

## Chapitre 1

### Étude de l'existant

Afin de mieux comprendre le besoin réel de la future application, il est essentiel d'analyser les solutions existantes sur le marché. Cette étude permet d'identifier les insuffisances des services actuels et de proposer des améliorations adaptées aux développeurs.

### Analyse de l'existant

Aujourd'hui, plusieurs services de notification existent et sont largement utilisés par les entreprises et les développeurs pour envoyer des emails, des SMS et des notifications push. Ces services incluent notamment :

- **Firestore Cloud Messaging (FCM)** : utilisé pour envoyer des notifications push sur les applications mobiles et les navigateurs web.
- **Twilio** : permet l'envoi de SMS, d'appels vocaux et de messages via WhatsApp.
- **AWS Simple Email Service (SES)** : service d'envoi d'emails transactionnels et marketing basé sur le cloud.
- **OneSignal** : plateforme de notifications push pour le web, iOS et Android.
- **SendGrid** : solution d'emailing populaire qui offre des API performantes pour l'envoi d'emails.

Ces plateformes permettent aux entreprises d'intégrer différents moyens de communication dans leurs applications, mais elles présentent également plusieurs limitations.

### Critique de l'existant

En analysant ces solutions, nous avons constaté plusieurs limites qui compliquent la gestion centralisée des notifications pour les développeurs :

### Multiplication des intégrations

Chaque service propose ses propres APIs, ce qui signifie que les développeurs doivent intégrer plusieurs services distincts s'ils souhaitent envoyer des notifications sur différents canaux (email, SMS, push). Cela entraîne :

- Une **complexité accrue** dans le développement et la maintenance.
- Des **coûts supplémentaires** liés à l'utilisation de plusieurs services tiers.

## Manque de flexibilité

Certaines plateformes, comme Firebase Cloud Messaging, ne prennent en charge que les notifications push, tandis que Twilio est spécialisé dans les SMS et les appels. Il n'existe pas de solution unique permettant d'envoyer des notifications sur plusieurs canaux de manière centralisée.

## Coût élevé et limitations

- La plupart des services facturent en fonction du volume de notifications envoyées, ce qui peut rapidement devenir coûteux.
- Certains services imposent des **quotas d'utilisation**, rendant l'évolutivité difficile pour les entreprises ayant un grand volume de notifications.
- Les coûts varient également selon les régions et les fournisseurs de télécommunications (notamment pour les SMS).

### Absence d'une gestion unifiée

- Les entreprises doivent développer leurs propres **mécanismes de routage et de priorisation** pour choisir le canal le plus adapté en fonction du type de notification et du contexte utilisateur.
- Il n'existe pas d'interface unique permettant de suivre et d'analyser l'ensemble des notifications envoyées sur différents canaux.

## Méthodologie Adoptée

Dans cette partie, nous avons adopté une **méthodologie Agile**, et plus précisément la **méthodologie Scrum**.

L'Agilité est une approche itérative et collaborative qui permet de prendre en compte **les besoins initiaux du client** ainsi que ceux liés aux évolutions du projet. Elle repose sur un **cycle de développement centré sur le client**, l'impliquant dans la réalisation du projet du début à la fin.

Les méthodologies agiles offrent une meilleure **visibilité** et une plus grande **flexibilité** par rapport aux méthodes classiques, permettant ainsi une gestion plus efficace du projet.

## SCRUM

Pour le développement de notre solution "**Unified Notification Service for Developers**", nous avons utilisé la méthodologie **Scrum**. C'est un **Framework Agile dédié à la gestion de projet**, particulièrement adapté aux projets informatiques et au génie logiciel.

Scrum repose sur trois **principes fondamentaux** :

- **La transparence** : Tous les membres de l'équipe doivent avoir connaissance des informations relatives au produit en développement.
- **L'inspection** : Des évaluations régulières permettent de s'assurer que le projet suit la bonne direction.
- **L'adaptation** : Si des écarts sont détectés, des ajustements sont réalisés pour s'adapter aux nouvelles contraintes.

