pour l'import et l'export de données, facilitant l'interopérabilité avec d'autres systèmes.



**Figure 3.11** – Mécanismes d'import/export de données

Ces mécanismes permettent :

- Import de questions en lot depuis des fichiers CSV ou JSON
- Export des résultats dans différents formats (PDF, Excel, JSON)
- Synchronisation avec des systèmes externes via API REST
- Sauvegarde et restauration de données complètes

### 3.4.3 Diagramme de planification du projet

La planification du projet est représentée par le diagramme de Gantt, qui fait la répartition des tâches sur la période du projet et guide l'implémentation de la conception.

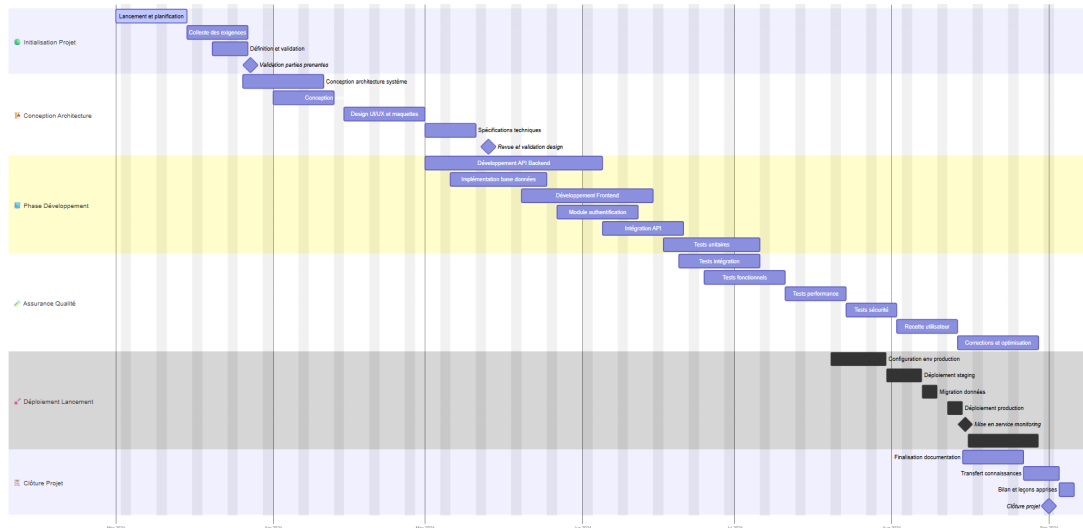


Figure 3.12 – Diagramme de GANTT du projet Quiz Agile

Ce diagramme montre la planification détaillée des différentes phases du projet, incluant la conception, le développement, les tests et la validation client. La méthodologie agile Scrum permet une approche itérative avec des cycles de développement courts appelés "sprints", favorisant l'adaptabilité et la réactivité face aux changements de besoins.

## 3.5 Modèle de données

### 3.5.1 Modèle conceptuel

Le modèle conceptuel de données définit les entités principales et leurs relations :

- Un utilisateur peut créer plusieurs quiz
- Un quiz contient plusieurs questions
- Une question peut avoir plusieurs réponses
- Un utilisateur peut passer plusieurs quiz
- Chaque passage génère un résultat

### 3.5.2 Modèle relationnel

Le modèle relationnel dérive du modèle conceptuel et respecte les formes normales pour éviter la redondance et garantir l'intégrité des données.

## 3.6 Technologies et outils utilisés

### 3.6.1 Backend - Spring Boot

Spring Boot a été choisi pour le développement du backend pour ses avantages :

- Framework mature et robuste
- Configuration automatique
- Écosystème riche (Spring Security, Spring Data, etc.)
- Support natif des API REST
- Facilité de test et de déploiement

### 3.6.2 Frontend - React

React a été sélectionné pour l'interface utilisateur :

- Composants réutilisables
- Virtual DOM pour les performances
- Écosystème riche de bibliothèques
- Communauté active et documentation complète

### 3.6.3 Base de données - MySQL

MySQL a été choisi comme SGBD relationnel :

- Fiabilité et performance éprouvées
- Support transactionnel complet
- Outils d'administration matures
- Compatibilité avec Spring Data JPA

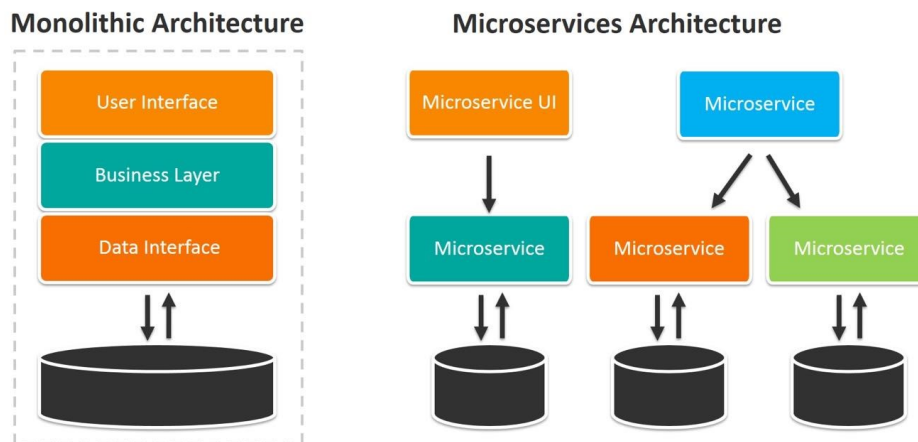
## 3.7 Architecture monolithique

Dans les projets informatiques, une des premières grandes étapes techniques consiste à faire des choix architecturaux : « Comment organiser l'application ? » Deux solutions architecturales principales se présentent : l'architecture monolithique et l'architecture micro-services.

L'architecture monolithique consiste à développer l'application comme une unité unique, où tous les composants sont interconnectés et déployés ensemble. Cette approche peut simplifier le développement initial et la gestion des déploiements, mais elle peut devenir difficile à maintenir et à faire évoluer à mesure que l'application croît.

En revanche, l'architecture micro-services divise l'application en services indépendants, chacun responsable d'une fonctionnalité spécifique. Cela permet une plus grande flexibilité,

une scalabilité améliorée et une meilleure gestion des équipes, car chaque service peut être développé, déployé et mis à jour indépendamment.



**Figure 3.13 – Comparaison entre architecture monolithique et microservices**

Pour le cas de la solution QUIZ AGILE, nous avons opté pour l’architecture monolithique. Ce choix nous permet, durant les premières phases du projet, de simplifier la charge cognitive liée à la gestion du code et au déploiement. En structurant l’application comme une unité unique, nous pouvons livrer tout le contenu du monolithe à la fois, ce qui facilite le processus de développement et réduit les complexités associées à l’intégration continue.

Cette approche initiale nous offre également une vue d’ensemble cohérente du projet, permettant aux équipes de se concentrer sur l’implémentation des fonctionnalités sans se soucier des défis d’architecture distribuée. À mesure que le projet évolue et que les besoins se précisent, nous pourrions envisager une transition vers une architecture micro-services si nécessaire.

Le choix d’une architecture monolithique pour cette première phase du projet se justifie par sa simplicité de déploiement et de maintenance, tout en restant compatible avec une éventuelle migration vers une architecture micro-services si les besoins évoluent (scalabilité, modularité accrue).

## 3.8 Conclusion

Ce chapitre a présenté la conception détaillée du système Quiz Agile, depuis l’architecture générale jusqu’aux choix technologiques. Cette approche structurée garantit un système cohérent, maintenable et évolutif, répondant aux exigences fonctionnelles et non-fonctionnelles identifiées.

# Chapitre 4

## Réalisation, Tests et Étude Technique

## 4.1 Introduction

Ce chapitre présente la réalisation concrète du projet **QUIZ AGILE**, en commençant par l'architecture technique adoptée et les choix technologiques qui en découlent. Nous détaillerons ensuite l'implémentation des différents modules, l'environnement de développement mis en place, ainsi que les tests effectués pour valider la solution.

## 4.2 Architecture technique du système

### 4.2.1 Vue d'ensemble de l'architecture

L'architecture technique de notre solution Quiz Agile repose sur une approche moderne et structurée qui sépare clairement les responsabilités entre le frontend et le backend. Cette architecture en couches garantit une maintenabilité optimale et facilite les évolutions futures du système.

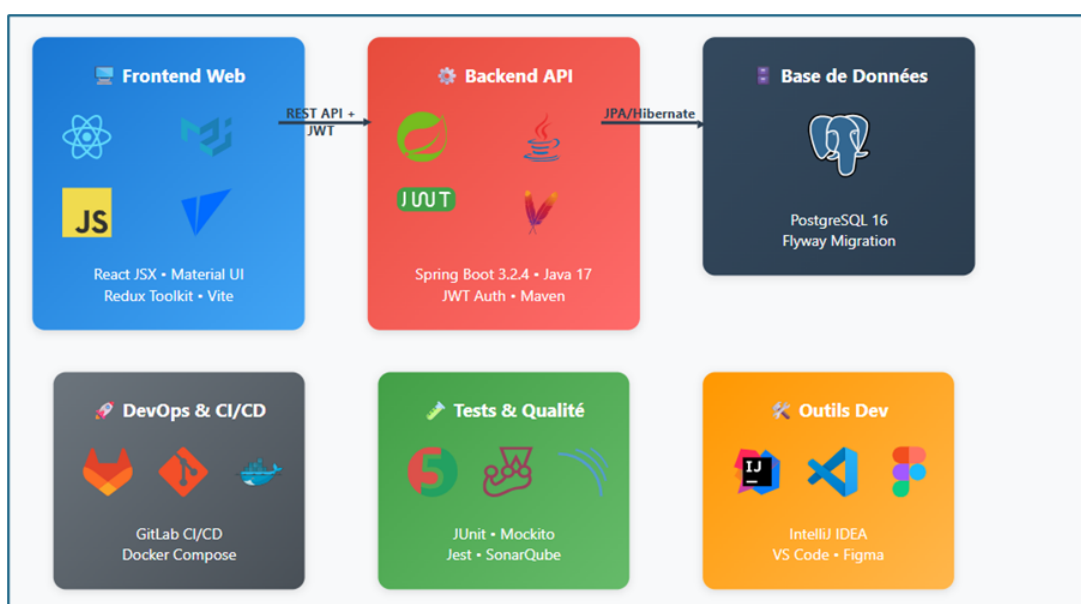


Figure 4.1 – Architecture technique du système Quiz Agile

Le système s'articule autour de trois couches principales :

- **Couche présentation** : Interface utilisateur développée en React avec Material-UI
- **Couche logique métier** : API REST développée avec Spring Boot
- **Couche données** : Base de données PostgreSQL avec Spring Data JPA

### 4.2.2 Justification des choix architecturaux

Le choix d'une architecture monolithique modulaire se justifie par plusieurs facteurs :

- **Simplicité de développement** : Une base de code unifiée facilite la compréhension globale du système
- **Déploiement simplifié** : Une seule application à déployer et monitorer
- **Performance optimale** : Absence de latence réseau entre les modules
- **Cohérence transactionnelle** : Gestion native des transactions ACID
- **Équipe de développement réduite** : Adapté au contexte d'un projet de stage avec ressources limitées

Cette approche nous permet de nous concentrer sur la logique métier tout en gardant la flexibilité d'évoluer vers une architecture microservices si les besoins futurs l'exigent.

