Cahier des charges CRM Logistique LiVrai

Diffusion : OpenClassrooms – Mentor : Jeoffrey Hernandez - Evaluateur :

Version	***Date***	***Auteur***	***Motif***
1	28/05/2025	Christophe Pierrès	Création du document
2	06/06/2025	Christophe Pierrès	Ajout use case avec include et exclude Disponibilité avec Kubernetes

Contenu **Page**

1. Contexte du projet 4
1.1. Besoins 4
1.1.1. Présentation de l'activité de l'entreprise et de la solution existante 4
1.1.2. Limites de la solution existante 4
1.2. Objectifs du projet 4
1.2.1. Mesure de la réussite du projet 5
1.2.1.1. Adoption par les utilisateurs 5
1.2.1.2. Performance technique 5
1.2.1.3. Sécurité 5
1.2.1.4. Satisfaction utilisateur 5
1.2.2. Contraintes réglementaires 5
1.2.2.1. RGPD (Règlement Général sur la Protection des Données) : 5
1.2.2.2. Accessibilité : 5
1.2.2.3. Sécurité des données : 5
1.2.3. Éléments hors périmètre 6
2. SFG : Spécifications Fonctionnelles Générales 7
2.1. Liste des fonctionnalités 7
2.1.1. Gestion des utilisateurs et des comptes 7
2.1.1.1. Inscription autonome des clients 7
2.1.1.2. Gestion des profils 7
2.1.1.3. Administration des comptes 7
2.1.1.4. Authentification et sécurité 7
2.1.2. Gestion des livraisons 7
2.1.2.1. Création de demandes de livraison 7
2.1.2.2. Suivi des livraisons 7
2.1.2.3. Traitement des livraisons (service Livraisons) 8
2.1.3. Gestion de la facturation 8
2.1.3.1. Établissement des factures 8
2.1.3.2. Consultation des factures 8
2.1.3.3. Reporting financier (service commercial) 8
2.2. Fonctionnement de l'application 9
2.2.1. Diagrammes de cas d'utilisation 9
2.2.2. Diagramme de séquence - Processus de livraison 11
2.2.3. Diagramme d'activité - Processus d'inscription et commande 12
2.2.4. Impact Social 12
2.2.4.1. Impact environnemental 12
2.2.4.2. Accessibilité 13
3. ST : Spécifications Techniques 14
3.1. Technologies utilisées 14
3.1.1. Base de données : MySQL 8.0 14
3.1.2. Backend : Spring Boot 3.4.4 (basé sur Spring 6) 14
3.1.2.1. Module pour couche d'exposition des API : Spring Web 14
3.1.2.2. Module de persistance : Spring Data JPA (base sur Hibernate) 14
3.1.2.3. Modules : Spring Security, OAuth2 Resource Server, Spring Security OAuth2 JOSE (JSON Object Signing and Encryption) 15
Informations complémentaires sur la sécurité 16
3.1.2.4. Java 17 (LTS) 18
3.1.3. Frontend : Angular v19.2.5 18
3.1.3.1. Librairie de composants visuels : Angular Materials (+ éventuel complément) 19
3.1.3.2. TypeScript 20
3.1.3.3. RxJs (en complément de l'API Signal incluse dans Angular) 20
3.1.4. Déploiement 21
3.1.5. Testing 21

3.1.5.1. Front-end testing 21
3.1.5.2. Back-end testing 22
3.2. Architecture de l'application 23
3.2.1. Architecture détaillée 24
3.2.1.1. Architecture Frontend (Angular) 24
3.2.1.2. Architecture Backend (Spring Boot) 24
3.2.1.3. Modèle de données 24
3.2.1.4. Communication Frontend-Backend 24
3.2.1.5. Déploiement 24
Possibilité de disponibilité horizontale grâce à Kubernetes 25
3.2.2. Modèle de données 26

1. Contexte du projet

1.1. Besoins

1.1.1. Présentation de l'activité de l'entreprise et de la solution existante

LiVrai est une société de 150 employés, spécialisée dans la livraison de marchandises en grande quantité pour les professionnels. L'entreprise connaît actuellement une forte croissance et souhaite étendre son offre à tous types de colis, ce qui entraînera une augmentation significative du nombre de clients.

Actuellement, LiVrai utilise une application web développée il y a plusieurs années pour gérer ses clients et leurs livraisons. Cette application permet aux administrateurs de créer des comptes clients, et aux clients de se connecter pour réserver des livraisons. Le processus actuel impose que chaque nouveau client passe par le service commercial pour établir un contrat de collaboration, après quoi le service commercial crée manuellement un compte sur l'application.

1.1.2. Limites de la solution existante

L'audit de l'application existante a révélé plusieurs limitations importantes :

- Processus client non autonome : Les clients ne peuvent pas créer leur propre compte et doivent passer par le service commercial, qui se retrouve inutilement sollicité.
- Interface utilisateur obsolète : L'application utilise des technologies anciennes (JSP/Servlets) avec une interface utilisateur basique et non responsive.
- Sécurité insuffisante : Les mots de passe sont stockés en clair dans la base de données (risque majeur de sécurité).
- Architecture monolithique : L'application ne sépare pas clairement le frontend du backend, ce qui limite l'évolutivité.
- Fonctionnalités limitées : Absence de fonctionnalités avancées comme la recherche, le filtrage ou les notifications.
- Configuration rigide : Les paramètres de connexion à la base de données sont codés en dur.
- Tests limités : Faible couverture de tests automatisés.

1.2. Objectifs du projet

La refonte complète de l'application vise à :

- Moderniser l'architecture technique en adoptant une approche SPA (Single-Page Application) pour le frontend et une API REST pour le backend.
- Améliorer l'expérience utilisateur avec une interface moderne, responsive et intuitive.
- Renforcer la sécurité notamment par le chiffrement des mots de passe et la mise en place de bonnes pratiques de sécurité.
- Permettre l'autonomie des clients en leur donnant la possibilité de créer leur propre compte et de gérer leurs informations.
- Optimiser les processus métier en différenciant les clients gérés par le service commercial des clients autonomes.
- Faciliter la maintenance et l'évolution de l'application grâce à une architecture modulaire et des tests automatisés.
- Préparer l'application à la croissance de l'entreprise et à l'augmentation du volume de clients et de livraisons.

1.2.1. Mesure de la réussite du projet

Le succès du projet sera mesuré selon les critères suivants :

1.2.1.1. Adoption par les utilisateurs

- 80% des nouveaux clients créent leur compte de manière autonome
- Réduction de 70% du temps consacré par le service commercial à la création de comptes clients

1.2.1.2. Performance technique

- Temps de chargement des pages < 2 secondes
- Application capable de gérer 1 000 utilisateurs simultanés
- Taux de disponibilité > 99,9%

1.2.1.3. Sécurité

- Aucune vulnérabilité critique identifiée lors des audits de sécurité
- Conformité totale au RGPD

1.2.1.4. Satisfaction utilisateur

- Score de satisfaction client > 8/10
- Réduction de 60% des demandes de support liées à l'utilisation de l'application

1.2.2. Contraintes réglementaires

L'application devra être conforme aux réglementations suivantes :

1.2.2.1. RGPD (Règlement Général sur la Protection des Données) :

- Recueil du consentement explicite des utilisateurs
- Droit à l'accès, à la rectification et à l'effacement des données
- Mise en place d'une politique de confidentialité claire
- Mise en place de mesures de sécurité appropriées
- Journalisation des accès aux données personnelles

1.2.2.2. Accessibilité :

- Conformité aux normes WCAG 2.1 niveau AA
- Support des technologies d'assistance

1.2.2.3. Sécurité des données :

- Chiffrement des données sensibles

- Authentification sécurisée
- Protection contre les attaques courantes (CSRF, XSS, injection SQL)

1.2.3. Éléments hors périmètre

Les éléments suivants sont exclus du périmètre du projet :

- Intégration avec des systèmes externes de logistique (place de marché) - Pourra être développée dans une phase ultérieure
- Application mobile native - Une version responsive web sera développée, mais pas d'application mobile native
- Système de paiement en ligne - La facturation restera gérée par le système comptable existant
- Système de Datawarehouse (Business Intelligence) - L'analyse avancée des données sera traitée dans un projet futur
- Migration des données historiques - Seules les données actives seront migrées

2. SFG : Spécifications Fonctionnelles Générales

2.1. Liste des fonctionnalités

2.1.1. Gestion des utilisateurs et des comptes

2.1.1.1. Inscription autonome des clients

- Formulaire d'inscription avec validation des données
- Confirmation par email
- Acceptation des conditions d'utilisation et politique de confidentialité

2.1.1.2. Gestion des profils

- Modification des informations personnelles
- Changement de mot de passe
- Gestion des préférences de notification

2.1.1.3. Administration des comptes

- Création de comptes clients par le service commercial
- Désignation des clients "gérés" versus "autonomes"
- Suspension/réactivation de comptes

2.1.1.4. Authentification et sécurité

- Connexion sécurisée
- Récupération de mot de passe
- Déconnexion automatique après inactivité

2.1.2. Gestion des livraisons

2.1.2.1. Création de demandes de livraison

- Formulaire de demande avec calcul automatique des coûts estimés
- Sélection de dates préférentielles
- Spécification des caractéristiques du colis (dimensions, poids, nature)

2.1.2.2. Suivi des livraisons

- Tableau de bord des livraisons en cours
- Historique des livraisons passées
- Filtres et recherche avancée
- Notifications de changement de statut

2.1.2.3. Traitement des livraisons (service Livraisons)

- Acceptation/refus des demandes
- Planification des livraisons
- Mise à jour du statut des livraisons
- Ajout de commentaires internes

2.1.3. Gestion de la facturation

2.1.3.1. Établissement des factures

- Génération automatique de devis
- Création de factures pour les livraisons terminées
- Application de tarifs spécifiques selon le client