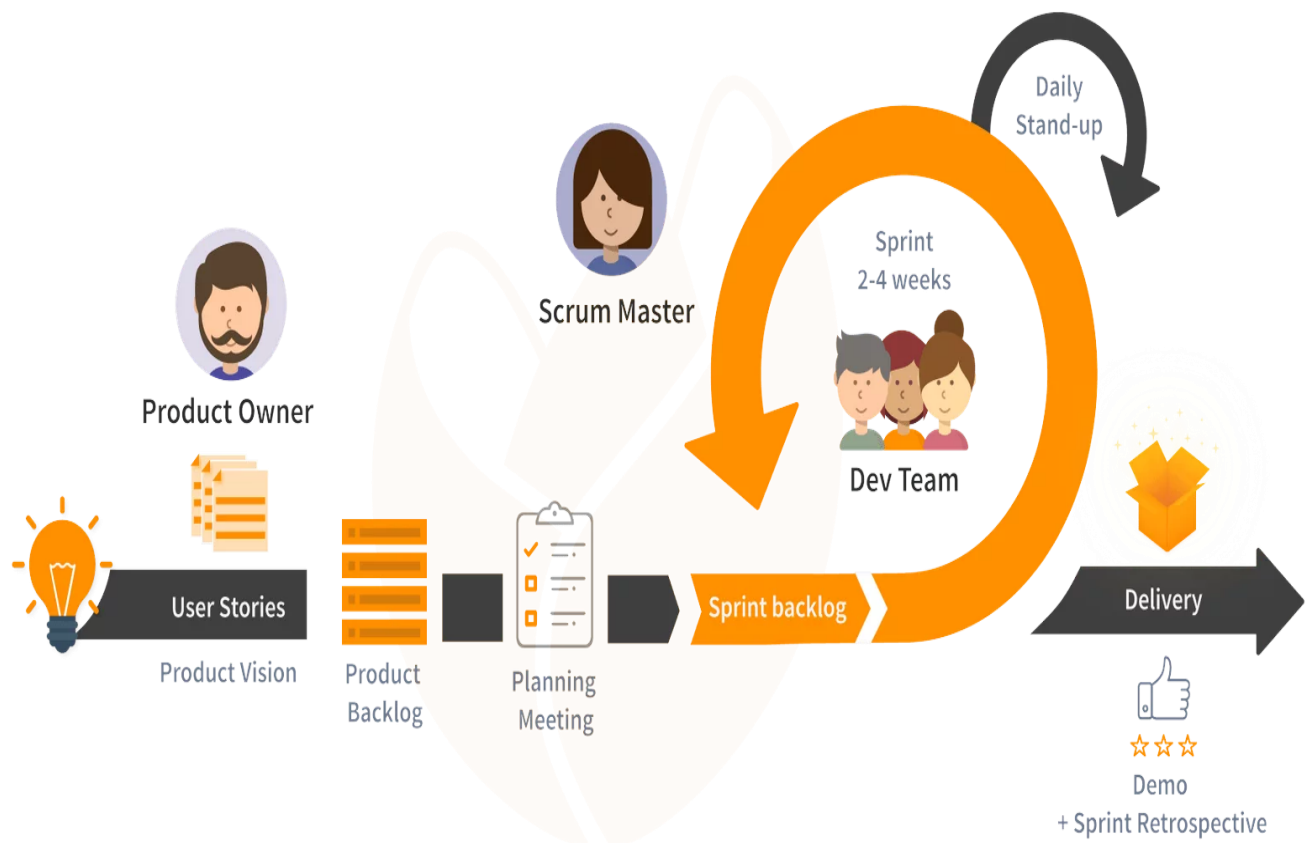


Figure 1: Cycle de la méthodologie Scrum

## Rôles dans Scrum

La méthodologie Scrum définit **trois rôles essentiels**, chacun ayant des responsabilités spécifiques :

Rôle	Responsabilités
<b>Scrum Master</b>	Facilite l'application de Scrum, supprime les obstacles et s'assure du bon déroulement du projet.
<b>Product Owner</b>	Définit les besoins, rédige les <b>User Stories</b> , et maintient le <b>Product Backlog</b> à jour.
<b>Équipe de développement</b>	Développe, teste et déploie les fonctionnalités selon les priorités définies dans les Sprints.

## Acteurs du projet

Rôle	Acteur
Product Owner	
Scrum Master	
Équipe de développement	

- Frontend : React.js, Material UI
- DevOps : Docker, Nginx
- Notifications : AWS SES, Twilio, Firebase Cloud Messaging |

## Artefacts Scrum

- **Product Backlog** : Liste de toutes les fonctionnalités et améliorations du produit, classées par priorité.
- **Sprint Backlog** : Liste des tâches à réaliser durant un **Sprint** (cycle de 2 semaines).
- **Increment** : Fonctionnalité terminée et prête à être utilisée à la fin de chaque Sprint.