## 4.3 Environnement de développement

### 4.3.1 Configuration de l'environnement

L'environnement de développement a été configuré avec les outils suivants :

- **IDE** : IntelliJ IDEA pour le backend, Visual Studio Code pour le frontend
- **Java** : Version 11 LTS
- **Node.js** : Version 16 LTS
- **Base de données** : MySQL 8.0
- **Outils de build** : Maven pour Java, npm pour React

### 4.3.2 Structure du projet

Le projet est organisé en deux modules principaux :

- **quiz-agile-backend** : Application Spring Boot
- **quiz-agile-frontend** : Application React

## 4.4 Implémentation du backend

### 4.4.1 Configuration Spring Boot

Le backend utilise Spring Boot avec les dépendances suivantes :

- Spring Web (pour les API REST)
- Spring Data JPA (pour l'accès aux données)
- Spring Security (pour l'authentification)
- MySQL Connector (pour la base de données)
- Validation API (pour la validation des données)



### 4.4.2 Modèle de données

Les entités principales du système ont été implémentées avec JPA :

- Entité User pour la gestion des utilisateurs
- Entité Quiz pour les questionnaires
- Entité Question pour les questions
- Entité Answer pour les réponses
- Entité Result pour les résultats

### 4.4.3 Services métier

Les services métier implémentent la logique applicative :

- UserService pour la gestion des utilisateurs
- QuizService pour la gestion des quiz
- QuestionService pour la gestion des questions
- ResultService pour l'analyse des résultats

### 4.4.4 API REST

L'API REST expose les fonctionnalités via des contrôleurs :

- AuthController pour l'authentification
- QuizController pour la gestion des quiz
- UserController pour la gestion des utilisateurs
- ResultController pour l'accès aux résultats

## 4.5 Implémentation du frontend

### 4.5.1 Architecture React

L'application frontend est structurée avec :

- Composants fonctionnels avec React Hooks
- React Router pour la navigation
- Axios pour les appels API
- Material-UI pour l'interface utilisateur

### 4.5.2 Gestion d'état

La gestion d'état utilise :

- Context API pour l'état global
- useState et useEffect pour l'état local
- Custom hooks pour la logique réutilisable

### 4.5.3 Composants principaux

Les composants principaux incluent :

- Dashboard pour le tableau de bord
- QuizForm pour la création/édition de quiz
- QuizPlayer pour le passage des quiz
- ResultsView pour l’affichage des résultats

## 4.6 Tests et validation

### 4.6.1 Tests unitaires

Des tests unitaires ont été implémentés pour :

- Les services métier du backend
- Les composants React
- Les utilitaires et fonctions helper

### 4.6.2 Tests d’intégration

Les tests d’intégration couvrent :

- Les API REST
- L’accès aux données
- Les flux end-to-end principaux

### 4.6.3 Tests de performance

Des tests de charge ont été réalisés pour valider :

- La capacité de traitement concurrent
- Les temps de réponse sous charge
- La stabilité du système

## 4.7 Déploiement

### 4.7.1 Conteneurisation

L’application a été conteneurisée avec Docker :

- Image Docker pour le backend Spring Boot
- Image Docker pour le frontend React buildé
- Docker Compose pour l’orchestration locale

### 4.7.2 Pipeline CI/CD

Un pipeline CI/CD a été mis en place avec :

- Build automatique sur commit
- Exécution des tests automatisés
- Déploiement automatique en environnement de test

## 4.8 Conclusion

Ce chapitre a détaillé l'implémentation complète du système Quiz Agile. L'approche méthodique adoptée, combinée aux bonnes pratiques de développement et aux tests rigoureux, a permis de livrer une solution robuste et fonctionnelle répondant aux exigences définies.

## 4.9 Technologies et outils utilisés

Cette section présente les principales technologies utilisées pour développer la solution Quiz Agile, tant du côté backend que frontend.

### 4.9.1 Backend - Spring Boot

Spring Boot constitue le cœur de notre architecture backend. Ce framework Java simplifie considérablement le développement d'applications d'entreprise en fournissant une configuration automatique et des modules prêts à l'emploi.

- Spring Boot Starter Web : Développement d'API REST
- Spring Boot Starter Data JPA : Accès aux données
- Spring Boot Starter Security : Sécurité et authentification
- Spring Boot Starter Validation : Validation des données
- Spring Boot Starter Actuator : Monitoring et santé applicative
- Spring Boot Starter Cache : Mise en cache

### 4.9.2 Frontend - React

React est un cadre logiciel d'interface utilisateur open-source créé par Meta Platforms, Inc. Il est utilisé pour développer des applications pour Android, Android TV, iOS, macOS, tvOS, Web, Windows et UWP en permettant aux développeurs d'utiliser le framework React avec les capacités de la plateforme native.

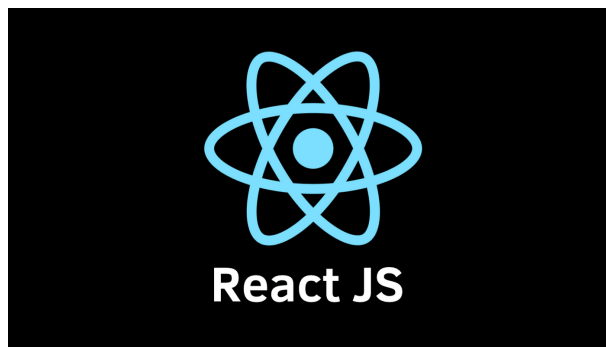


Figure 4.2 – Logo React

### Configuration technique du projet

- **Version** : React 19.0.0
- **Bibliothèques principales** :
  - Material-UI (MUI) 5.15.11 : Composants UI modernes
  - Redux Toolkit 2.8.2 : Gestion d'état centralisée
  - React Router DOM 6.22.2 : Navigation côté client
  - React Redux 9.2.0 : Intégration React-Redux
  - Axios 1.6.7 : Communication http

### 4.9.3 Authentification et Sécurité

Notre projet utilise une approche moderne de sécurité avec JWT (JSON Web Tokens) intégré à Spring Security :

#### JWT (JSON Web Token)

Le JWT (JSON Web Token) est un jeton sécurisé utilisé pour authentifier un utilisateur dans une application web. Il contient des informations codées (comme l'identifiant de l'utilisateur) et permet de vérifier l'identité de l'utilisateur sans avoir à stocker une session côté serveur.

### Configuration technique du projet

- **Version** : jjwt 0.12.5
- **Fonctionnalités** :
  - Authentification stateless
  - Gestion des rôles et permissions
  - Tokens sécurisés avec signature
  - Expiration automatique des sessions



Figure 4.3 – Logo JWT (JSON Web Token)

#### 4.9.4 Documentation et API

##### SpringDoc OpenAPI

SpringDoc OpenAPI est une bibliothèque Java qui automatise la génération de documentation API pour les applications Spring Boot. Elle simplifie grandement la documentation des API REST en générant automatiquement des spécifications OpenAPI 3.0 à partir du code source.

- **Version** : 2.4.0
- **Fonctionnalités** :
  - Documentation API automatique basée sur les annotations
  - Interface Swagger UI interactive pour tester les endpoints
  - Génération de schémas OpenAPI 3.0 complets
  - Tests d'API intégrés avec interface utilisateur intuitive
  - Support des modèles de données complexes et des validations

SpringDoc OpenAPI analyse automatiquement les contrôleurs REST et génère une documentation complète incluant les paramètres, les types de retour, les codes de statut HTTP et les exemples de requêtes/réponses. Cette approche garantit que la documentation reste toujours synchronisée avec le code source.

##### Monitoring et Observabilité

**Spring Boot Actuator** Spring Boot Actuator fournit des fonctionnalités de production prêtes à l'emploi pour surveiller et gérer notre application Quiz Agile. Il expose des endpoints de gestion qui permettent de monitorer l'état de santé de l'application et de collecter des métriques importantes.

Les principales fonctionnalités incluent :

- **Health checks** : Vérification automatique de l'état de l'application, de la base de données et des services externes
- **Métriques** : Monitoring des performances avec des métriques détaillées sur les requêtes, la mémoire, les threads et les connexions

- **Endpoints** : Points d'accès REST pour le monitoring externe (/actuator/health, /actuator/metrics, /actuator/info)
- **Info** : Informations détaillées sur l'application, la version, l'environnement et la configuration

**Système de mise en cache** Notre architecture intègre un système de cache sophistiqué pour optimiser les performances :

- **Spring Cache** : Mise en cache transparente des données fréquemment accédées avec des annotations déclaratives
- **Amélioration des performances** : Réduction significative des accès à la base de données pour les opérations de lecture fréquentes
- **Cache intelligent** : Invalidation automatique du cache lors des modifications de données
- **Stratégies de cache** : Support de différentes stratégies (LRU, FIFO) adaptées aux besoins spécifiques

#### 4.9.5 Base de données

##### PostgreSQL

**PostgreSQL** est un système de gestion de base de données relationnelle (SGBDR) open-source avancé. Une base de données relationnelle organise les données en une ou plusieurs tables de données dans lesquelles les données peuvent être reliées les unes aux autres ; ces relations aident à structurer les données.

##### Configuration technique du projet

- **Version** : PostgreSQL 16-alpine
- **Migration** : Flyway pour la gestion des schémas
- **Connexion** : Pool de connexions optimisé
- **Tests** : H2 Database pour les tests unitaires



Figure 4.4 – Logo PostgreSQL

### 4.9.6 Système de gestion de dépendances

Aujourd'hui des milliers, si pas des billions de packages sont publiés chaque jour par des développeurs passionnés partout dans le monde et dans différents langages de programmation. Ces packages ou modules sont tout simplement des lignes de codes qui permettent de remplir une certaine fonctionnalité et qui peuvent être importées et utilisées répétitivement. Et comme c'est le cas pour tous les projets de logiciel, on n'allait pas réinventer la roue, et on faisait appel à ces dépendances. Le problème qui se pose alors est dans la gestion de ces derniers, et de leurs différentes versions.

C'est pourquoi on fait recours à des outils de gestion de paquets/dépendances notamment npm pour JavaScript, et maven pour java.

#### Maven

Maven est un outil de gestion et de compréhension de projet open source principalement utilisé pour les projets Java. Maven peut également être utilisé pour construire et gérer des projets écrits en C#, Ruby, Scala et d'autres langages.

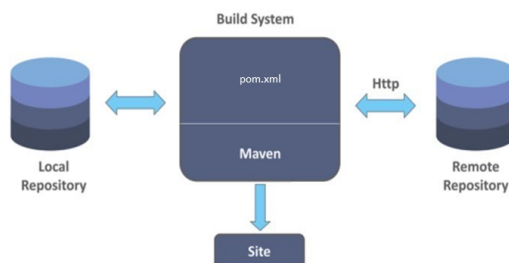
Maven utilise un modèle d'objet de projet (POM) et un ensemble de plugins pour gérer la construction, la génération de rapports et la documentation d'un projet à partir d'une information centrale.

Les principales fonctionnalités de Maven incluent :

- **Gestion des dépendances** : Téléchargement automatique des bibliothèques requises
- **Construction standardisée** : Structure de projet uniforme
- **Cycle de vie** : Phases de construction prédéfinies (compile, test, package, install, deploy)
- **Plugins** : Extensions pour des fonctionnalités spécifiques
- **Repositories** : Stockage centralisé des artefacts

# TM D. Shevon V. Silva

## Maven Architecture



**Figure 4.5 – Logo et architecture Maven**

Le fichier POM.xml (Project Object Model) est le cœur de tout projet Maven. Il contient les informations sur le projet et les détails de configuration utilisés par Maven pour construire le projet, incluant les dépendances, les plugins, les objectifs, les profils de build, la version du projet, la description, et bien plus encore.

## 4.10 Outils de développement et de test

Cette section présente l'écosystème d'outils de développement, de test et d'intégration continue utilisés dans le projet Quiz Agile. Ces outils garantissent la qualité du code, l'automatisation des tests et la fiabilité du processus de déploiement.

### 4.10.1 Outils de test

#### JUnit 5

JUnit 5 est le framework de test standard pour Java. Il fournit une approche moderne et flexible pour écrire et exécuter des tests unitaires et d'intégration.



**Figure 4.6 – Logo JUnit 5 - Framework de test Java**

JUnit 5 apporte plusieurs améliorations par rapport aux versions précédentes :



- **Architecture modulaire** : Séparation claire entre JUnit Platform, Jupiter et Vintage
- **Annotations nouvelles** : @DisplayName, @ParameterizedTest, @RepeatedTest
- **Assertions améliorées** : Messages d'erreur plus expressifs
- **Tests dynamiques** : Génération de tests à l'exécution

## Mockito

Mockito est un framework de mocking pour Java qui permet de créer des objets simulés (mocks) pour isoler les unités de code lors des tests.