2.1.3.2. Consultation des factures

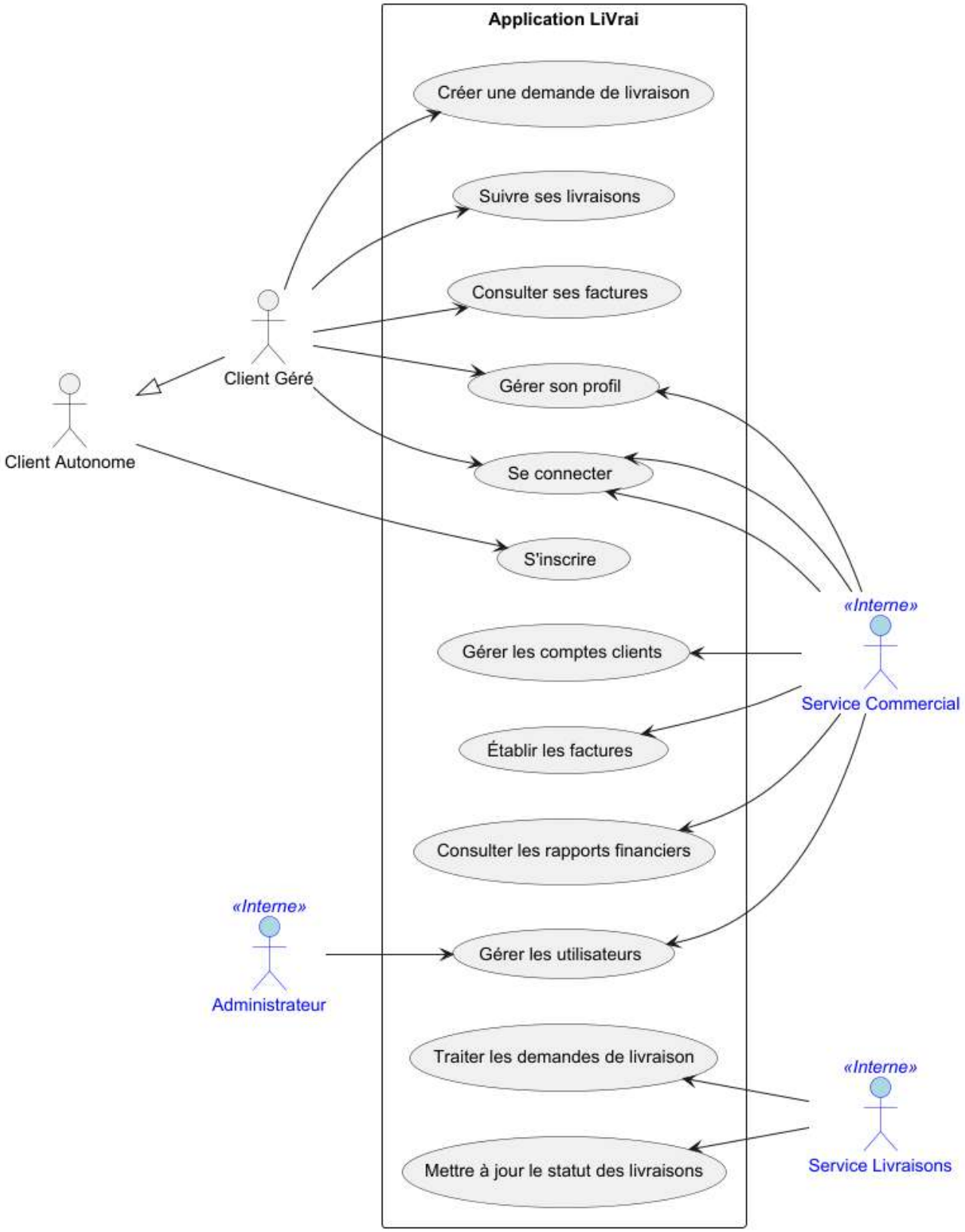
- Visualisation des factures par le client
- Export en PDF
- Historique des factures

2.1.3.3. Reporting financier (service commercial)

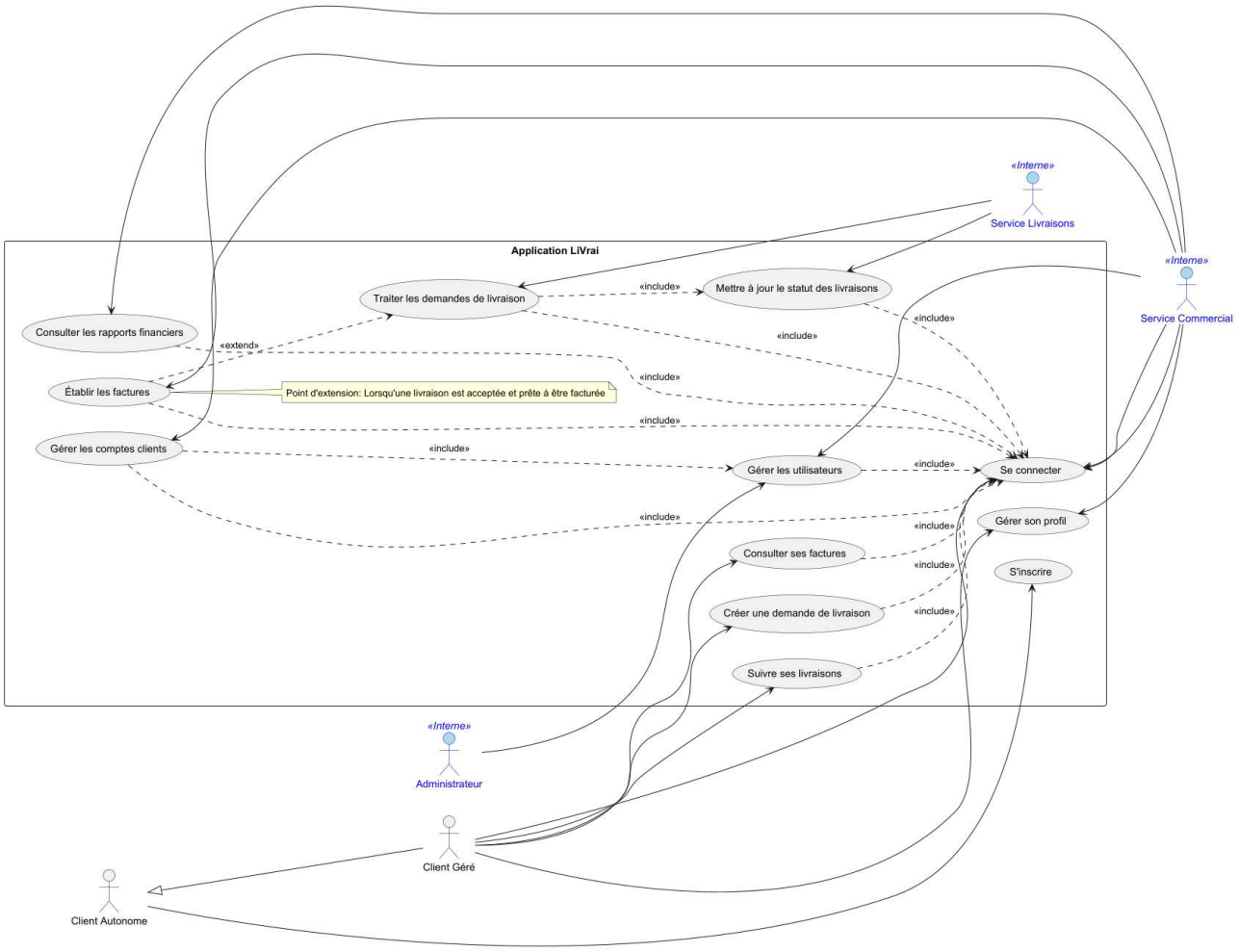
- Tableau de bord des revenus
- Statistiques par client
- Rapports périodiques