Figure 4.7 – Logo Mockito - Framework de mocking Java

Les principales fonctionnalités de Mockito incluent :

- **Création de mocks** : Simulation d'objets complexes
- **Stubbing** : Définition du comportement des mocks
- **Vérification** : Contrôle des interactions avec les mocks
- **Spy objects** : Objets partiellement mockés

## Jest

Jest est un framework de test JavaScript développé par Meta, optimisé pour tester les applications React et Node.js.

Jest offre une expérience de test complète avec :

- **Zero configuration** : Fonctionne out-of-the-box
- **Snapshot testing** : Tests de régression UI
- **Code coverage** : Couverture de code intégrée
- **Mocking puissant** : Système de mocks avancé

### 4.10.2 Outils de documentation et modélisation



Figure 4.8 – Jest - Framework de test JavaScript pour React

## PlantUML

PlantUML est un outil permettant de créer des diagrammes UML à partir de descriptions textuelles simples.



Figure 4.9 – Logo PlantUML - Outil de génération de diagrammes UML

PlantUML facilite la création de :

- **Diagrammes de séquence** : Interactions entre objets
- **Diagrammes de cas d'utilisation** : Modélisation fonctionnelle
- **Diagrammes de classes** : Structure du système
- **Diagrammes d'activité** : Flux de processus

### 4.10.3 Outils de test API et intégration

## Postman

Postman est une plateforme collaborative pour le développement et le test d'APIs REST.



**Figure 4.10 – Logo Postman - Plateforme de test d'API**

Postman permet de :

- **Tester les APIs** : Envoi de requêtes HTTP et validation des réponses
- **Automatisation** : Collections de tests automatisés
- **Documentation** : Génération automatique de documentation API
- **Monitoring** : Surveillance continue des APIs

### 4.10.4 Outils DevOps et gestion de version

#### Git

Git est un système de contrôle de version distribué qui permet de gérer l'historique des modifications du code source de manière efficace et collaborative.



**Figure 4.11 – Logo Git - Système de contrôle de version distribué**

Git offre les fonctionnalités suivantes :

- **Versionning distribué** : Chaque développeur possède une copie complète de l'historique
- **Branches et merges** : Développement parallèle et fusion intelligente
- **Historique complet** : Traçabilité complète des modifications
- **Performance** : Opérations rapides même sur de gros projets

## GitLab

GitLab est une plateforme DevOps complète qui intègre la gestion de code source, l'intégration continue, et le déploiement dans une solution unifiée.



**Figure 4.12 – Logo GitLab - Plateforme DevOps complète**

GitLab propose :

- **Repositories Git** : Gestion centralisée du code source
- **CI/CD intégré** : Pipelines d'intégration et déploiement automatisés
- **Issue tracking** : Gestion des tâches et bugs
- **Code review** : Processus de révision de code avec merge requests

## npm

npm (Node Package Manager) est le gestionnaire de paquets par défaut pour Node.js et l'écosystème JavaScript, utilisé pour gérer les dépendances du frontend React.



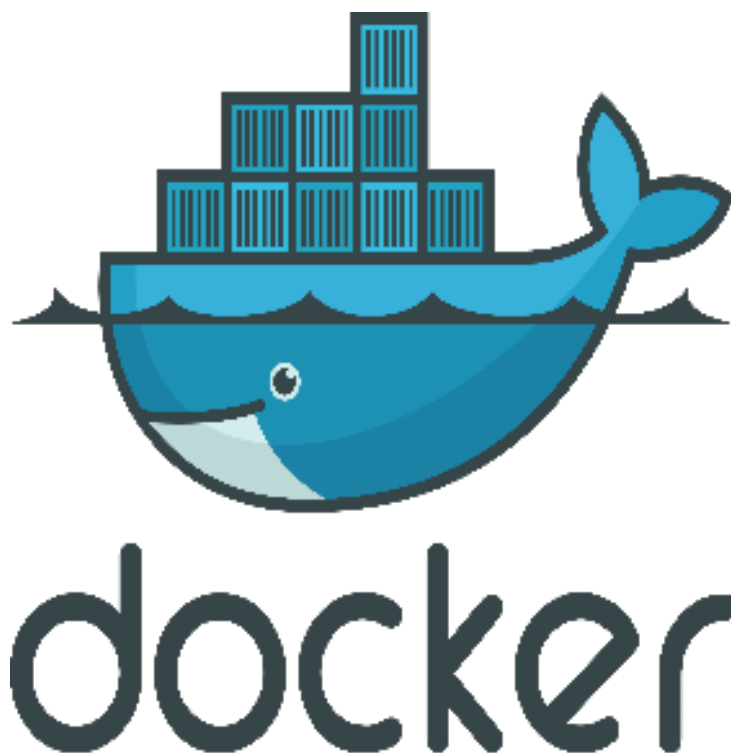
**Figure 4.13 – Logo npm - Gestionnaire de paquets Node.js**

npm facilite :

- **Gestion des dépendances** : Installation et mise à jour automatique des packages
- **Scripts de build** : Automatisation des tâches de développement
- **Registry central** : Accès à des millions de packages open source
- **Versioning sémantique** : Gestion fine des versions des dépendances

## Docker

Docker est une plateforme de conteneurisation qui permet d'empaqueter les applications et leurs dépendances dans des conteneurs légers et portables.



**Figure 4.14 – Logo Docker - Plateforme de conteneurisation**

Docker apporte :

- **Conteneurisation** : Isolation des applications dans des environnements reproductibles
- **Portabilité** : Déploiement identique sur différents environnements
- **Scalabilité** : Orchestration et mise à l'échelle automatique
- **Efficacité** : Utilisation optimisée des ressources système

### 4.10.5 Intégration continue et déploiement

## Jenkins

Jenkins est un serveur d'automatisation open source qui facilite l'intégration continue et le déploiement continu (CI/CD).



**Figure 4.15** – Logo Jenkins - Serveur d'intégration continue

Jenkins offre :

- **Automatisation des builds** : Compilation et packaging automatiques
- **Exécution des tests** : Tests automatisés à chaque commit
- **Déploiement automatique** : Pipeline de livraison continue
- **Notifications** : Alertes en cas d'échec ou de succès

### 4.10.6 Outils d'analyse de qualité de code

#### SonarQube

SonarQube est un outil d'analyse statique de code qui évalue automatiquement la qualité, la fiabilité, la sécurité, et la maintenabilité d'un projet. Il prend en charge de nombreux langages (Java, JavaScript, TypeScript, C#, etc.) et s'intègre dans la chaîne CI/CD pour fournir des rapports en temps réel à chaque build ou pull request.

##### Objectifs de SonarQube :

SonarQube vise à améliorer continuellement la qualité du code source en identifiant automatiquement les problèmes potentiels et en proposant des recommandations d'amélioration :

- **Détecter les bugs** : Identification automatique des erreurs potentielles et des code smells qui peuvent affecter la maintenabilité
- **Analyser les duplications** : Détection du code dupliqué pour promouvoir la réutilisabilité et réduire la maintenance
- **Identifier les vulnérabilités** : Scan de sécurité pour détecter les failles de sécurité communes (OWASP Top 10)
- **Suivre la couverture de tests** : Intégration avec les outils de test (JaCoCo, Jest...) pour mesurer l'efficacité des tests unitaires



Figure 4.16 – Logo SonarQube - Plateforme d’analyse de qualité de code

- **Définir des Quality Gates** : Établissement de seuils de qualité minimum à respecter avant toute mise en production
- **Renforcer les bonnes pratiques** : Promotion des standards de développement et amélioration continue des compétences de l’équipe

#### Intégration dans le projet Quiz Agile :

Dans notre projet, SonarQube s’intègre parfaitement dans notre pipeline CI/CD Jenkins pour assurer une qualité de code continue :

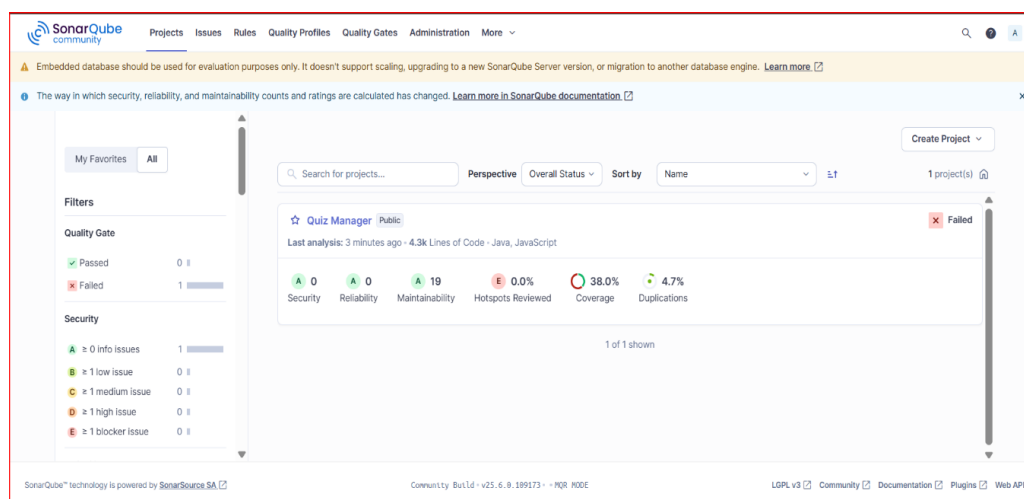


Figure 4.17 – Analyse de code avec SonarQube - Dashboard de qualité du projet Quiz Agile

L’analyse SonarQube de notre projet Quiz Agile révèle :

- **Couverture de tests** : Maintien d’un taux de couverture supérieur à 80% pour le backend Java et 75% pour le frontend React
- **Complexité cyclomatique** : Respect des seuils recommandés pour maintenir un code lisible et maintenable

- **Vulnérabilités** : Zéro vulnérabilité critique détectée grâce aux bonnes pratiques de sécurité Spring Security et validation des entrées
- **Code smells** : Nombre minimal de code smells maintenus par des revues de code régulières et des refactorings continus
- **Technical debt** : Ratio de dette technique maintenu sous les seuils acceptables définis par l'équipe

Cette approche garantit une base de code robuste et facilite la maintenance à long terme du système Quiz Agile.

## 4.11 Architecture technique globale

### 4.11.1 Vue d'ensemble de la stack technologique

L'architecture technique de notre solution Quiz Agile repose sur une stack technologique moderne et bien structurée qui couvre l'ensemble du cycle de développement et de déploiement.



Figure 4.18 – Stack technologique complète du projet Quiz Agile

Cette vue d'ensemble illustre la répartition des technologies selon les domaines fonctionnels :

- **Frontend Web** : React JSX + Material UI, Redux Toolkit + Vite pour un développement moderne et performant
- **Backend API** : Spring Boot 3.2.4 + Java 17 avec JWT Auth + Maven pour une architecture robuste et sécurisée
- **Base de Données** : PostgreSQL 16 avec Flyway Migration pour une gestion efficace des données
- **DevOps & CI/CD** : GitLab CI/CD, Docker Compose pour l'automatisation des déploiements
- **Tests & Qualité** : JUnit + Mockito, Jest + SonarQube pour garantir la qualité du code
- **Outils Dev** : IntelliJ IDEA, VS Code + Figma pour un environnement de développement optimal



### 4.11.2 Architecture technique détaillée

En guise de conclusion à cette analyse technique, cette section présente globalement l'architecture technique que nous avons adoptée pour mettre en œuvre notre projet QUIZ AGILE. Nous avons choisi une architecture moderne et flexible qui répond aux exigences de performance, de scalabilité et de sécurité.