2.2. Fonctionnement de l'application

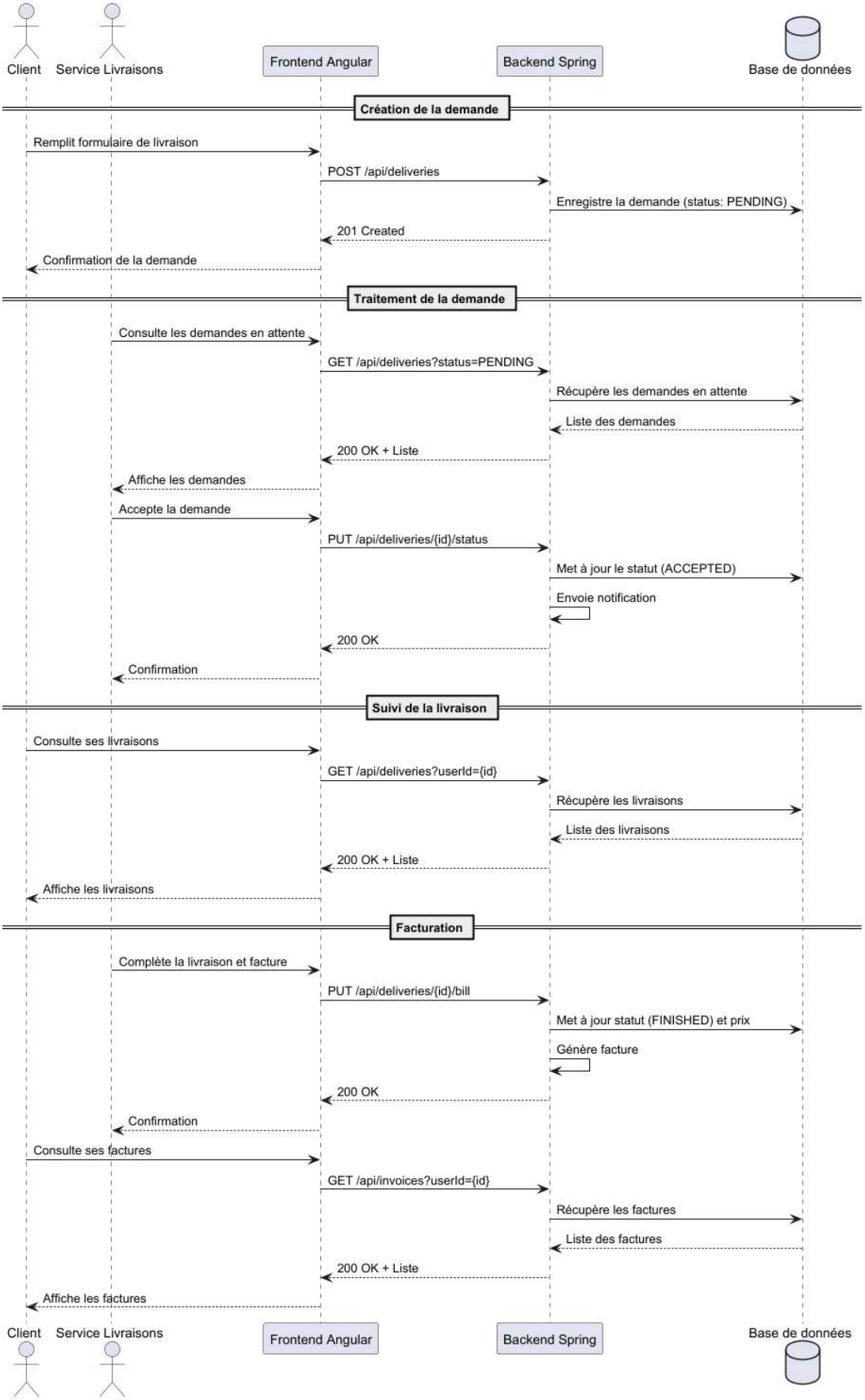
2.2.1. Diagrammes de cas d'utilisation



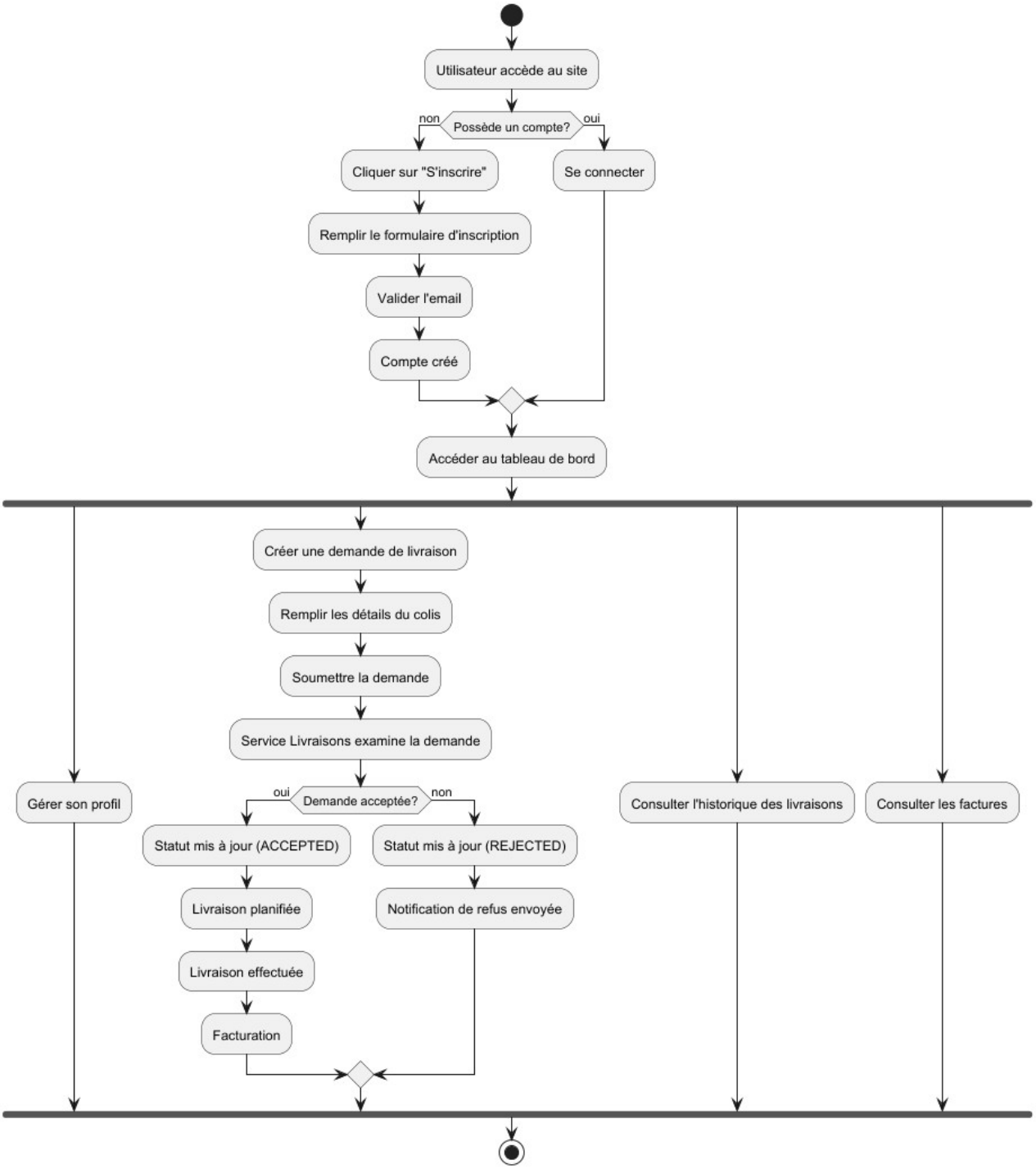
Version plus complète avec include et exclude :



2.2.2. Diagramme de séquence - Processus de livraison



2.2.3. Diagramme d'activité - Processus d'inscription et commande



2.2.4. Impact Social

2.2.4.1. Impact environnemental

La nouvelle application contribuera à réduire l'impact environnemental de l'entreprise par :

- Réduction de la consommation de papier : Dématérialisation complète des processus de commande et de facturation.

- Optimisation des trajets de livraison : Meilleure planification des livraisons grâce à des données plus précises sur les colis.
- Réduction des déplacements clients : L'autonomie des clients via l'application web élimine le besoin de déplacements pour établir des contrats.
- Efficacité énergétique : L'architecture moderne de l'application permettra une utilisation plus efficace des ressources serveur.

2.2.4.2. Accessibilité

L'application sera conçue pour être accessible aux personnes en situation de handicap :

- Conformité WCAG 2.1 niveau AA : Respect des normes d'accessibilité web.
- Support des technologies d'assistance : Compatibilité avec les lecteurs d'écran et autres technologies d'assistance.
- Design inclusif : Interface utilisateur conçue pour être utilisable par tous, avec des contrastes suffisants, des tailles de texte ajustables, et des alternatives textuelles pour les éléments non-textuels.
- Tests d'accessibilité : Réalisation de tests avec des utilisateurs en situation de handicap pour valider l'accessibilité réelle de l'application.

3. ST : Spécifications Techniques

3.1. Technologies utilisées

Reportez-vous à notre site de veille technologique : https://veille.cpierres.dscloud.me/ si vous souhaitez en savoir plus par rapport aux choix proposés et alternatifs.

3.1.1. Base de données : MySQL 8.0

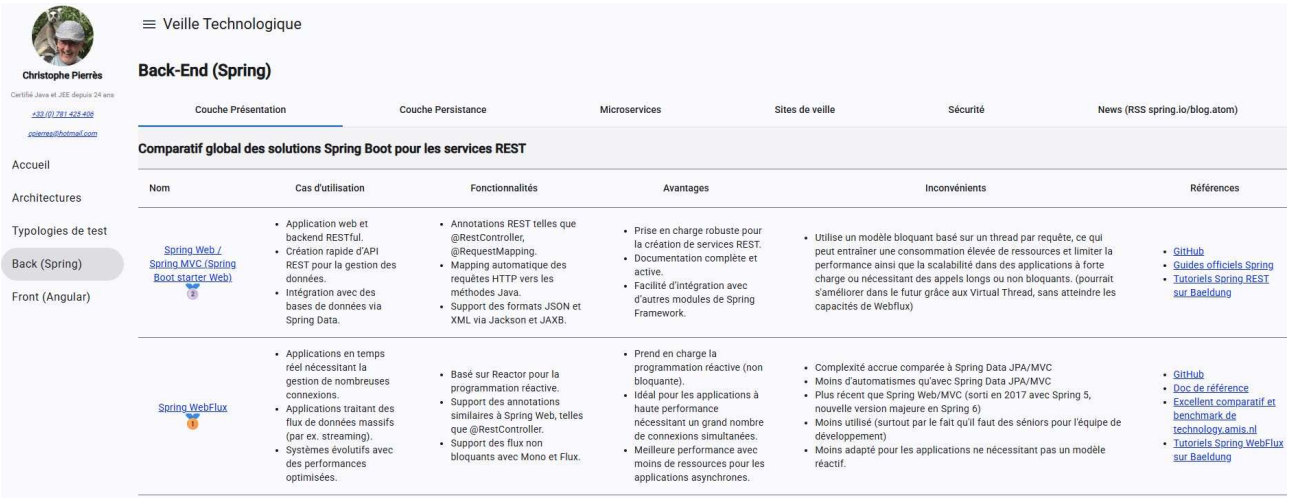
- Justification : SGBDR éprouvé, déjà utilisé dans l'application existante.
- Avantages : Compatibilité avec les outils existants, familiarité pour l'équipe.

3.1.2. Backend : Spring Boot 3.4.4 (basé sur Spring 6)

- Justification : Framework Java moderne facilitant le développement d'applications robustes avec un minimum de configuration.
- Avantages : Configuration automatique, serveur Tomcat embarqué, gestion des dépendances simplifiée, écosystème riche.

SpringBoot d'une version donnée permet de choisir parmi des **modules** en assurant leur cohérence de version et d'intégration.

3.1.2.1. Module pour couche d'exposition des API : Spring Web



Christophe Pierrès

Certifié Java et JEE depuis 24 ans

+33 (0) 781 425 406

cpierres@hotmail.com

Accueil

Architectures

Typologies de test

Back (Spring)

Front (Angular)

≡ Veille Technologique

Back-End (Spring)

Couche Présentation | Couche Persistance | Microservices | Sites de veille | Sécurité | News (RSS spring.io/blog.atom)

Comparatif global des solutions Spring Boot pour les services REST

Nom	Cas d'utilisation	Fonctionnalités	Avantages	Inconvénients	Références
Spring Web / Spring MVC (Spring Boot starter Web)	• Application web et backend RESTful. • Création rapide d'API REST pour la gestion des données. • Intégration avec des bases de données via Spring Data.	• Annotations REST telles que @RestController, @RequestMapping. • Mapping automatique des requêtes HTTP vers les méthodes Java. • Support des formats JSON et XML via Jackson et JAXB.	• Prise en charge robuste pour la création de services REST. • Documentation complète et active. • Facilité d'intégration avec d'autres modules de Spring Framework.	• Utilise un modèle bloquant basé sur un thread par requête, ce qui peut entraîner une consommation élevée de ressources et limiter la performance ainsi que la scalabilité dans des applications à forte charge ou nécessitant des appels longs ou non bloquants. (pourrait s'améliorer dans le futur grâce aux Virtual Thread, sans atteindre les capacités de Webflux)	• GitHub • Guides officiels Spring • Tutoriels Spring REST sur Baeldung
Spring WebFlux	• Applications en temps réel nécessitant la gestion de nombreuses connexions. • Applications traitant des flux de données massifs (par ex. streaming). • Systèmes évolutifs avec des performances optimisées.	• Basé sur Reactor pour la programmation réactive. • Support des annotations similaires à Spring Web, telles que @RestController. • Support des flux non bloquants avec Mono et Flux.	• Prend en charge la programmation réactive (non bloquante). • Idéal pour les applications à haute performance nécessitant un grand nombre de connexions simultanées. • Meilleure performance avec moins de ressources pour les applications asynchrones.	• Complexité accrue comparée à Spring Data JPA/MVC • Moins d'automatismes qu'avec Spring Data JPA/MVC • Plus récent que Spring Web/MVC (sorti en 2017 avec Spring 5, nouvelle version majeure en Spring 6) • Moins utilisé (surtout par le fait qu'il faut des séniors pour l'équipe de développement) • Moins adapté pour les applications ne nécessitant pas un modèle réactif.	• GitHub • Doc de référence • Excellent comparatif et benchmark de technology.amis.nl • Tutoriels Spring WebFlux sur Baeldung

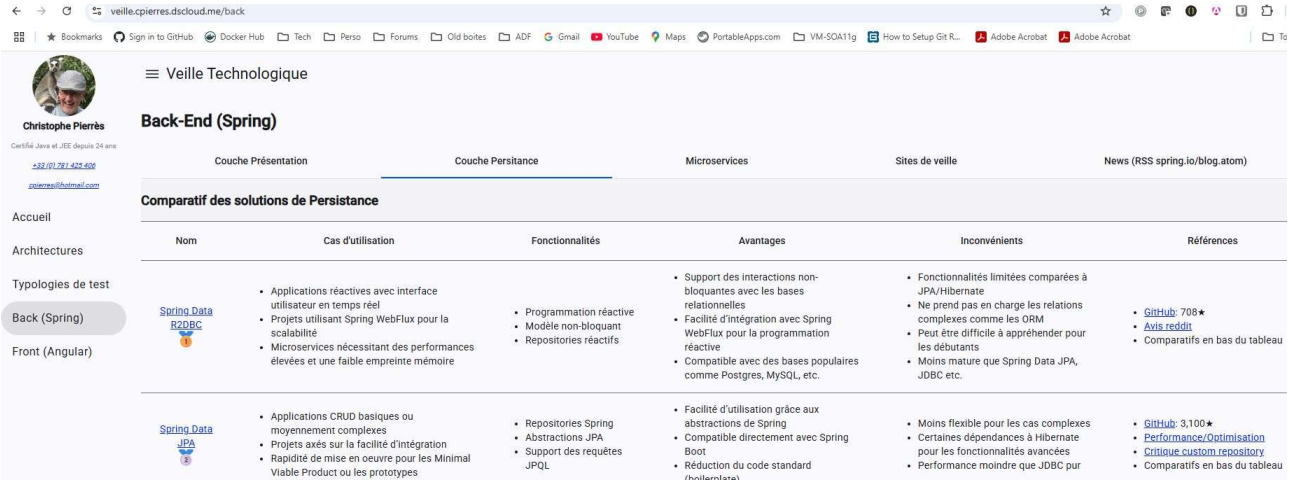
3.1.2.2. Module de persistance : Spring Data JPA (base sur Hibernate)

Spring Data JPA (intègre l'implémentation Hibernate)

- Justification : Simplification de l'accès aux données via JPA (norme JEE d'accès aux données).
- Avantages : Réduction du code boilerplate, requêtes dérivées des noms de méthodes, pagination intégrée.