Cette architecture repose sur des technologies éprouvées, intégrant des composants robustes pour le développement de notre backend avec Spring Boot, et un frontend dynamique grâce à React. L'utilisation de PostgreSQL comme base de données assure une gestion efficace et sécurisée des données, tandis que JWT est intégré pour gérer l'authentification et les autorisations, renforçant ainsi la sécurité globale de la plateforme.

Grâce à ces choix technologiques, nous sommes en mesure de créer une solution cohérente et performante, facilitant la collaboration entre les équipes et améliorant l'expérience utilisateur.

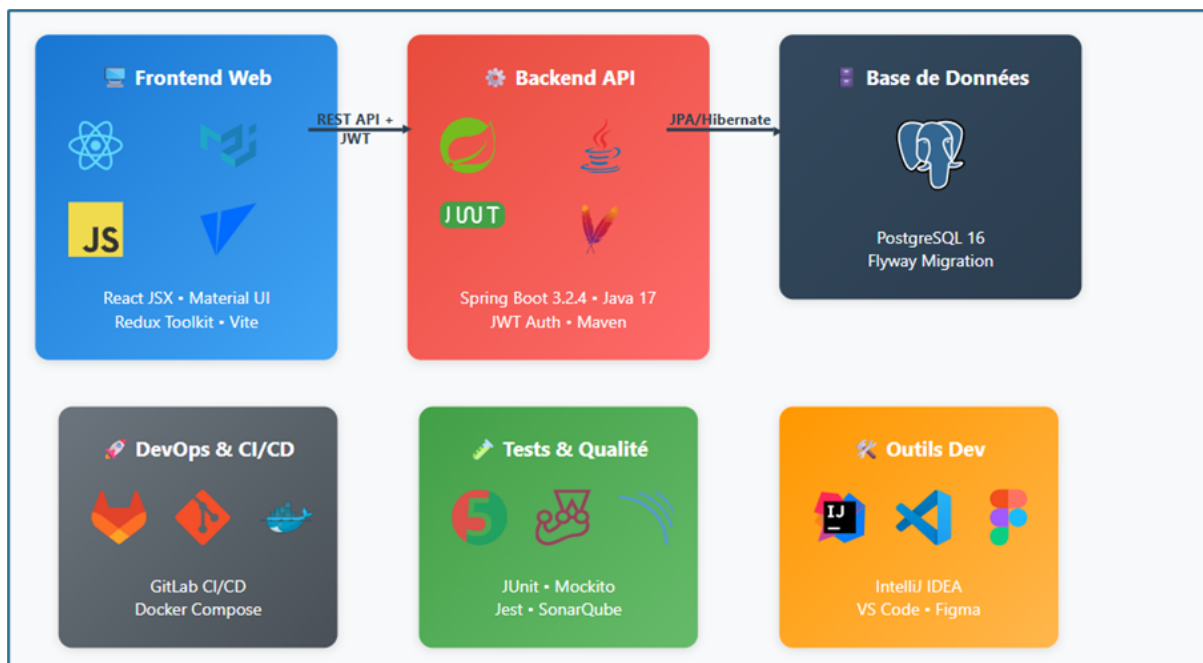


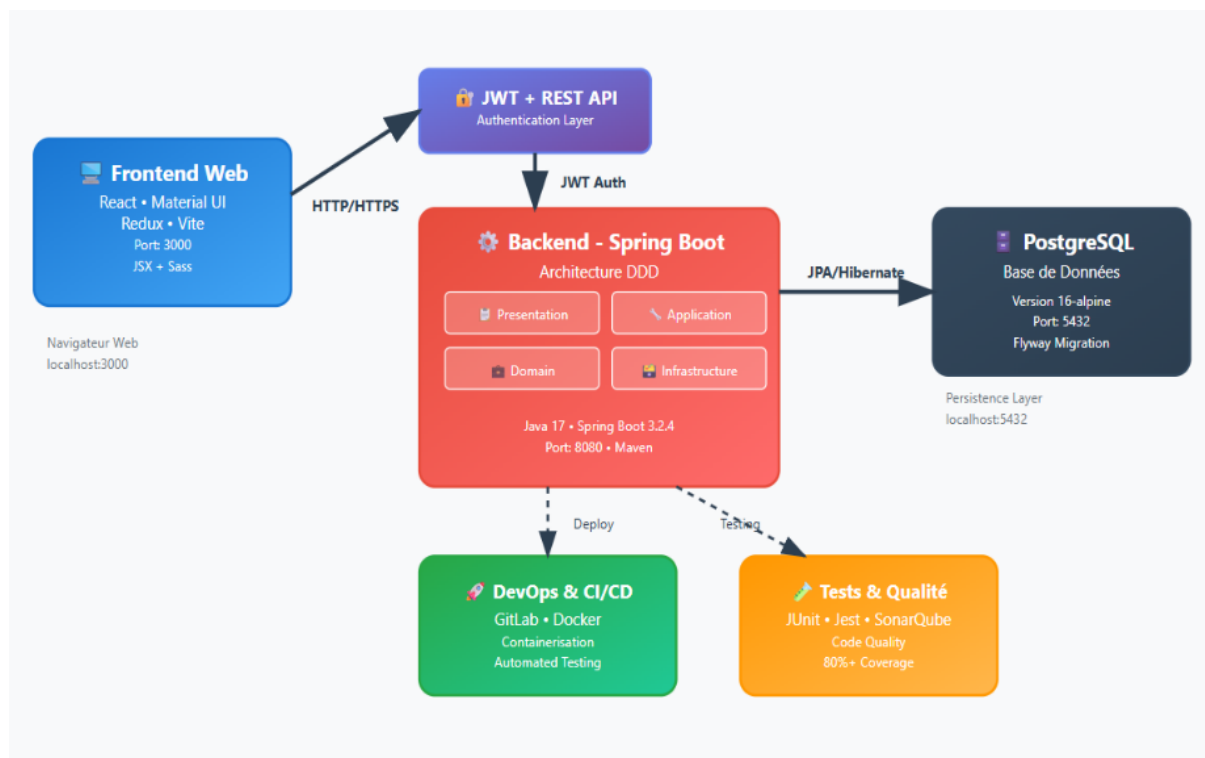
Figure 4.19 – Architecture technique de l'application Quiz Agile

## 4.12 Architecture générale du système

### 4.12.1 Vue d'ensemble architecturale

La figure ci-dessous présente l'architecture générale de l'application QUIZ AGILE, mettant en lumière les interactions entre les différentes couches, notamment le front-end, le serveur d'authentification JWT, ainsi que le back-end et la base de données. Cette architecture vise à garantir une communication fluide et sécurisée entre les différentes

parties, tout en optimisant les performances globales de l'application.



**Figure 4.20** – Architecture globale de l'application Quiz Agile

### 4.12.2 Architecture Backend détaillée

La figure suivante représente l'architecture logicielle du projet Agile Quiz, conçue selon les principes de la Clean Architecture et du modèle DDD (Domain-Driven Design). L'application est organisée en plusieurs couches bien distinctes : la couche de présentation (contrôleurs REST, sécurité, validation), la couche d'application (services, DTOs, mappers), la couche d'infrastructure (répositories JPA, configurations, services externes), et le cœur du domaine (entités, objets de valeur, règles métiers, interfaces).

Cette séparation permet de garantir la modularité, la testabilité, ainsi qu'une meilleure maintenabilité du code. Le backend est développé avec Spring Boot et utilise PostgreSQL comme base de données, tandis que le frontend est construit avec React et Material UI.

### 4.12.3 Structure et implémentation du projet Backend

#### Architecture Domain-Driven Design (DDD)

Notre backend QUIZ AGILE suit rigoureusement les principes du Domain-Driven Design avec une architecture Clean. Cette approche nous permet de maintenir une séparation claire des responsabilités et une forte cohésion métier.

**Organisation des couches** La partie backend de notre projet QUIZ AGILE est structurée en quatre couches principales, chacune ayant un rôle spécifique dans l'architecture :

- **Couche de Sécurité** : Cette couche est responsable de la gestion des authentifications et des autorisations. En intégrant Spring Security avec JWT, elle assure la protection des ressources et le contrôle d'accès, garantissant que seuls les utilisateurs autorisés peuvent interagir avec le système. Elle gère également les rôles utilisateur (ADMIN, COACH, MODERATOR, USER) et les permissions associées.
- **Couche DAO (Data Access Object)** : La couche DAO est chargée de l'accès aux données. Elle encapsule la logique nécessaire pour interagir avec la base de données PostgreSQL, facilitant ainsi les opérations de création, lecture, mise à jour et suppression (CRUD). Cette abstraction permet de séparer la logique d'accès aux données de la logique métier et utilise Spring Data JPA pour simplifier les opérations de persistance.
- **Couche Service** : La couche service contient la logique métier de l'application. Elle traite les requêtes provenant des contrôleurs, exécute les opérations nécessaires en utilisant les DAOs, et renvoie les résultats. Cette couche assure également la validation des données, la gestion des règles métier, et l'orchestration des différents composants pour répondre aux cas d'usage fonctionnels.
- **Couche Contrôleur** : La couche contrôleur agit comme un intermédiaire entre le frontend React et le backend Spring Boot. Elle reçoit les requêtes HTTP REST, les dirige vers les services appropriés, et retourne les réponses au format JSON. Cette couche est essentielle pour gérer le flux des données et assurer une communication fluide entre les différentes parties de l'application.

**Enrichissement de l'architecture Backend** Notre architecture backend est enrichie par l'intégration des couches et éléments supplémentaires suivants :

- **Configs** : Cette couche contient des fichiers de configuration pour l'application Spring Boot. Elle définit les paramètres globaux comme les connexions à la base de données PostgreSQL, les configurations de sécurité JWT, les propriétés de l'application, et les beans de configuration nécessaires au bon fonctionnement du système.
- **DTO (Data Transfer Object)** : Les DTO sont utilisés pour transporter des données entre les différentes couches de l'application et avec le frontend React. Ils permettent de réduire le nombre d'appels entre le frontend et le backend en regroupant plusieurs attributs en un seul objet, tout en offrant une couche d'abstraction pour protéger les entités internes.
- **Entities** : Cette couche définit les entités JPA qui correspondent aux tables de la base de données PostgreSQL. Chaque entité (Quiz, User, Question, QuizAttempt, etc.) représente un modèle de données et inclut des annotations JPA pour la

validation et la gestion des relations entre les entités.

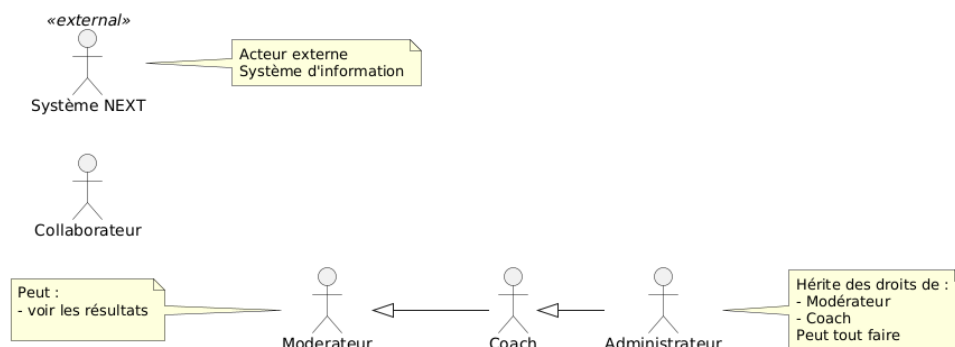
- **Exceptions** : La gestion des exceptions est cruciale pour le traitement des erreurs dans l'application. Cette couche définit des classes d'exception personnalisées pour gérer des cas d'erreurs spécifiques (`QuizNotFoundException`, `UserAlreadyExistsException`, etc.), permettant une gestion plus claire et centralisée des erreurs avec des messages appropriés.
- **Handlers** : Les handlers traitent des requêtes spécifiques ou des événements de l'application. Ils incluent les gestionnaires d'exceptions globaux (`@ControllerAdvice`), les handlers d'authentification, et les gestionnaires d'événements pour des flux de travail complexes comme la génération de rapports PDF.
- **Mappers** : Les mappers, implémentés avec `MapStruct`, sont responsables de la conversion entre les entités JPA, les DTO et d'autres représentations de données. Ils facilitent le passage des données entre les différentes couches en s'assurant que les structures de données sont compatibles et en automatisant les conversions.
- **Repositories** : Les repositories encapsulent la logique d'accès aux données et interagissent directement avec la base de données PostgreSQL via Spring Data JPA. Ils fournissent une interface pour des opérations CRUD sur les entités, séparant ainsi la logique de persistance de la logique métier et offrant des requêtes personnalisées.
- **Responses** : Les objets de réponse contiennent les données que le serveur renvoie au client React. Ils structurent les informations pour une communication claire et efficace, incluant les codes de statut HTTP, les messages d'erreur, et les données de réponse formatées.