Hibernate

- Justification : Implémentation JPA mature et performante.
- Avantages : Mapping objet-relationnel puissant, cache de second niveau, optimisations de performance.



3.1.2.3. Modules : Spring Security, OAuth2 Resource Server, Spring Security OAuth2 JOSE (JSON Object Signing and Encryption)

- Justification : Framework de sécurité complet pour les applications Java.
- Avantages : Authentification et autorisation robustes, protection contre les vulnérabilités courantes.

JWT (JSON Web Tokens)

- Justification : Standard pour l'authentification et l'échange d'informations sécurisées.
- Avantages : Sans état (stateless), facilement intégrable dans une architecture SPA/API.

BCrypt

- Justification : Algorithme de hachage sécurisé pour les mots de passe.
- Avantages : Résistance aux attaques par force brute, salage intégré.

Spring Security est le module principal pour la sécurité.

Le module **Oauth2 Resource Server** est une configuration permettant à une application de se comporter comme un serveur de ressources OAuth 2.0. Un serveur de ressources est responsable de protéger les ressources (données, API) et de vérifier que les requêtes entrantes contiennent des tokens valides avant d'autoriser l'accès.

Dans l'architecture OAuth 2.0 complète, on distingue généralement :

- Le serveur d'autorisation : responsable de l'authentification des utilisateurs et de l'émission des tokens
- Le serveur de ressources : responsable de la validation des tokens et de la protection des ressources

Dans le cas de LiVrai, l'application jouera les deux rôles à la fois, d'une manière autonome :

- Elle émet des tokens JWT (rôle de serveur d'autorisation) via les endpoints /api/auth/login et /api/auth/register (utilisation du **module JOSE**)
- Elle valide ces tokens pour protéger ses propres ressources (rôle de serveur de ressources – conféré grâce au module **Oauth2 Resource Server**). Un serveur de ressources est responsable de protéger les ressources (données, API) et de vérifier que les requêtes entrantes contiennent des tokens valides avant d'autoriser l'accès

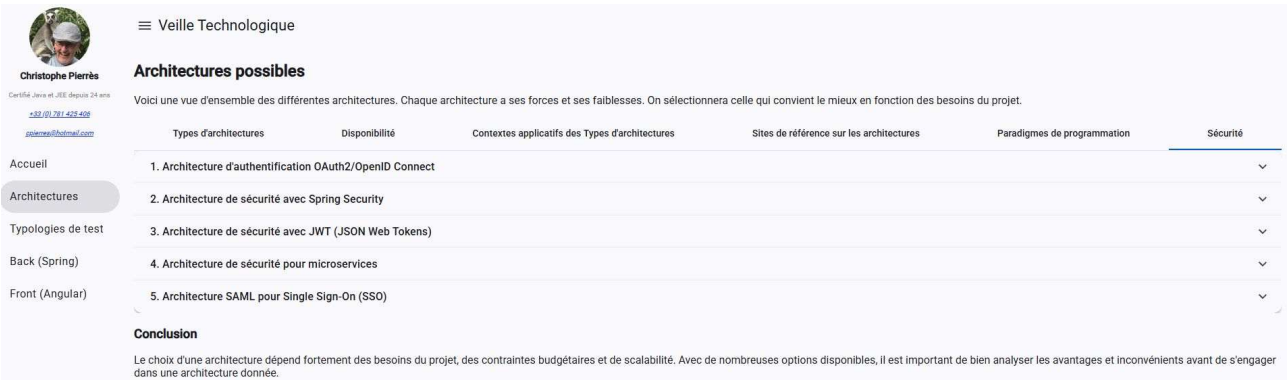
Cette approche "tout-en-un" est courante pour les applications qui n'ont pas besoin de partager leur système d'authentification avec d'autres applications. Elle simplifie l'architecture tout en bénéficiant des avantages de sécurité du protocole OAuth 2.0 et des JWT.

Nous utiliserons une clé symétrique avec encryptage HS256. Une clé asymétrique se justifierait dans le cas où il y aurait un partage de la sécurité entre plusieurs services distribués, ce qui n'est pas le cas pour MDD.

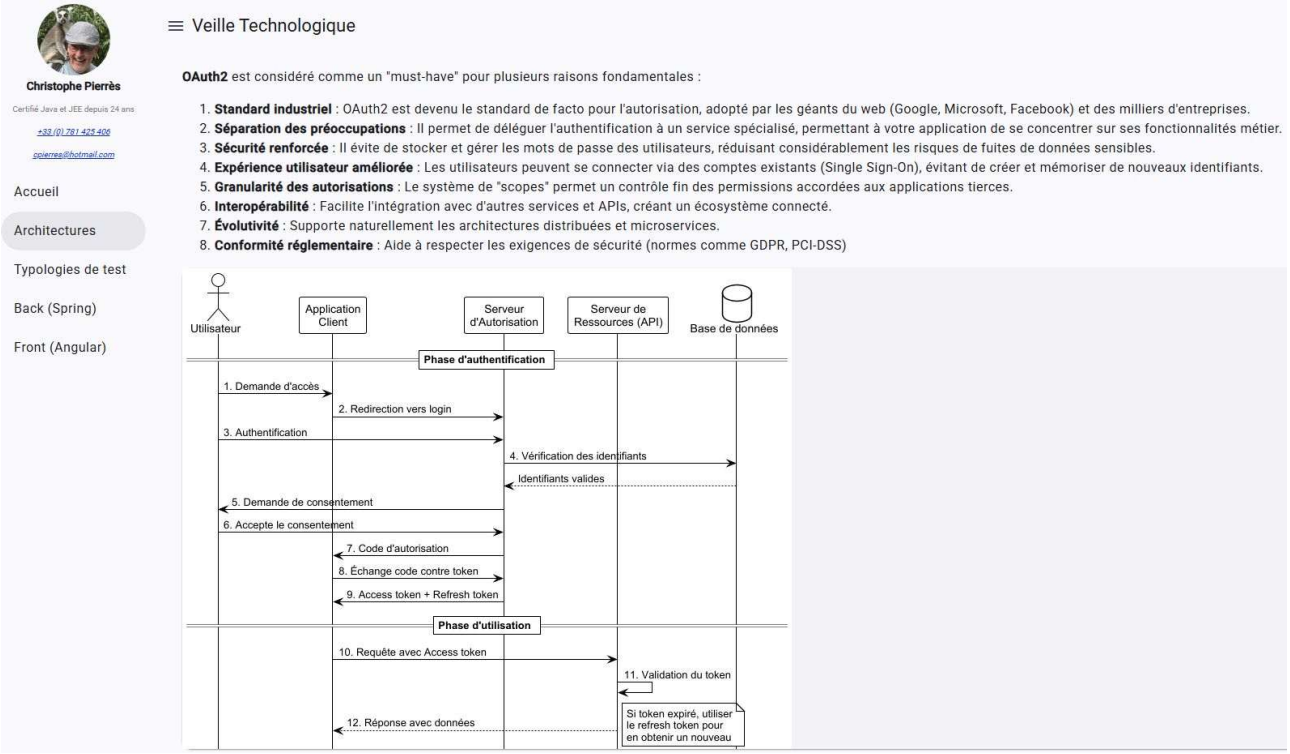
Informations complémentaires sur la sécurité

Cette section est une information complémentaire pour cadrer nos choix de modules de Spring Security.

Reportez-vous au menu Architectures puis à l'onglet Sécurité :



Dépliez "1. Architecture d'authentification OAuth2/OpenID Connect", qui est l'architecture de sécurité générale retenue de facto pour LiVrai :



Christophe Pierrès
Certifié Java et JEE depuis 24 ans
+33 (0) 781 425 406
cpierres@hotmail.com

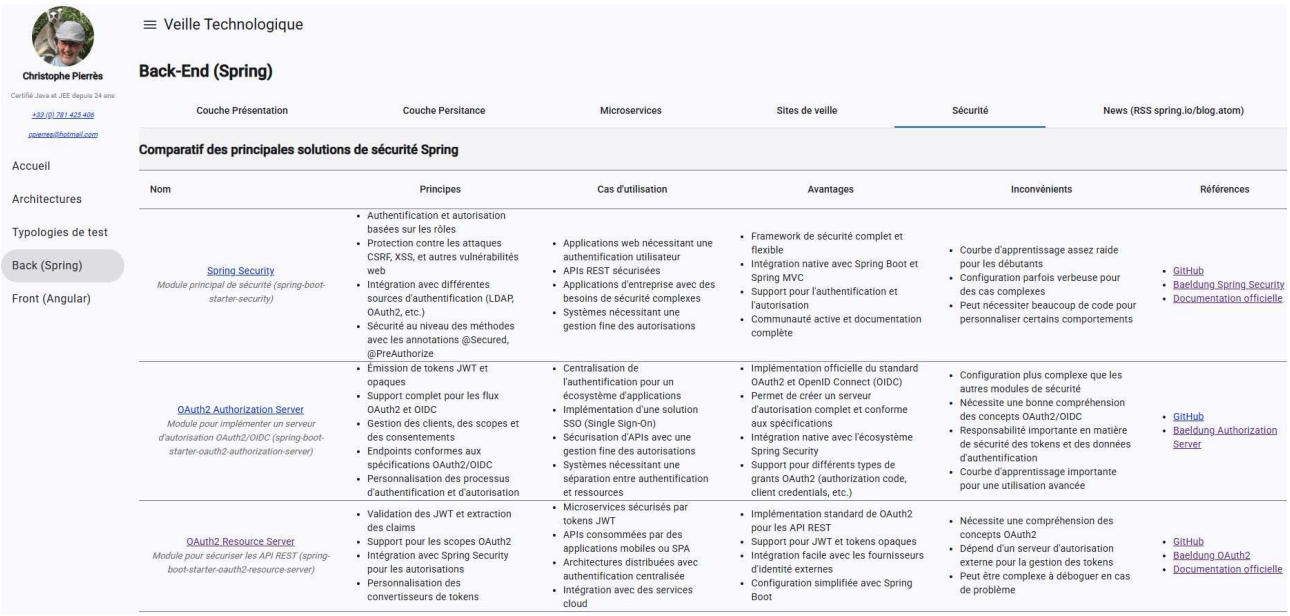
Accueil
Architectures
Typologies de test
Back (Spring)
Front (Angular)

≡ Veille Technologique

OAuth2 est considéré comme un "must-have" pour plusieurs raisons fondamentales :

1. **Standard industriel** : OAuth2 est devenu le standard de facto pour l'autorisation, adopté par les géants du web (Google, Microsoft, Facebook) et des milliers d'entreprises.
2. **Séparation des préoccupations** : Il permet de déléguer l'authentification à un service spécialisé, permettant à votre application de se concentrer sur ses fonctionnalités métier.
3. **Sécurité renforcée** : Il évite de stocker et gérer les mots de passe des utilisateurs, réduisant considérablement les risques de fuites de données sensibles.
4. **Expérience utilisateur améliorée** : Les utilisateurs peuvent se connecter via des comptes existants (Single Sign-On), évitant de créer et mémoriser de nouveaux identifiants.
5. **Granularité des autorisations** : Le système de "scopes" permet un contrôle fin des permissions accordées aux applications tierces.
6. **Interopérabilité** : Facilite l'intégration avec d'autres services et APIs, créant un écosystème connecté.
7. **Évolutivité** : Supporte naturellement les architectures distribuées et microservices.
8. **Conformité réglementaire** : Aide à respecter les exigences de sécurité (normes comme GDPR, PCI-DSS)

Maintenant, accédez au menu Back (Spring) puis Sécurité. Voici les modules de sécurité « classiques » :



Christophe Pierrès
Certifié Java et JEE depuis 24 ans
+33 (0) 781 425 406
cpierres@hotmail.com

Accueil
Architectures
Typologies de test
Back (Spring)
Front (Angular)

≡ Veille Technologique

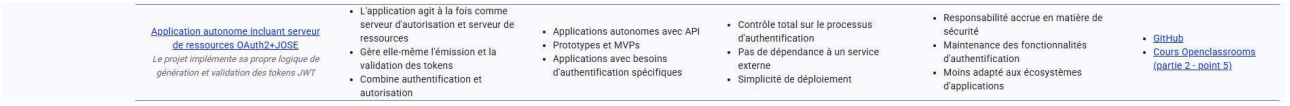
Back-End (Spring)

Couche Présentation | Couche Persitance | Microservices | Sites de veille | Sécurité | News (RSS spring.io/blog.atom)

Comparatif des principales solutions de sécurité Spring

Nom	Principes	Cas d'utilisation	Avantages	Inconvénients	Références
Spring Security *Module principal de sécurité (spring-boot-starter-security)*	• Authentification et autorisation basées sur les rôles • Protection contre les attaques CSRF, XSS, et autres vulnérabilités web • Intégration avec différentes sources d'authentification (LDAP, OAuth2, etc.) • Sécurité au niveau des méthodes avec les annotations @Secured, @PreAuthorize	• Applications web nécessitant une authentification utilisateur • APIs REST sécurisées • Applications d'entreprise avec des besoins de sécurité complexes • Systèmes nécessitant une gestion fine des autorisations	• Framework de sécurité complet et flexible • Intégration native avec Spring Boot et Spring MVC • Support pour l'authentification et l'autorisation • Communauté active et documentation complète	• Courbe d'apprentissage assez raide pour les débutants • Configuration parfois verbeuse pour des cas complexes • Peut nécessiter beaucoup de code pour personnaliser certains comportements	• GitHub • Baeldung Spring Security • Documentation officielle
OAuth2 Authorization Server *Module pour implémenter un serveur d'autorisation OAuth2/OIDC (spring-boot-starter-oauth2-authorization-server)*	• Émission de tokens JWT et opaques • Support complet pour les flux OAuth2 et OIDC • Gestion des clients, des scopes et des consentements • Endpoints conformes aux spécifications OAuth2/OIDC • Personnalisation des processus d'authentification et d'autorisation	• Centralisation de l'authentification pour un écosystème d'applications • Implémentation d'une solution SSO (Single Sign-On) • Sécurisation d'APIs avec une gestion fine des autorisations • Systèmes nécessitant une séparation entre authentification et ressources	• Implémentation officielle du standard OAuth2 et OpenID Connect (OIDC) • Permet de créer un serveur d'autorisation complet et conforme aux spécifications • Intégration native avec l'écosystème Spring Security • Support pour différents types de grants OAuth2 (authorization code, client credentials, etc.)	• Configuration plus complexe que les autres modules de sécurité • Nécessite une bonne compréhension des concepts OAuth2/OIDC • Responsabilité importante en matière de sécurité des tokens et des données d'authentification • Courbe d'apprentissage importante pour une utilisation avancée	• GitHub • Baeldung Authorization Server
OAuth2 Resource Server *Module pour sécuriser les API REST (spring-boot-starter-oauth2-resource-server)*	• Validation des JWT et extraction des claims • Support pour les scopes OAuth2 • Intégration avec Spring Security pour les autorisations • Personnalisation des convertisseurs de tokens	• Microservices sécurisés par tokens JWT • APIs consommées par des applications mobiles ou SPA • Architectures distribuées avec authentification centralisée • Intégration avec des services cloud	• Implémentation standard de OAuth2 pour les API REST • Support pour JWT et tokens opaques • Intégration facile avec les fournisseurs d'identité externes • Configuration simplifiée avec Spring Boot	• Nécessite une compréhension des concepts OAuth2 • Dépend d'un serveur d'autorisation externe pour la gestion des tokens • Peut être complexe à déboguer en cas de problème	• GitHub • Baeldung OAuth2 • Documentation officielle

Nous avons choisi d'intégré les 3 modules ci-dessus pour aboutir à la configuration décrite ci-dessous :



Application autonome incluant serveur de ressources OAuth2+JOSE *Le projet implémente sa propre logique de génération et validation des tokens JWT*	• L'application agit à la fois comme serveur d'autorisation et serveur de ressources • Gère elle-même l'émission et la validation des tokens • Combine authentification et autorisation	• Applications autonomes avec API • Prototypes et MVPs • Applications avec besoins d'authentification spécifiques	• Contrôle total sur le processus d'authentification • Pas de dépendance à un service externe • Simplicité de déploiement	• Responsabilité accrue en matière de sécurité • Maintenance des fonctionnalités d'authentification • Moins adapté aux écosystèmes d'applications	• GitHub • Cours Openclassrooms (partie 2 - point 5)

3.1.2.4. Java 17 (LTS)

- Justification : Version LTS récente de Java, offrant de nombreuses améliorations de performance et les nouvelles fonctionnalités suffisantes pour notre version de Spring.
- Avantages : Support à long terme, performances améliorées, nouvelles fonctionnalités.

3.1.3. Frontend : Angular v19.2.5

Justification : Framework mature et robuste pour le développement de SPA, offrant une architecture modulaire, un système de composants réutilisables, et un écosystème riche.

Avantages : Performances optimisées, binding bidirectionnel, routing intégré, support TypeScript, large communauté.

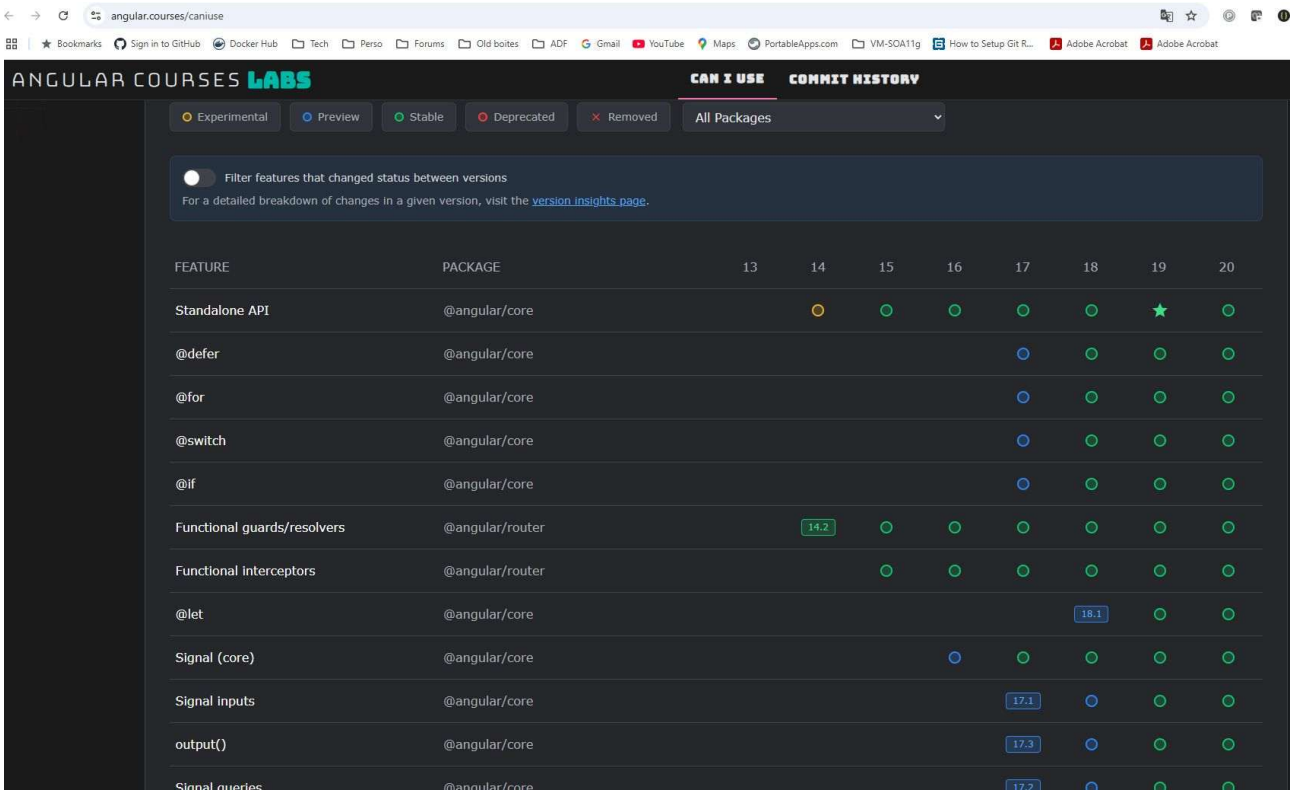
Nous choisissons cette version récente d'Angular car les technologies avancées sur lesquelles elle se base ont été introduites depuis la version 14 ou 16 et on peut les considérer aujourd'hui comme matures.

Il est bien temps d'utiliser tous les concepts modernes d'Angular qui apportent : performance, meilleure maintenabilité (standalone components) et plus de simplicité.

Citons l'API Signal : Lorsqu'une valeur contenue dans un Signal est modifiée, Angular réagit automatiquement aux mises à jour de manière optimisée, sans nécessiter un cycle complet de détection des changements.

Les performances du **runtime Angular** et du nouveau **compilateur Ivy** permettent une amélioration significative des temps de chargement des applications.