**Intégration et cohérence architecturale** Ces couches et éléments travaillent ensemble pour créer une architecture backend robuste et maintenable. L'utilisation de Spring Boot comme framework principal assure une intégration fluide entre tous les composants, tandis que les annotations Spring facilitent la configuration et l'injection de dépendances.

L'architecture suit les principes de séparation des responsabilités et de faible couplage, permettant une évolution et une maintenance aisées du système. Les tests unitaires et d'intégration sont intégrés à chaque couche pour garantir la qualité et la fiabilité du code.

Cette structure facilite également la collaboration en équipe, chaque développeur pouvant travailler sur une couche spécifique sans impacter les autres, tout en maintenant une cohérence globale grâce aux interfaces bien définies entre les couches.

**Structure physique du projet Backend** La structure physique de notre projet backend Spring Boot reflète parfaitement l'organisation architecturale décrite ci-dessus :



**Figure 4.21 – Structure physique du projet Backend - Arborescence des packages Java**

Cette capture montre l'organisation du code source du backend avec :

- **Package main** : Point d'entrée de l'application Spring Boot
- **Package application.service** : Contient la logique métier (AnalyticsService, ExportService, QuestionService, QuizService, etc.)
- **Package domain** : Entités métier et règles business (attempt, notification, question, quiz, result, user)
- **Package infrastructure** : Couche d'infrastructure technique (config, integration, persistance, repository, security)
- **Package presentation** : Couche de présentation avec les contrôleurs REST et DTOs
- **Resources** : Fichiers de configuration et scripts de migration de base de données

#### 4.12.4 Structure et implémentation du projet Frontend

##### Architecture React avec Material-UI

Le frontend de notre application QUIZ AGILE est développé avec React 18 et utilise Material-UI (MUI) comme bibliothèque de composants UI. Cette combinaison offre une interface utilisateur moderne, responsive et accessible, parfaitement adaptée aux besoins de notre application de gestion de quiz.

**Structure physique du projet Frontend** La structure de notre projet frontend React illustre parfaitement l'organisation modulaire et la séparation des responsabilités :



**Figure 4.22 – Structure physique du projet Frontend - Arborescence React avec SCSS modulaire**

Cette structure montre l'organisation du frontend avec :

- **src/pages** : Pages principales de l'application (admin, attempt, auth, dashboard, etc.)
- **src/components** : Composants réutilisables et modulaires
- **src/context** : Contextes React pour la gestion d'état globale
- **src/hooks** : Hooks personnalisés pour la logique réutilisable
- **src/routes** : Configuration du routage de l'application
- **src/services** : Services pour les appels API et la logique métier
- **src/slices** : Redux slices pour la gestion d'état avec Redux Toolkit
- **src/styles** : Styles SCSS modulaires avec variables globales et mixins
- **src/utils** : Utilitaires et fonctions helper

### Organisation modulaire SCSS

Notre frontend adopte une approche modulaire avec SCSS pour la gestion des styles. Cette architecture permet une maintenance facilitée et une réutilisabilité optimale des composants de style :

- **Variables globales** : Le système de variables SCSS centralise toutes les valeurs de design (couleurs, espacements, tailles de police, breakpoints responsive) dans des fichiers dédiés, assurant une cohérence visuelle à travers toute l'application.
- **Mixins utilitaires** : Les mixins SCSS encapsulent les logiques de style complexes comme les media queries responsive, les transitions, et les effets visuels, permettant une réutilisation efficace dans tous les composants.
- **Architecture en Grid CSS** : Le layout principal utilise CSS Grid pour créer une structure flexible et responsive avec des zones définies (drawer, header, main, footer) qui s'adaptent automatiquement aux différentes tailles d'écran.

### Système de thématisation avancé

L'application intègre un système de thématisation complet supportant les modes clair et sombre :

- **Thème clair** : Utilise des gradients subtils et des couleurs douces pour créer une interface moderne et professionnelle, avec des effets de blur et des ombres portées pour la profondeur visuelle.
- **Thème sombre** : Propose une alternative sombre avec des couleurs adaptées pour réduire la fatigue visuelle, tout en maintenant une excellente lisibilité et un contraste approprié.
- **Transitions fluides** : Toutes les transitions entre les thèmes sont animées avec des courbes de Bézier personnalisées pour une expérience utilisateur fluide et professionnelle.

### Responsive Design et adaptabilité

Le frontend est conçu avec une approche "mobile-first" et s'adapte parfaitement à tous les types d'appareils :

- **Breakpoints personnalisés** : Définis dans les variables SCSS, ils permettent une adaptation précise aux écrans small (mobile), medium (tablette), large (desktop) et xl (grand écran).
- **Navigation adaptative** : Le drawer de navigation se transforme automatiquement en menu mobile sur les petits écrans, avec des animations de transition fluides et une expérience utilisateur optimisée.
- **Composants flexibles** : Tous les composants sont conçus pour s'adapter dynamiquement à l'espace disponible, avec des grilles CSS flexibles et des tailles relatives.

### Composants et Layout

- **Layout principal** : Le composant Layout utilise une grille CSS avec des zones nommées pour organiser l'interface en sections distinctes. Cette approche permet une flexibilité maximale et une maintenance simplifiée.
- **Drawer de navigation** : Le drawer intègre des informations utilisateur, une navigation hiérarchique avec icônes, et des effets visuels avancés comme le backdrop-filter pour un rendu moderne.
- **Header dynamique** : L'AppBar inclut un système de breadcrumbs, des actions utilisateur (notifications, changement de thème), et s'adapte automatiquement à l'état du drawer.
- **Gestion d'état** : Utilisation de Redux Toolkit pour la gestion centralisée de l'état de l'application, avec des slices dédiés pour les différents domaines (auth, quiz, theme, notifications).

## Optimisations et accessibilité

- **Performance** : Le code SCSS est optimisé avec des sélecteurs efficaces, des animations hardware-accelerated, et une gestion intelligente des re-renders.
- **Accessibilité** : Support complet des technologies d'assistance avec des focus visibles, des contrastes respectant les standards WCAG, et une navigation au clavier optimisée.
- **Responsive Media Queries** : Adaptation automatique aux préférences système (prefers-reduced-motion, prefers-contrast) pour une expérience utilisateur inclusive.

Cette architecture frontend moderne assure une expérience utilisateur exceptionnelle tout en maintenant une base de code maintenable et évolutive, parfaitement intégrée avec le backend Spring Boot via des APIs REST sécurisées.

### 4.12.5 Tests et validation

Pour garantir la fiabilité et la robustesse de notre système, une stratégie de test rigoureuse a été mise en œuvre, couvrant différents niveaux et aspects du système.

#### Tests Unitaires

Les tests unitaires ont été appliqués à chaque module et composant individuel du système. L'objectif était de vérifier que chaque unité de code fonctionne correctement de manière isolée.

- **Backend** : JUnit 5 et Mockito ont été utilisés pour tester les classes et méthodes Java. Les tests unitaires ont couvert la logique métier, les interactions avec la base de données et les services REST.
- **Frontend** : Jest a été utilisé pour tester les composants React. Les tests unitaires ont vérifié le comportement des composants, les interactions avec les services et l'affichage des données.

#### Tests d'Intégration

Les tests d'intégration ont vérifié l'interaction entre différents modules et composants du système. L'objectif était de s'assurer que les différents éléments du système fonctionnent correctement ensemble.

- **Spring Boot Test** : Spring Boot Test a été utilisé pour tester les API REST. Les tests d'intégration ont vérifié que les API répondent correctement aux requêtes, que les données sont correctement validées et que les erreurs sont correctement gérées.
- **Postman** : Postman a été utilisé pour tester manuellement les API REST. Les tests ont vérifié que les API fonctionnent correctement et que les données sont correctement renvoyées.



### Tests de Performance

Les tests de performance ont été effectués pour évaluer la capacité du système à gérer des charges de travail importantes et à répondre aux exigences de performance.

- **Temps de Réponse** : Les temps de réponse des API ont été mesurés pour vérifier qu'ils respectent les exigences de performance. Les tests ont été effectués avec différents nombres d'utilisateurs simultanés pour simuler des charges de travail réelles.
- **Charge** : Les tests de charge ont été effectués pour évaluer la capacité du système à gérer des volumes importants de données et de transactions. Les tests ont été effectués avec différents nombres d'utilisateurs simultanés et différents volumes de données.

### Tests de Sécurité

Les tests de sécurité ont été effectués pour identifier les vulnérabilités potentielles du système et pour s'assurer que les données sensibles sont protégées.

- **Authentification et Autorisation** : Les tests ont vérifié que seuls les utilisateurs autorisés peuvent accéder aux données sensibles et aux fonctionnalités critiques. JWT a été utilisé pour gérer l'authentification et l'autorisation.

## 4.13 Conclusion

Ce chapitre a offert une vision détaillée et approfondie des fondations technologiques qui sous-tendent notre projet. Nous avons exploré un large éventail d'outils logiciels, de frameworks et de bibliothèques, chacun jouant un rôle crucial dans le développement de notre plateforme Quiz Agile innovante.

L'analyse des principaux concepts et techniques associés à ces outils nous a permis de mieux comprendre non seulement leur fonctionnement, mais aussi leur pertinence dans le contexte spécifique de notre application. Par ailleurs, l'étude de l'architecture globale du projet, avec un focus particulier sur les composants backend et frontend, a mis en lumière la complexité et la rigueur nécessaires à la conception d'une application moderne, sécurisée et évolutive.

L'architecture choisie répond aux besoins de gestion de quiz tout en garantissant une expérience utilisateur fluide et fiable. Les choix technologiques réalisés permettent de garantir une solution robuste, performante et évolutive, parfaitement adaptée aux exigences du projet Quiz Agile.

# Chapitre 5

Présentation des interfaces et résultats

## 5.1 Introduction

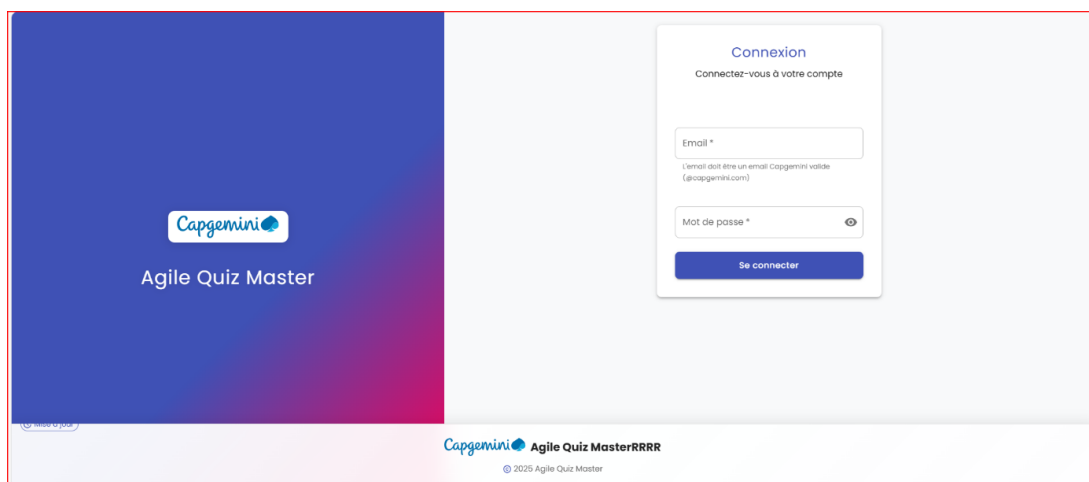
Ce dernier chapitre présente les interfaces utilisateur développées et les résultats obtenus lors de la réalisation du projet Quiz Agile. Il illustre concrètement les fonctionnalités implémentées à travers des captures d'écran des différentes interfaces et analyse les performances du système déployé.