Le très utile tableau « can I use ? » https://www.angular.courses/caniuse permet de se rendre compte que nombre de fonctionnalités sont arrivées à maturité dans la version 19 :



Mais plus important encore, certaines fonctionnalités très courantes comme les *ngIf, *ngFor et *ngSwitch vont devenir deprecatead en version 20 (et pire encore, cela n'apparait pas encore dans ce tableau, mais elles sont destinées à être supprimées dans la version 22 !). Par conséquent, il est grand temps d'utiliser les nouvelles syntaxes …

FEATURE	PACKAGE	13	14	15	16	17	18	19	20
rxResource()	@angular/core								
httpResource()	@angular/core							19.2	
linkedSignal	@angular/core								
withRequestsMadeViaParent	@angular/common								
ngOptimizedImage	@angular/common		14.2						
provideClientHydration	@angular/platform-browser								
*ngIf	@angular/common								
*ngFor	@angular/common								
*ngSwitch	@angular/common								
HammerJS integration	@angular/platform-browser								
platformServerTesting	@angular/platform-server/testing								
ServerTestingModule	@angular/platform-server/testing								
TestBed.get	@angular/core/testing								
TestBed.flushEffects	@angular/core/testing								

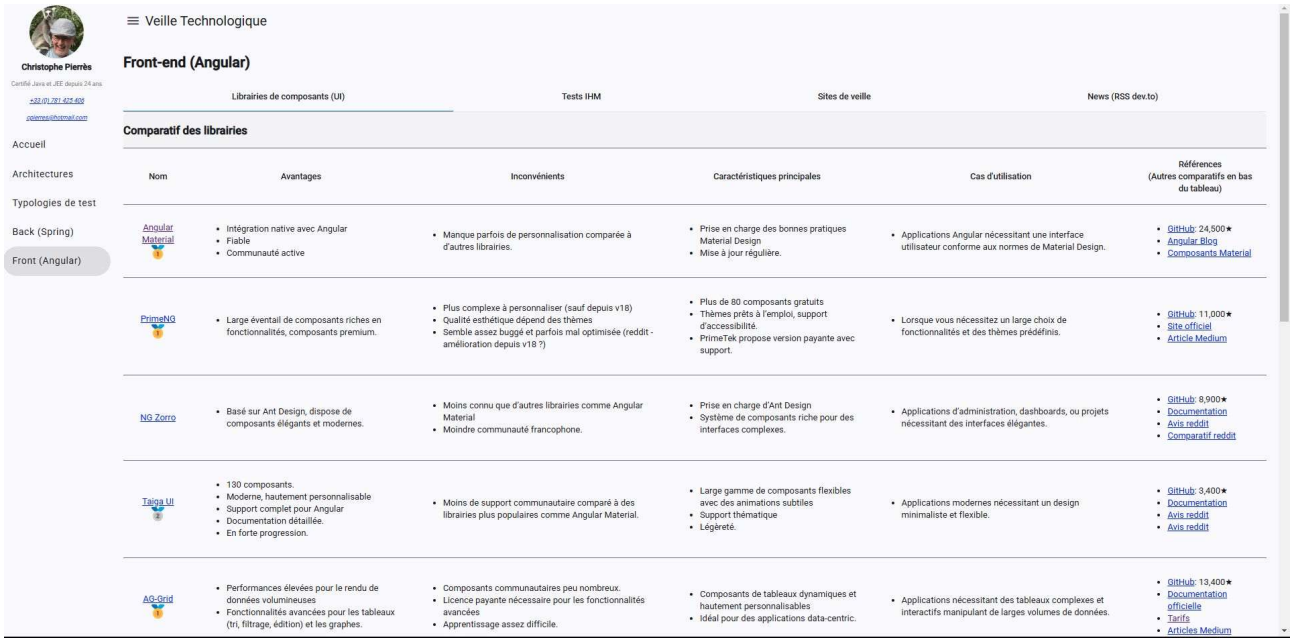
3.1.3.1. Librairie de composants visuels : Angular Materials (+ éventuel complément)

- Justification : Bibliothèque de composants UI respectant les principes du Material Design.
- Avantages : Composants prêts à l'emploi, cohérence visuelle, accessibilité intégrée.

Une analyse des librairies de composants est disponible sur : https://veille.cpierres.dscloud.me menu Front (Angular).

La solution incluse dans Angular est : **Angular Materials** 🥇. Nous la préconisons car robuste et intéressante.

Les composants de cette librairie ne couvrent pas toujours tous les besoins (ex : diagrammes de Gantt). Pour compléter les manques éventuels, nous proposons **PrimeNG** 🥇. Cette solution est la plus populaire d'après l'historique des stars. Néanmoins, ce tableau pourrait permettre d'affiner ce choix car si vous avez des demandes spécifiques (grosse volumétrie sur les tableaux ou composants spécifiques par exemple), d'autres solutions peuvent être décidées entre **AG-Grid** (payante) et la prometteuse librairie **Taigai UI** :



Christophe Pierrès
Certifié Java et JEE depuis 24 ans
+33 (0) 781 425 406
cpierres@hotmail.com

Accueil
Architectures
Typologies de test
Back (Spring)
Front (Angular)

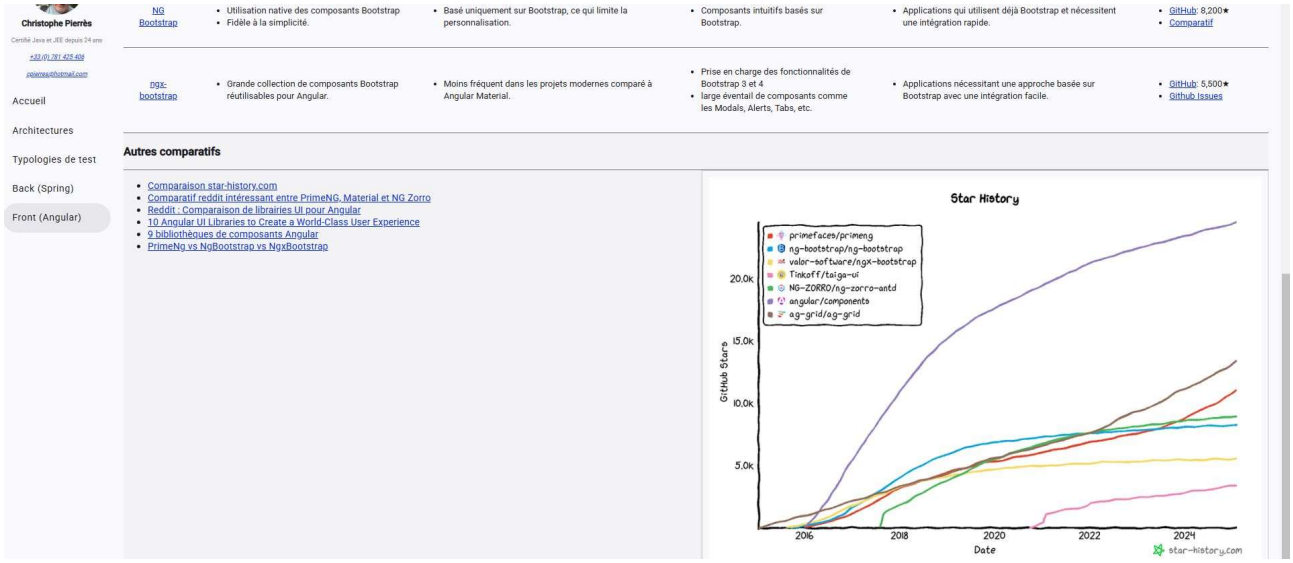
Veille Technologique

Front-end (Angular)

Librairies de composants (UI) | Tests IHM | Sites de veille | News (RSS dev.to)

Comparatif des librairies

Nom	Avantages	Inconvénients	Caractéristiques principales	Cas d'utilisation	Références (Autres comparatifs en bas du tableau)
Angular Material	• Intégration native avec Angular • Fiable • Communauté active	• Manque parfois de personnalisation comparée à d'autres librairies.	• Prise en charge des bonnes pratiques Material Design • Mise à jour régulière.	• Applications Angular nécessitant une interface utilisateur conforme aux normes de Material Design.	• GitHub: 24,500★ • Angular Blog • Composants Material
PrimeNG	• Large éventail de composants riches en fonctionnalités, composants premium.	• Plus complexe à personnaliser (sauf depuis v18) • Qualité esthétique dépend des thèmes • Semble assez buggé et parfois mal optimisée (reddit - amélioration depuis v18 ?)	• Plus de 80 composants gratuits • Thèmes prêts à l'emploi, support d'accessibilité. • PrimeTek propose version payante avec support.	• Lorsque vous nécessitez un large choix de fonctionnalités et des thèmes prédéfinis.	• GitHub: 11,000★ • Site officiel • Article Medium
NG Zorro	• Basé sur Ant Design, dispose de composants élégants et modernes.	• Moins connu que d'autres librairies comme Angular Material • Moindre communauté francophone.	• Prise en charge d'Ant Design • Système de composants riche pour des interfaces complexes.	• Applications d'administration, dashboards, ou projets nécessitant des interfaces élégantes.	• GitHub: 8,900★ • Documentation • Avis reddit • Comparatif reddit
Taiga UI	• 130 composants. • Moderne, hautement personnalisable • Support complet pour Angular • Documentation détaillée. • En forte progression.	• Moins de support communautaire comparé à des librairies plus populaires comme Angular Material.	• Large gamme de composants flexibles avec des animations subtiles • Support thématique • Légèreté.	• Applications modernes nécessitant un design minimaliste et flexible.	• GitHub: 3,400★ • Documentation • Avis reddit • Avis reddit
AG-Grid	• Performances élevées pour le rendu de données volumineuses • Fonctionnalités avancées pour les tableaux (tri, filtrage, édition) et les graphes.	• Composants communautaires peu nombreux. • Licence payante nécessaire pour les fonctionnalités avancées • Apprentissage assez difficile.	• Composants de tableaux dynamiques et hautement personnalisables • Idéal pour des applications data-centric.	• Applications nécessitant des tableaux complexes et interactifs manipulant de larges volumes de données.	• GitHub: 13,400★ • Documentation officielle • Tarifs • Articles Medium



Christophe Pierrès
Certifié Java et JEE depuis 24 ans
+33 (0) 781 425 406
cpierres@hotmail.com

Accueil
Architectures
Typologies de test
Back (Spring)
Front (Angular)

NG Bootstrap	• Utilisation native des composants Bootstrap • Fidèle à la simplicité.	• Basé uniquement sur Bootstrap, ce qui limite la personnalisation.	• Composants intuitifs basés sur Bootstrap.	• Applications qui utilisent déjà Bootstrap et nécessitent une intégration rapide.	• GitHub: 8,200★ • Comparatif
ngx-bootstrap	• Grande collection de composants Bootstrap réutilisables pour Angular.	• Moins fréquent dans les projets modernes comparé à Angular Material.	• Prise en charge des fonctionnalités de Bootstrap 3 et 4 • large éventail de composants comme les Modals, Alerts, Tabs, etc.	• Applications nécessitant une approche basée sur Bootstrap avec une intégration facile.	• GitHub: 5,500★ • Github Issues

Autres comparatifs

- Comparaison star-history.com
- Comparatif reddit intéressant entre PrimeNG, Material et NG Zorro
- Reddit : Comparaison de librairies UI pour Angular
- 10 Angular UI Libraries to Create a World-Class User Experience
- 9 bibliothèques de composants Angular
- PrimeNg vs NgBootstrap vs NgxBootstrap

3.1.3.2. TypeScript

- Justification : Langage typé qui améliore la maintenabilité et la robustesse du code JavaScript.
- Avantages : Détection d'erreurs à la compilation, meilleur support IDE, documentation intégrée via les types.

3.1.3.3. RxJs (en complément de l'API Signal incluse dans Angular)

- Justification : Bibliothèque pour la programmation réactive.
- Avantages : Gestion avancée des flux de données asynchrones, facilite la gestion des événements et des requêtes HTTP.

3.1.4. Déploiement

Docker

- Justification : Conteneurisation pour faciliter le déploiement et la scalabilité.
- Avantages : Environnements cohérents, isolation, facilité de déploiement.

Docker Compose

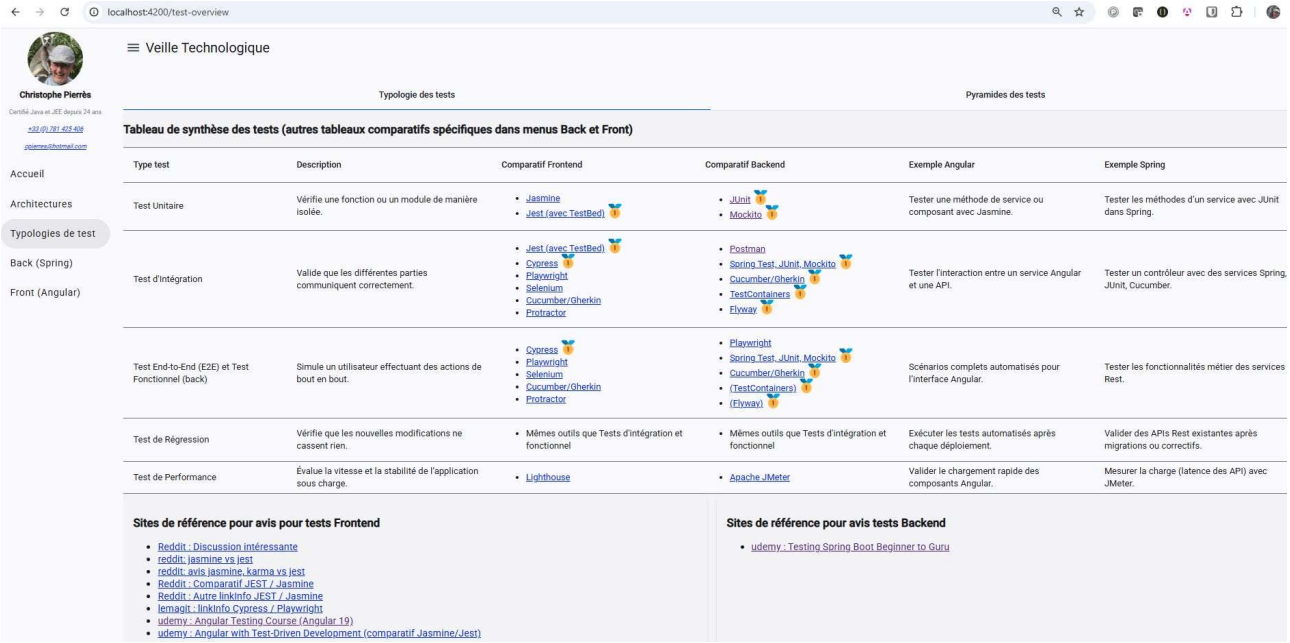
- Justification : Orchestration simple de conteneurs multiples.
- Avantages : Configuration déclarative, gestion des dépendances entre services.

3.1.5. Testing

Cette problématique ne fait pas partie du scope du MVP. Cependant, ces choix devront arriver vite dès la confirmation du lancement du projet. Par conséquent, voici nos préconisations.

Ce tableau présente une synthèse des choix de solutions de testing pour le front et le back :

https://veille.cpierres.dscloud.me menu Typologies de test



Les deux sous-sections suivantes présentent les justifications de ces choix.

3.1.5.1. Front-end testing

Pour une analyse détaillée des choix de librairies, consultez le lien : https://veille.cpierres.dscloud.me menu **Front (Angular)**, puis onglet **Tests IHM** :

Christophe Pierrès

Certifié Java et JEE depuis 24 ans

+33 (0) 781 424 406

spierres@hotmail.com

Accueil

Architectures

Typologies de test

Back (Spring)

Front (Angular)

localhost:4200/front

≡ Veille Technologique

Front-end (Angular)

Librairies de composants (UI) | Tests IHM | Sites de veille | News (RSS dev.to)

Comparatif des librairies de test

Nom	Type de Test	Caractéristiques	Points forts	Points faibles	Références (comparatifs bas du tableau)
Jasmine	• Unit Testing	• Framework de test unitaire populaire • Écrit en JavaScript • Approche BDD (Behavior Driven Development)	• Intégré par défaut dans Angular • Syntaxe simple et facile à comprendre • Bien adapté aux tests frontend • Écosystème mature et bien documenté	• Pas conçu pour exécuter les tests dans les navigateurs ou directement avec Node.js • Pas conçu pour gérer les rythmes de test asynchrone avancés	• GitHub: 15,800★ • StackShare • Comparatifs en bas du tableau
Karma	• Test Runner	• Permet l'exécution des tests dans des navigateurs réels • Compatible avec plusieurs frameworks (Jasmine, Mocha, etc.) • Intégration avec CI/CD (Continuous Integration)	• Intégré par défaut dans Angular • S'exécute sur plusieurs navigateurs • Prend en charge de nombreux frameworks de test	• Configuration complexe pour certains projets • Performance plus lente par rapport aux solutions modernes comme Jest	• GitHub: 12,000★ • StackShare • Reddit : Avis et linkinfo • Comparatifs en bas du tableau
TestBed	• Unit Testing (Angular Test Framework)	• Solution native pour le test unitaire dans Angular • Permet de tester les composants, modules et services Angular • Par défaut a besoin de Jasmine et Karma pour l'exécution • Ou bien de Jest	• Solution introduite et maintenue directement par Angular • Excellente intégration avec Angular • Prend en charge les tests unitaires et semi-intégrés	• Dépend de Karma pour l'exécution, ce qui peut être lent (mais Jest peut remplacer Karma) • Limité aux tests d'unités ou composés liés à Angular	• GitHub: 97,000★ • Github issues • StackShare • Testing Angular Guide • Tutoriel intéressant
Jest	• Unit Testing • Integration Testing	• Framework tout-en-un : assertion, mocking, backend intégrés • Compatible avec JavaScript, TypeScript, Angular, React, Vue.js • Approche moderne avec tests parallèles automatiques	• Exécution des tests rapide grâce au parallélisme • Le meilleur choix pour les projets modernes utilisant Angular ou React • Excellent support des tests d'instantanés (snapshot testing)	• Léger apprentissage requis pour les développeurs habitués à Jasmine/Karma • Plus difficile à configurer pour les environnements multi-navigateurs	• GitHub: 44,500★ • StackShare • Comparatifs en bas du tableau
Protractor	• End-to-end (E2E) Testing	• Framework de test automatisé pour les applications Angular • Support natif des synchronisations Angular • Compatible avec Selenium et WebDriver	• Conçu spécifiquement pour Angular • Large support grâce aux API Selenium • Bonne documentation officielle	• Déprécié depuis 2020 (recherche d'alternatives nécessaire) • Capacité limitée pour les applications non-Angular	• GitHub: 8,700★ • StackShare
Cypress	• E2E Testing	• Tests end-to-end interactifs pour les applications web modernes • Rapports intégrés et faciles à configurer • Moins dépendant d'environnements externes par rapport à d'autres outils	• Excellente expérience utilisateur pour les tests frontend • Facile à intégrer avec CI/CD • Documentation bien écrite et claire	• Principalement conçu pour tester les navigateurs (pas d'autre type d'applications)	• GitHub: 46,000★ • StackShare • Avis sur Reddit (vs playwright) • Autre avis sur Reddit • OpenClassrooms : Cypress
Playwright	• E2E Testing	• Framework moderne pour le test E2E • Compatible avec les navigateurs modernes (Chromium, Firefox, WebKit) • Supporte les tests via plusieurs langages (JavaScript, TypeScript, Python, Java, C#)	• Performances rapides grâce à des sessions de navigateur headless • Fonctionnalités avancées comme les tests parallèles, les fixtures et les traces interactives • Documentation riche et exemples faciles à suivre • Automatisation multiplateforme (Windows, macOS, Linux)	• Pas conçu pour les tests unitaires • Configuration initiale plus complexe pour les débutants • Maîtrise requise des promesses pour gérer les tests asynchrones	• GitHub: 69,200★ • StackShare • Reddit : avis Playwright (vs cypress et selenium)

Autres comparatifs et sites de référence

- Reddit : Discussion intéressante
- reddit: jasmine vs jest
- reddit: avis jasmine, karma vs jest
- Reddit : Comparatif JEST / Jasmine
- Reddit : Autre linkinfo JEST / Jasmine
- lemagit : linkinfo Cypress / Playwright
- udemy : Angular Testing Course (Angular 19)
- udemy : Angular with Test-Driven Development (comparatif Jasmine/Jest)

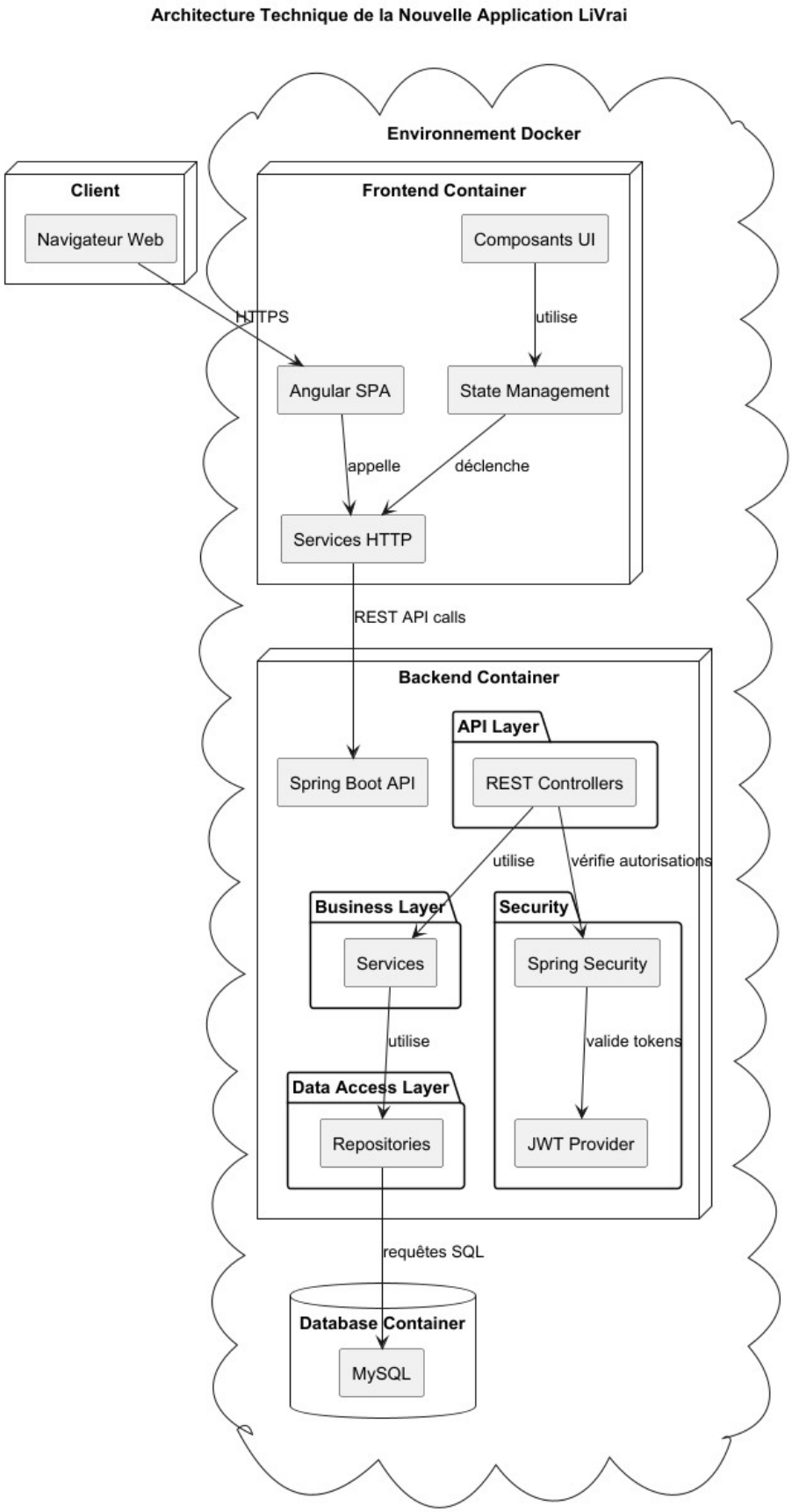
Angular intègre par défaut Jasmine, Karma et TestBed. Cependant, notre veille technologique montre que beaucoup se sont tournés vers **Jest** et **Cypress,** que nous retenons. Dans un post récent sur reddit, on apprend que l'équipe d'Angular elle-même envisage de changer leurs propositions par défaut (éventuellement vers Jest), ce qui confirme notre proposition. Par ailleurs, **TestBed** fonctionne parfaitement avec Jest.