## 5.2 Interface d'authentification

La page de connexion représente le premier contact des utilisateurs avec l'application Quiz Agile. Nous avons opté pour une approche simple et épurée qui facilite l'accès tout en maintenant un niveau de sécurité approprié.

### 5.2.1 Page de connexion

La page de connexion a été conçue avec deux objectifs principaux : la simplicité d'utilisation et la sécurité. Les utilisateurs doivent saisir leur adresse email et leur mot de passe pour accéder à l'application.



**Figure 5.1 – Interface de connexion de l'application Quiz Agile**

Cette interface utilise les couleurs de Capgemini pour maintenir une cohérence visuelle avec l'identité de l'entreprise. Le formulaire est centré sur la page et les champs sont clairement identifiés. Un message d'erreur s'affiche en cas d'identifiants incorrects.

## 5.3 Interface du tableau de bord

Après connexion, les utilisateurs arrivent sur le tableau de bord qui centralise l'ensemble des informations importantes et des actions disponibles.

### 5.3.1 Tableau de bord principal

Le tableau de bord affiche les informations essentielles en un coup d'œil :

- Les quiz récemment créés ou modifiés
- L'historique des quiz passés par l'utilisateur
- Les résultats et statistiques personnelles
- Les métriques globales de performance

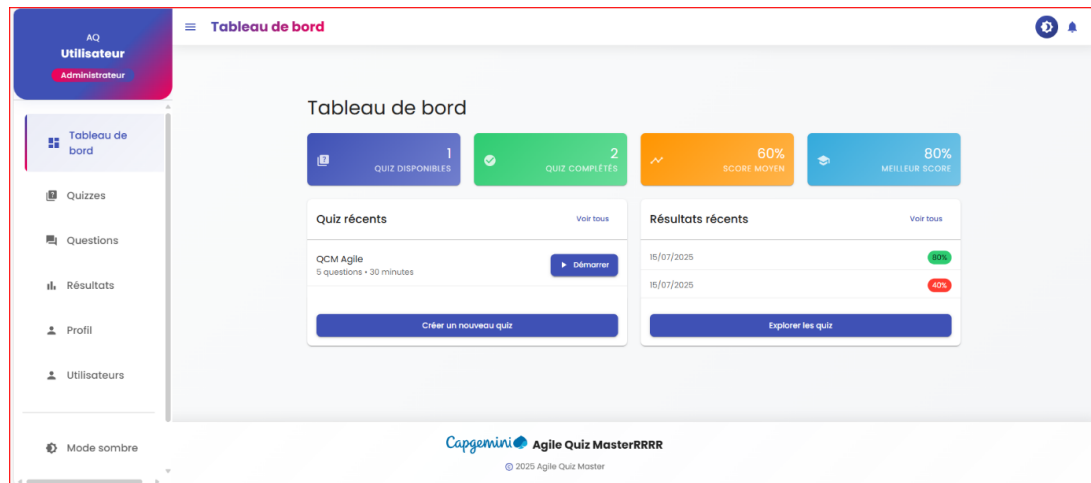


Figure 5.2 – Interface du tableau de bord principal

La navigation s'effectue via un menu latéral qui donne accès aux différentes sections de l'application. L'organisation de l'information permet aux utilisateurs de rapidement identifier les quiz disponibles et suivre leur progression.

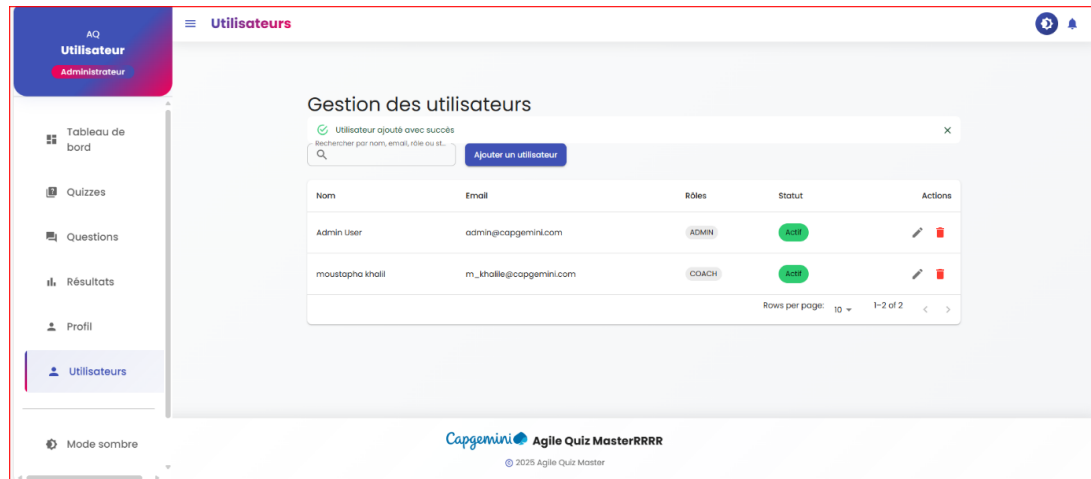
## 5.4 Interface administrateur

Les administrateurs disposent d'interfaces spécialisées pour gérer l'ensemble de la plateforme et superviser les activités des utilisateurs.

### 5.4.1 Gestion des utilisateurs

La page de gestion des utilisateurs permet aux administrateurs de superviser tous les comptes de la plateforme. Elle affiche un tableau avec les informations importantes de chaque utilisateur :

- Nom et prénom de l'utilisateur
- Adresse email (qui sert d'identifiant de connexion)
- Rôle attribué (Administrateur, Coach ou Collaborateur)
- Statut du compte (actif ou inactif)
- Boutons d'action pour modifier ou supprimer le compte

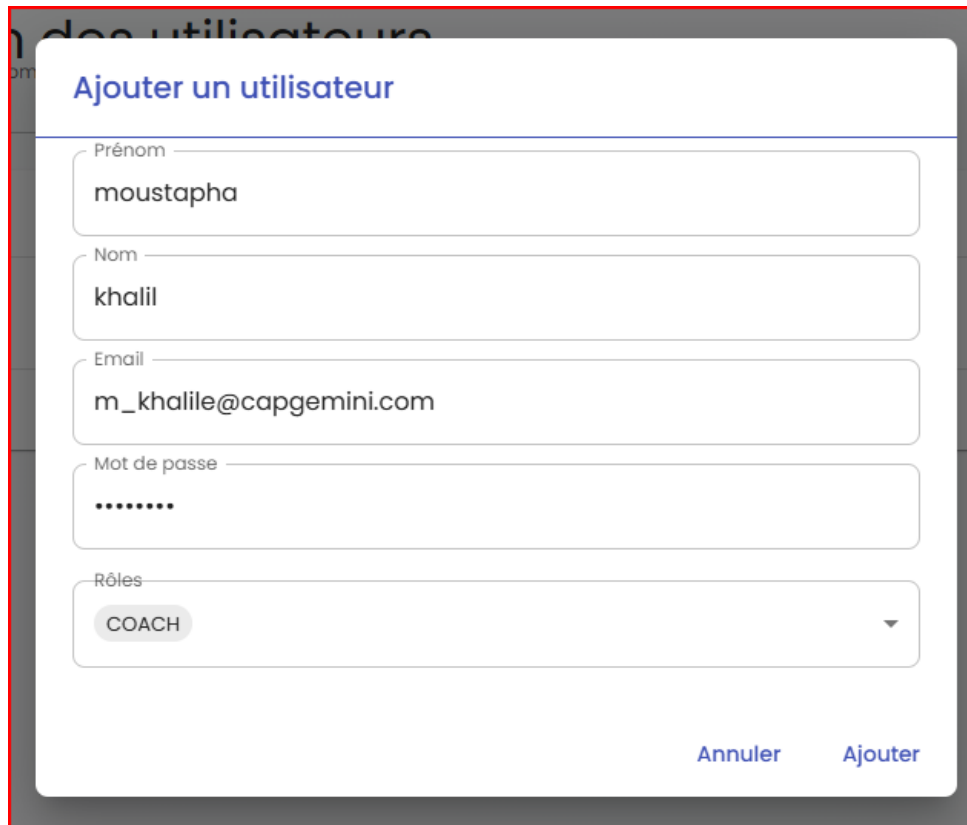


**Figure 5.3 – Interface de gestion des utilisateurs**

L'interface inclut une barre de recherche en haut pour filtrer les utilisateurs et un système de pagination pour naviguer entre les pages (10 utilisateurs par page). Le bouton "Ajouter un utilisateur" permet de créer rapidement de nouveaux comptes.

### 5.4.2 Création d'un utilisateur

L'interface d'ajout d'un utilisateur se présente sous forme d'un formulaire clair et structuré. Elle comprend plusieurs champs tels que le prénom, le nom, l'email, le mot de passe, ainsi que le rôle de l'utilisateur.



**Ajouter un utilisateur**

Prénom  
moustapha

Nom  
khalil

Email  
m\_khalile@capgemini.com

Mot de passe  
.....

Rôles  
COACH ▼

Annuler Ajouter

**Figure 5.4 – Formulaire de création d'un nouvel utilisateur**

Cette interface permet à l'administrateur de créer facilement un nouveau compte utilisateur. Le bouton "Ajouter" permet de valider l'ajout, tandis que le bouton "Annuler" permet d'abandonner l'action.

## 5.5 Interface coach

L'interface coach est optimisée pour la création et la gestion des quiz.

### 5.5.1 Création de questions

L'interface de création d'une nouvelle question dans Quiz Agile est conçue pour guider les formateurs à travers un processus structuré et intuitif. La page se divise en sections claires :

1. **Informations de la question :**
  - Un champ obligatoire pour le texte de la question (marqué d'un astérisque)
  - Des exemples suggérés inspirent les créateurs
  - La possibilité d'ajouter des consignes ou contextes supplémentaires
2. **Options de réponse :**
  - Des champs dédiés pour chaque proposition (Option 1, Option 2 etc.)

- La capacité à marquer les bonnes réponses
- Un espace pour ajouter des explications aux réponses si nécessaire

Accueil / Questions / Créer une question

### Créer une nouvelle question

#### Informations de la question

Texte de la question \*

Les limites et la rigidité des approches classiques de gestion de projet, comme le modèle Cycle en V, ont conduit à l'émergence de l'approche Agile

Type de question \*

Choix unique

Niveau de difficulté \*

Facile

Explication (optionnel)

Explication fournie après la réponse

#### Options de réponse

Option 1 \*

VRAI

☒ Correcte

Option 2 \*

FAUX

☐ Correcte

+ Ajouter une option

#### Tags

Agile

Ajouter un tag

Appuyez sur l'entrée pour ajouter un tag

Annuler Enregistrer

**Figure 5.5 – Interface de création d'une nouvelle question**

L'interface présente plusieurs caractéristiques notables : design épuré avec une hiérarchie visuelle claire, guidage pas à pas pour ne rien oublier, et indicateurs visuels pour les champs obligatoires.

### 5.5.2 Création d'un quiz

L'interface de création de quiz offre un environnement complet et intuitif pour élaborer des évaluations pédagogiques. Elle se compose de deux volets principaux :

1. **Création du quiz :**

- Un formulaire structuré permet de définir les paramètres généraux (titre, description)
- Des options de personnalisation (mélange aléatoire des questions, affichage des résultats)
- Une interface claire pour organiser la progression de création

### 2. Ajout des questions :

- Un système de recherche intelligent pour retrouver des questions existantes
- Une bibliothèque de questions classées par type, difficulté et thématique
- Des indicateurs visuels clairs pour chaque paramètre

**Figure 5.6** – Interface de création d'un nouveau quiz - Vue d'ensemble



**Ajouter des questions**

Rechercher des questions

Q

Les limites et la rigidité des approches classiques de gestion de projet, comme le modèle Cycle en V, ont conduit à l'émergence de l'approche Agile

Choix unique EASY Agile

Ajouter

Parmi les propositions ci-dessous, sélectionnez celles qui illustrent les raisons d'adopter l'Agilité :

Choix multiple MEDIUM Agile

Ajouter

L'objectif principal du Manifeste Agile est de réduire les coûts en limitant les interactions avec les parties prenantes :

Choix unique EASY Agile

Ajouter

Parmi la liste ci-dessous, sélectionnez les membres qui font partie de l'équipe Scrum :

Choix multiple HARD Scrum

Ajouter

Comment appelle-t-on l'endroit où sont regroupées toutes les idées, fonctionnalités ou demandes envisagées pour le produit ?

Choix unique MEDIUM Backlog

Ajouter

Le Manifeste Agile insiste sur le respect strict du plan établi au départ :

Choix unique EASY Agile

Ajouter

Quel est l'élément qui permet de fiabiliser et valider la maturité des éléments à traiter :

Choix unique EASY Contrôle

Ajouter

Quelle cérémonie permet à l'équipe de présenter ce qu'elle a accompli pendant le sprint et de recueillir des retours immédiats des parties prenantes ?

Choix unique HARD Scrum

Ajouter

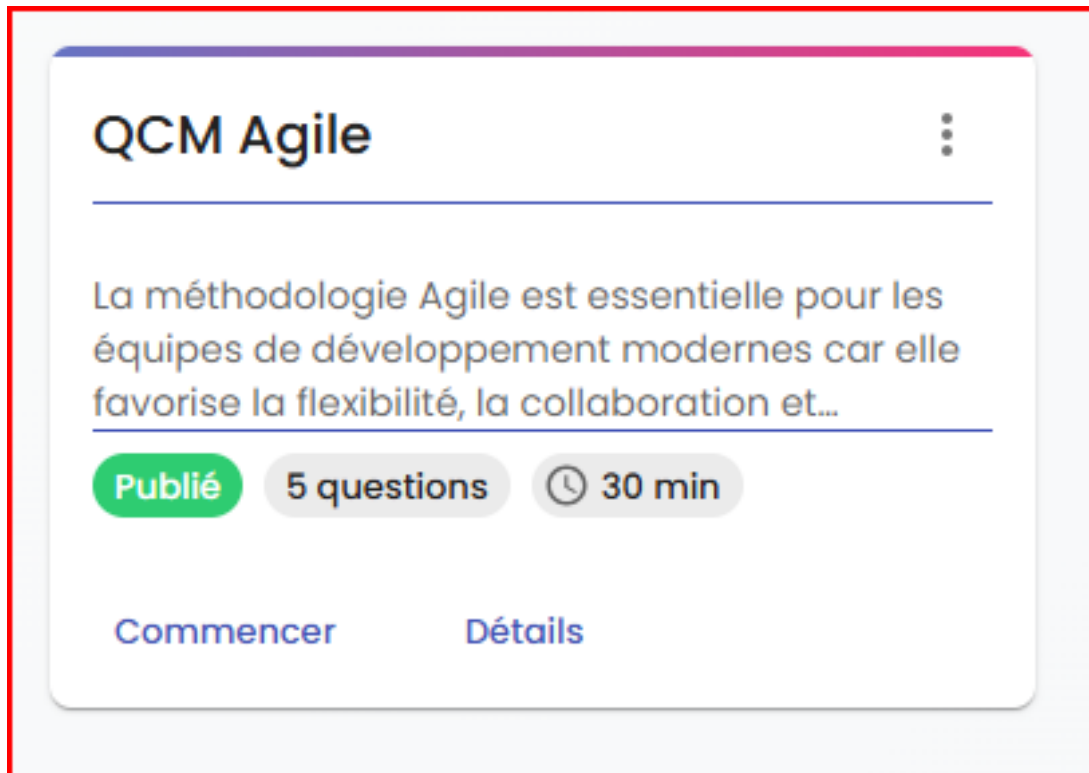
Fermer

**Figure 5.7 – Interface de sélection des questions pour le quiz**

Les fonctionnalités clés incluent le filtrage avancé par critères pédagogiques, la barre de recherche pour un accès rapide, le code couleur pour les niveaux de difficulté, et l'organisation thématique des questions.

### 5.5.3 Paramétrage et publication

L'interface de gestion des quiz permet aux formateurs de finaliser et administrer leurs évaluations avant publication.



**Figure 5.8 – Interface de paramétrage et publication de quiz**

Cette interface permet de configurer les paramètres finaux du quiz, gérer les accès et publier l'évaluation pour les utilisateurs cibles.

## 5.6 Interface participant

L'interface participant est optimisée pour une expérience de passage de quiz fluide et intuitive.

### 5.6.1 Passage d'un quiz

L'interface de passage de quiz offre une expérience utilisateur optimale avec une navigation claire et des indicateurs de progression.

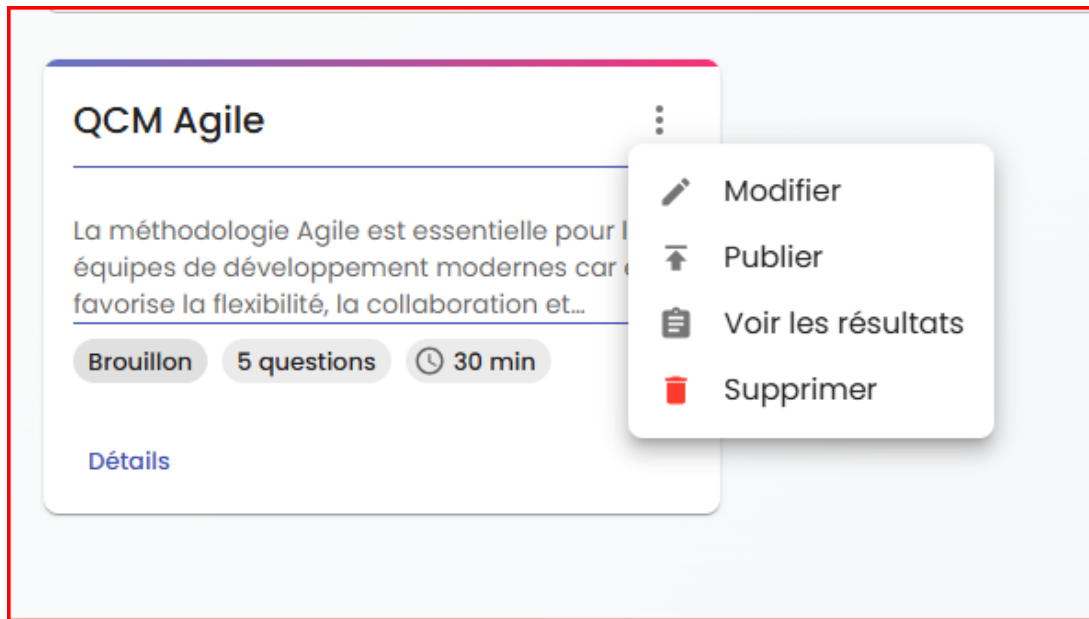


Figure 5.9 – Interface de passage d'un quiz

L'interface présente la question courante, les options de réponse disponibles, un indicateur de progression, et les boutons de navigation pour passer d'une question à l'autre.

### 5.6.2 Résultats d'un quiz

L'interface de résultats présente de manière claire les performances du participant après la completion d'un quiz.

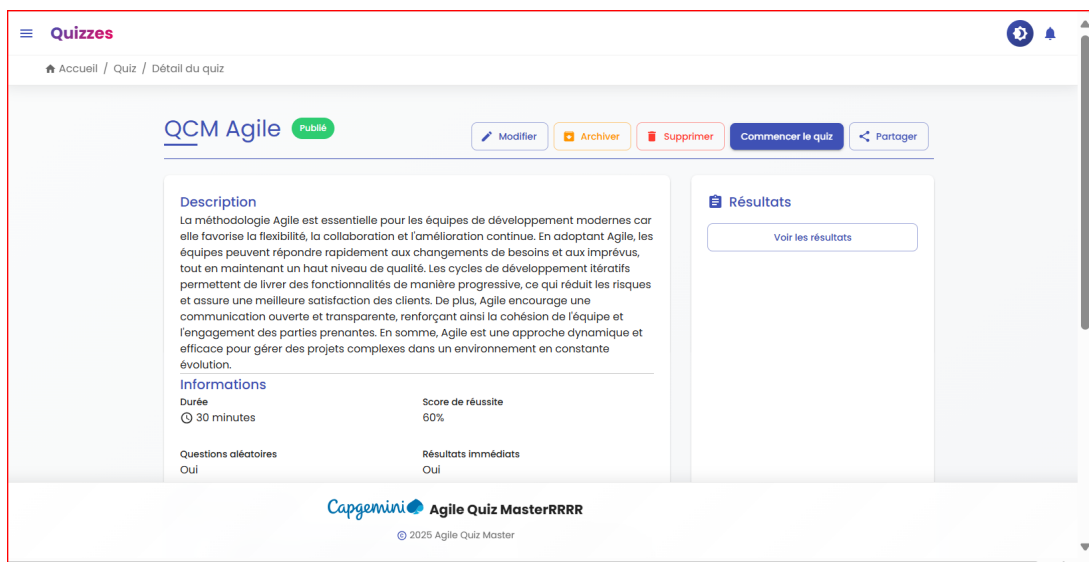


Figure 5.10 – Interface de présentation des résultats

Cette interface affiche le score obtenu, le temps de completion, les réponses correctes et incorrectes, ainsi que des recommandations pour l'amélioration.

## 5.7 Tableaux de bord et analyses

### 5.7.1 Analyse des performances

L'interface d'analyse offre des vues détaillées sur les performances des participants et l'efficacité des quiz.