3.1.5.2. Back-end testing

Pour Spring, l'intégration des capacités de test est particulièrement bien soignée et les propositions par défaut sont excellentes. Spring permet d'intégrer avec facilité des briques optionnelles que nous avons retenues (TestContainers). Le tableau vu précédemment au début de cette section confirme les choix suivants :

SpringTest (Junit, **Mockito)**, **Cucumber/Gherkin** (expression des tests en langage naturel permettant aux fonctionnels de mieux comprendre le résultat des tests d'intégration ou fonctionnels), **TestContainers** et **Flyway** pour la mise en œuvre de containers réalistes si besoin pour des jeux de tests plus sophistiqués/poussés que de simples mock.

3.2. Architecture de l'application



3.2.1. Architecture détaillée

3.2.1.1. Architecture Frontend (Angular)

- Modules : Core, Shared, Features (User, Delivery, Billing)
- Services : Authentication, User, Delivery, Billing, Notification
- State Management : NgRx pour la gestion d'état
- Routing : Angular Router avec lazy loading

3.2.1.2. Architecture Backend (Spring Boot)

- API RESTful : Endpoints pour toutes les fonctionnalités
- Sécurité : Authentification JWT, autorisation basée sur les rôles
- Validation : Validation des entrées avec Bean Validation
- Exception Handling : Gestion centralisée des exceptions
- Logging : Journalisation structurée avec SLF4J et Logback
- Tests : Tests unitaires et d'intégration avec JUnit 5 et Mockito

3.2.1.3. Modèle de données

- Utilisateurs : Différents types (Admin, Commercial, Service Livraisons, Client)
- Livraisons : Cycle de vie complet avec états et transitions
- Facturation : Génération et suivi des factures
- Audit : Journalisation des actions importantes

3.2.1.4. Communication Frontend-Backend

- API RESTful : Communication JSON via HTTPS
- Authentification : Tokens JWT
- Validation : Validation côté client et serveur
- Gestion d'erreurs : Codes HTTP appropriés et messages d'erreur clairs

3.2.1.5. Déploiement

- Conteneurisation : Docker pour tous les composants
- Orchestration : Docker Compose pour le développement, potentiellement Kubernetes pour la production
- CI/CD : Pipeline d'intégration et déploiement continus

Possibilité de disponibilité horizontale grâce à Kubernetes

Pour plus d'informations, rendez-vous sur https://veille.cpierres.dscloud.me/architecture onglet Disponibilité :

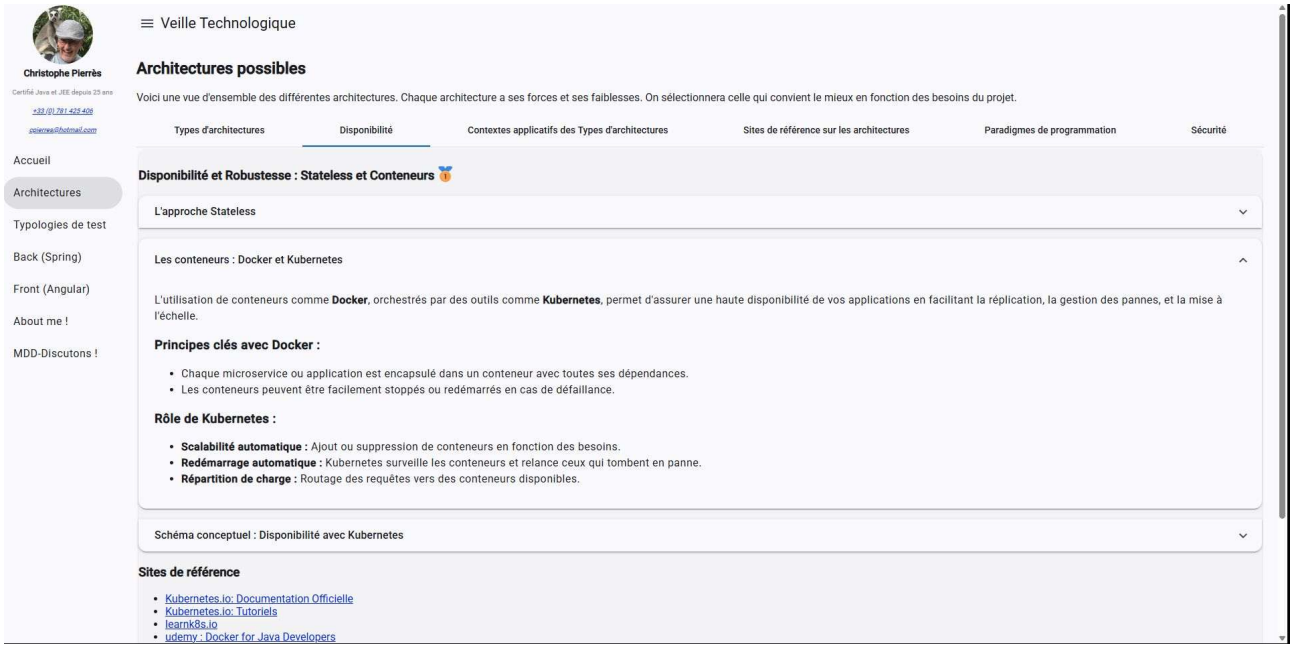
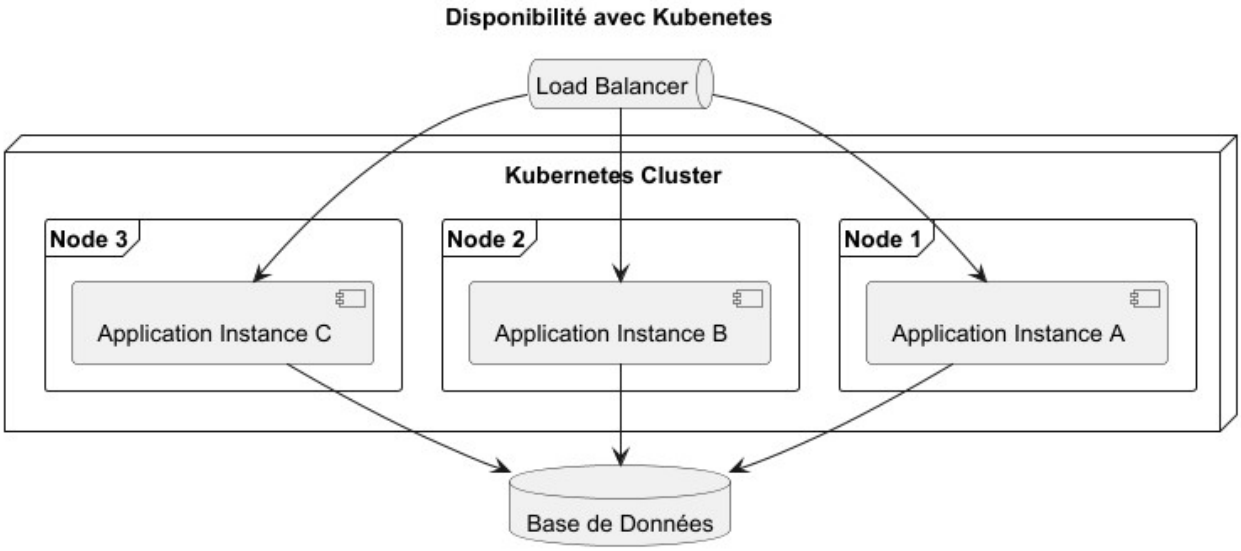
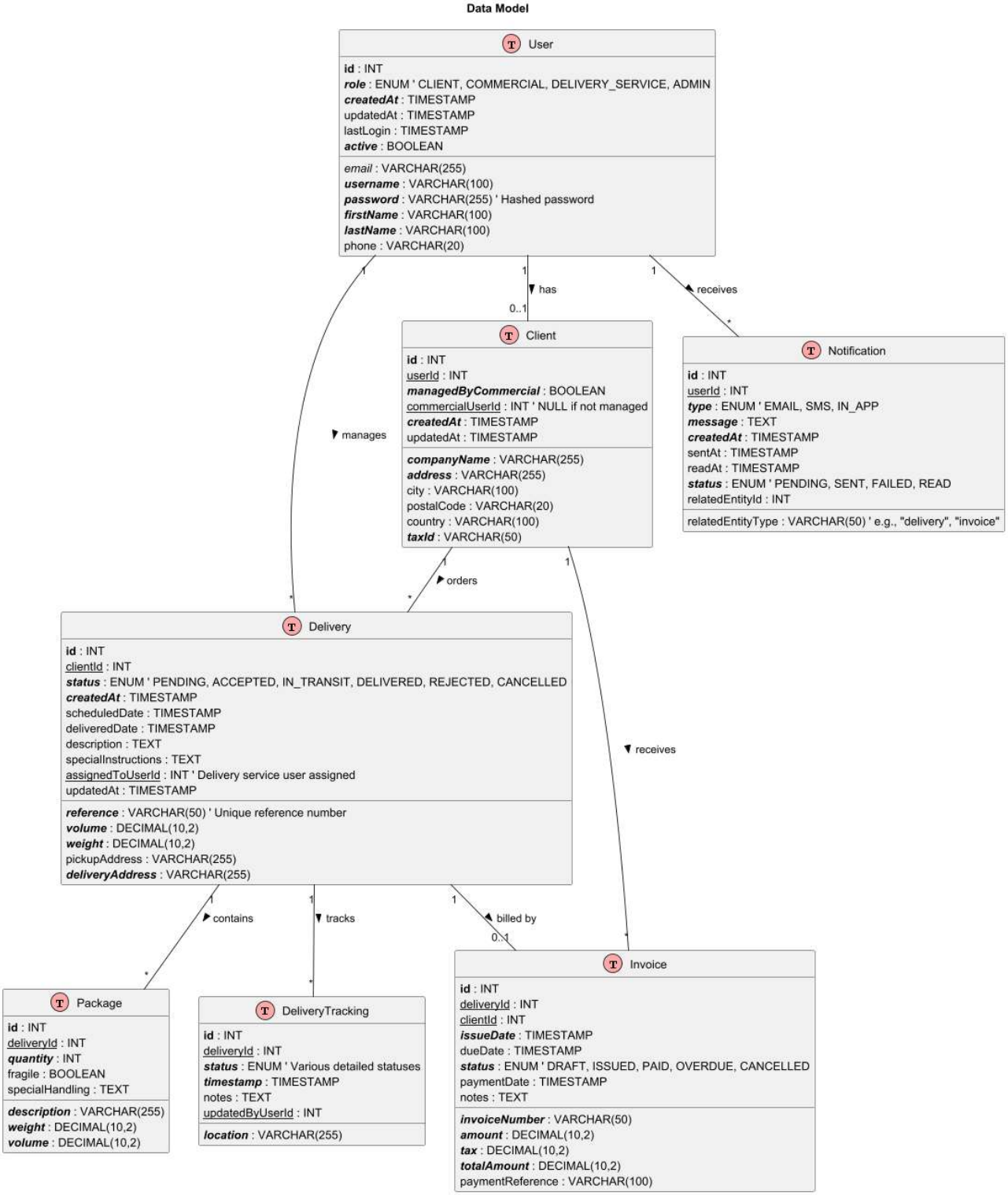


Schéma conceptuel : Disponibilité avec Kubernetes



3.2.2. Modèle de données