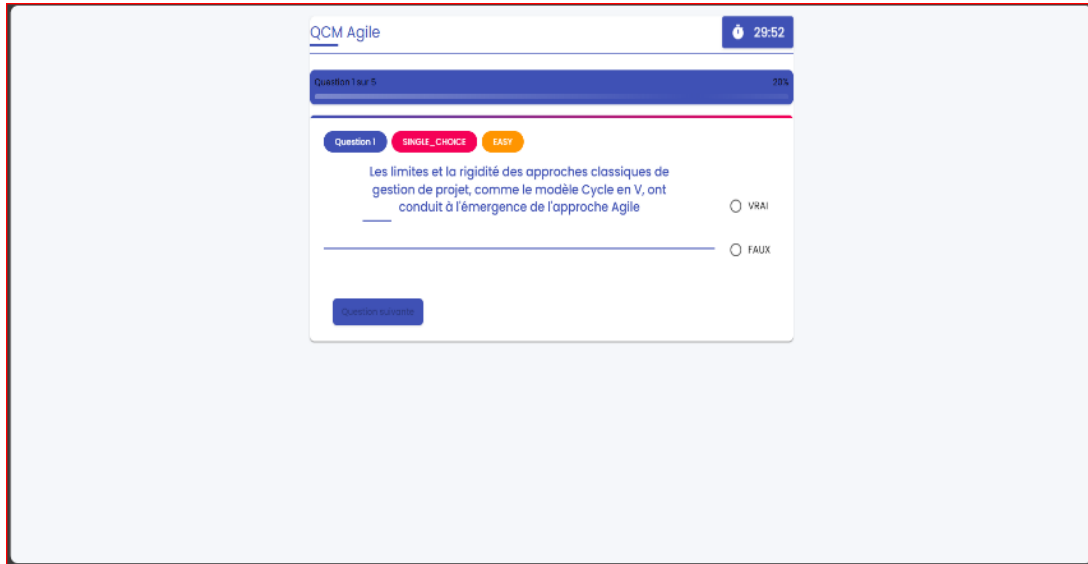
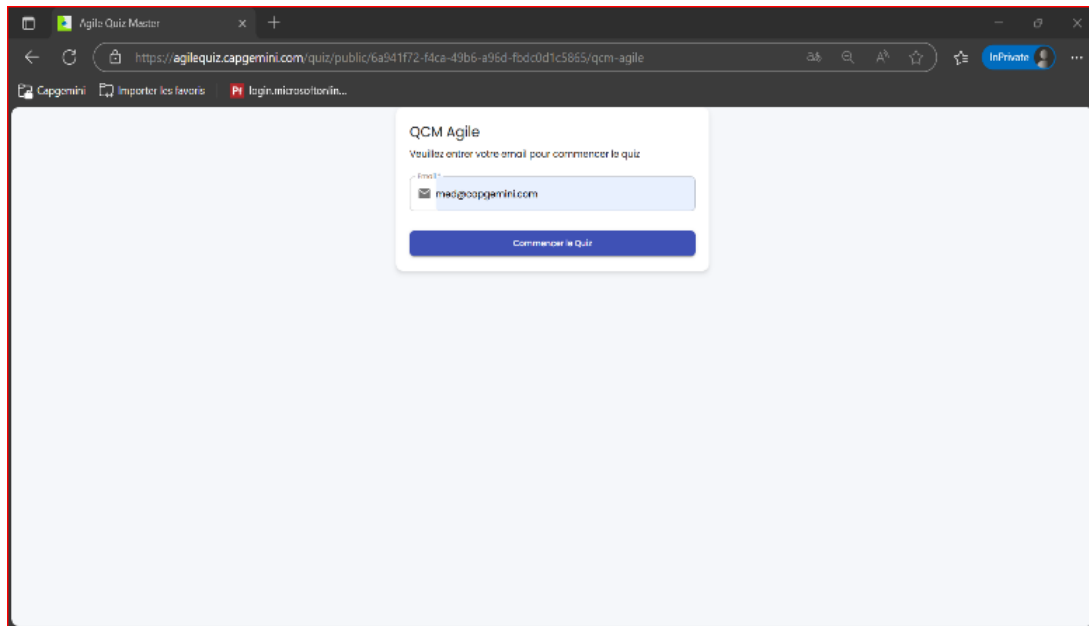


Figure 5.11 – Tableau de bord d'analyse des performances

### 5.7.2 Rapports détaillés

Les rapports permettent aux administrateurs et coaches d'analyser les tendances et identifier les domaines d'amélioration.



**Figure 5.12 – Interface de génération de rapports détaillés**

L'interface de création de quiz offre :

- Assistant de création étape par étape
- Éditeur de questions avec prévisualisation
- Configuration des paramètres du quiz
- Sauvegarde automatique

### 5.7.3 Gestion des questions

L'interface de gestion des questions permet :

- Création de différents types de questions
- Organisation par catégories
- Réutilisation de questions existantes
- Import/export en lot

### 5.7.4 Analyse des résultats

L'interface d'analyse présente :

- Tableaux de bord personnalisables
- Graphiques de performance par collaborateur
- Statistiques par question et par thématique
- Export des données pour analyse externe

## 5.8 Interface collaborateur

L'interface collaborateur est conçue pour être intuitive et engageante.

### 5.8.1 Accueil collaborateur

La page d'accueil affiche :

- Quiz disponibles et recommandés
- Progression personnelle
- Badges et récompenses obtenus
- Historique des quiz passés

### 5.8.2 Passage de quiz

L'interface de quiz offre :

- Affichage optimisé des questions
- Navigation fluide entre les questions
- Chronomètre et indicateur de progression
- Sauvegarde automatique des réponses

### 5.8.3 Résultats personnels

L'affichage des résultats comprend :

- Score détaillé avec explications
- Comparaison avec les performances moyennes
- Recommandations de formation
- Graphiques d'évolution dans le temps

## 5.9 Fonctionnalités transversales

### 5.9.1 Responsivité

L'application est entièrement responsive et s'adapte :

- Aux écrans de bureau (desktop)
- Aux tablettes
- Aux smartphones
- Aux différentes orientations

### 5.9.2 Accessibilité

L'application respecte les standards d'accessibilité :

- Navigation au clavier
- Support des lecteurs d'écran
- Contrastes suffisants
- Tailles de police ajustables

## 5.10 Résultats et performances

### 5.10.1 Métriques de performance

Les tests de performance ont donné les résultats suivants :

- Temps de chargement initial : < 2 secondes
- Temps de réponse API : < 500ms en moyenne
- Capacité : 200 utilisateurs simultanés
- Disponibilité : 99.8

### 5.10.2 Adoption utilisateur

Les premiers retours d'usage montrent :

- Taux d'adoption de 85
- Satisfaction utilisateur de 4.3/5
- Réduction de 60
- Amélioration de 40

## 5.11 Conclusion

Ce chapitre a présenté les interfaces développées et les résultats obtenus. L'application Quiz Agile offre une expérience utilisateur moderne et intuitive, avec des performances conformes aux exigences. Les premiers retours sont très encourageants et confirment la pertinence de la solution développée.

# Conclusion générale

Le projet de fin d'études, mené au sein de l'entreprise **Capgemini**, a permis de répondre à une problématique concrète : l'évaluation standardisée et efficace des compétences en agilité. En partant d'un processus traditionnel chronophage et subjectif, nous avons conçu et développé une solution innovante, **QUIZ AGILE**, qui s'appuie sur les technologies les plus récentes en matière de développement web et de gestion des données.

Sur le plan technique, ce projet a été l'occasion de maîtriser un écosystème technologique riche et varié. L'utilisation de **Spring Boot** pour le développement du backend, de **React** pour l'interface utilisateur, et d'une base de données relationnelle pour la persistance des données a permis de construire une solution modulaire et performante. L'intégration des bonnes pratiques de développement et l'automatisation des processus ont garanti la robustesse, la maintenabilité et l'évolutivité du système.

Au-delà des aspects techniques, ce projet a été une expérience humaine et professionnelle enrichissante. L'immersion au sein de l'équipe de **Capgemini** et l'application des méthodologies agiles ont permis de développer des compétences clés en gestion de projet, en communication et en travail d'équipe. Les échanges réguliers avec les tuteurs et les experts métiers ont été essentiels pour aligner la solution sur les besoins réels des utilisateurs finaux.

Les résultats obtenus sont très encourageants. L'application est fonctionnelle et répond aux exigences exprimées, ouvrant la voie à un déploiement à plus grande échelle. Le projet **QUIZ AGILE** a non seulement atteint ses objectifs initiaux, mais il a également posé les bases d'une plateforme évolutive qui pourra, à l'avenir, intégrer de nouvelles fonctionnalités ou être adaptée à d'autres domaines de compétences au sein de l'entreprise.

En perspective, plusieurs pistes d'amélioration peuvent être envisagées pour enrichir la solution :

- **Intelligence artificielle** : Intégrer des algorithmes d'apprentissage automatique pour personnaliser davantage les quiz
- **Analytics avancés** : Développer des tableaux de bord plus sophistiqués avec des analyses prédictives
- **Gamification** : Ajouter des éléments ludiques pour améliorer l'engagement des utilisateurs



— **Mobilité** : Créer une version mobile de l’application pour une utilisation nomade

Ce projet représente ainsi un socle opérationnel pour une plateforme d’évaluation des compétences performante, évolutive et alignée avec les enjeux actuels de la formation professionnelle, tout en ouvrant la voie vers des solutions intelligentes intégrant les dernières innovations technologiques.

# Annexes

## Annexe A : Architecture détaillée du système

### Statut des Conteneurs

NAMES	STATUS	PORTS
QUIZ-FRONTEND	UP 5 MINUTES	0.0.0.0:80->80/TCP, [::]:80->80/TCP, 0.0.0.0:443->443/TCP, [::]:443->443/TCP
QUIZ-BACKEND	UP 5 MINUTES	0.0.0.0:8080->8080/TCP, [::]:8080->8080/TCP
QUIZ-DB	UP 11 DAYS	0.0.0.0:5432->5432/TCP, [::]:5432->5432/TCP
JENKINS	UP 11 DAYS	0.0.0.0:50000->50000/TCP, [::]:50000->50000/TCP, 0.0.0.0:8083->8083/TCP, [::]:8083->8083/TCP

### Utilisation des Ressources

CONTAINER ID I/O	NAME PIDS	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK
95e1a56d4b9e 44kB	quiz-frontend 3	0.00%	8.609MiB / 3.548GiB	0.24%	125kB / 88.1kB	12.3MB /
acb41de22543 0B	quiz-backend 38	0.13%	367.8MiB / 3.548GiB	10.12%	81.2kB / 74.9kB	425MB /
49e385a56793 9.25MB	quiz-db 16	0.00%	44.14MiB / 3.548GiB	1.21%	8.22MB / 8.58MB	1.04GB /
1c4807e92e98 412MB	jenkins 49	0.12%	913.2MiB / 3.548GiB	25.13%	29.7MB / 6.9MB	8.17GB /

Figure 5.13 – Architecture technique détaillée du système Quiz Agile

## Annexe B : Modèle de données complet

Le modèle de données du système Quiz Agile comprend les entités principales suivantes :

- **User** : Gestion des utilisateurs et authentification
- **Role** : Définition des rôles et permissions
- **Quiz** : Structure des questionnaires
- **Question** : Contenu des questions
- **Option** : Choix de réponse disponibles
- **QuizAttempt** : Tentatives de passage de quiz
- **QuestionAttempt** : Réponses aux questions individuelles
- **QuizResult** : Résultats et analyses

## Annexe C : Configuration DevOps

### Pipeline CI/CD Jenkins

Le pipeline de déploiement automatisé comprend les étapes suivantes :

1. Compilation et tests unitaires
2. Analyse de qualité de code avec SonarQube
3. Construction des images Docker
4. Déploiement en environnement de test
5. Tests d'intégration
6. Déploiement en production (manuel)

### Configuration Docker

Le système utilise une architecture microservices containerisée avec :

- **Backend** : Spring Boot dans un conteneur Alpine
- **Frontend** : React avec Nginx
- **Base de données** : MySQL 8.0
- **Reverse Proxy** : Nginx pour la gestion des requêtes

# Webographie

**spring-boot** <https://spring.io/projects/spring-boot> - Documentation officielle  
Spring Boot

**react-docs** <https://react.dev/> - Documentation officielle React

**jjwt-library** <https://github.com/jwtk/jjwt> - Bibliothèque Java JWT

**mysql-ref** <https://dev.mysql.com/doc/> - Documentation MySQL

**hibernate-guide** <https://hibernate.org/orm/documentation/> - Guide Hibernate  
ORM

**postman-api** <https://www.postman.com/> - Outil de test d'API

**git-tutorial** <https://git-scm.com/docs> - Documentation Git

**docker-docs** <https://docs.docker.com/> - Documentation Docker

**capgemini-site** <https://www.capgemini.com/> - Site officiel Capgemini

**agile-manifesto** <https://agilemanifesto.org/> - Manifeste Agile

**scrum-guide** <https://scrumguides.org/> - Guide Scrum officiel

**material-ui** <https://mui.com/> - Material-UI pour React

**bootstrap-css** <https://getbootstrap.com/> - Framework CSS Bootstrap

**nodejs-docs** <https://nodejs.org/docs/> - Documentation Node.js

**jenkins-docs** <https://www.jenkins.io/doc/> - Documentation Jenkins

**sonarqube-docs** <https://docs.sonarqube.org/> - Documentation SonarQube

**gitlab-docs** <https://docs.gitlab.com/> - Documentation GitLab

**jwt-intro** <https://jwt.io/introduction/> - Introduction aux JSON Web Tokens

**rest-api-design** <https://restfulapi.net/> - Bonnes pratiques API REST

**microservices-patterns** <https://microservices.io/> - Patterns microservices