## Déroulement d'un Sprint

Chaque Sprint dure **deux semaines** et suit un processus bien défini :




1. **Sprint Planning** : Définition des tâches à réaliser pendant le Sprint.
2. **Daily Scrum** : Réunion quotidienne de 15 minutes pour suivre l'avancement.
3. **Sprint Review** : Présentation des fonctionnalités développées à la fin du Sprint.
4. **Sprint Retrospective** : Analyse des points positifs et des améliorations à apporter pour le prochain Sprint.

## Organisation des Sprints pour notre projet

Sprint	Objectifs
<b>Sprint 1</b>	Mise en place de l'architecture microservices, connexion avec PostgreSQL & Redis
<b>Sprint 2</b>	Développement du module Email (AWS SES, SMTP, templates EJS)
<b>Sprint 3</b>	Développement du module SMS (Twilio)
<b>Sprint 4</b>	Développement du module Push Notifications (FCM, Web-Push)
<b>Sprint 5</b>	Création du tableau de bord React.js pour le suivi des notifications
<b>Sprint 6</b>	Tests, documentation et optimisation du projet

## Avantages de Scrum pour notre projet

- ☒ **Flexibilité** : Possibilité d'adapter les fonctionnalités en fonction des retours des utilisateurs.

-  **Meilleure visibilité** : Suivi constant de l'avancement grâce aux réunions quotidiennes.
-  **Livraison continue** : Fonctionnalités testées et livrées à chaque Sprint.
-  **Collaboration améliorée** : Communication fluide entre les développeurs, le Product Owner et les utilisateurs.

## Conclusion

La méthodologie **Scrum** permet d'assurer une gestion efficace et agile du projet "**Unified Notification Service for Developers**". Grâce à son approche **itérative et flexible**, elle nous permet d'adapter rapidement le produit aux besoins réels des développeurs et d'optimiser le processus de notification unifiée.

## **Chapitre 2**

### **Analyse et Spécification des Besoins**

#### **Introduction**

La partie analyse et spécification des besoins est une étape fondamentale pour comprendre les fonctionnalités requises par le système à développer. Durant ce chapitre, nous allons décrire les objectifs majeurs de notre application, les acteurs du système et les besoins des utilisateurs que nous allons modéliser à travers des diagrammes de cas d'utilisation. Nous terminerons en précisant les besoins non fonctionnels que le système doit assurer pour répondre aux attentes de ses utilisateurs.

#### **Identification des Acteurs**

Un acteur représente une entité externe (utilisateur humain, dispositif matériel ou autre système) qui interagit directement avec le système. Notre projet introduit les acteurs suivants :

##### **Développeur**

Un développeur est un utilisateur qui intègre le service de notification unifié dans son application et utilise l'API pour envoyer des notifications à ses propres utilisateurs.

##### **Utilisateur Final**

L'utilisateur final est la personne qui reçoit les notifications envoyées par le système via différents canaux (email, SMS, push notifications, in-app).

##### **Système Externe**

Un système tiers qui interagit avec notre service pour assurer la livraison des notifications (ex : AWS SES, Twilio, Firebase Cloud Messaging).

#### **Spécification des Besoins**

La solution à développer doit satisfaire aux exigences identifiées lors de l'étude préalable. Nous définissons ainsi les besoins fonctionnels qui seront assurés par le système, ainsi que les besoins non fonctionnels qui garantiront la qualité logicielle du projet.

##### **Besoins Fonctionnels**

Notre solution doit offrir plusieurs fonctionnalités aux différents utilisateurs.

## Développeur

- Créer un compte et obtenir une clé API.
- Configurer les canaux de notification (email, SMS, push, in-app).
- Envoyer des notifications via une API REST.
- Suivre les statistiques de livraison et les taux d'ouverture.

## Utilisateur Final

- Recevoir des notifications sur plusieurs canaux (email, SMS, push, in-app).
- Gérer ses préférences de notification (activer/désactiver certains types de messages).

## Besoins Non Fonctionnels

Le système doit respecter les exigences suivantes :

- **Scalabilité** : Capable de gérer un grand nombre de notifications simultanément.
- **Sécurité** : Authentification via JWT, gestion sécurisée des API keys.
- **Haute Disponibilité** : Utilisation de Redis pour la file d'attente et d'une architecture distribuée.
- **Performance** : Temps de réponse optimisé pour le traitement des notifications.
- **Personnalisation** : Support des templates de notification dynamique.

## Diagramme de Cas d'Utilisation Global

Le diagramme suivant présente les principales fonctionnalités du système et les interactions entre les acteurs et le système.

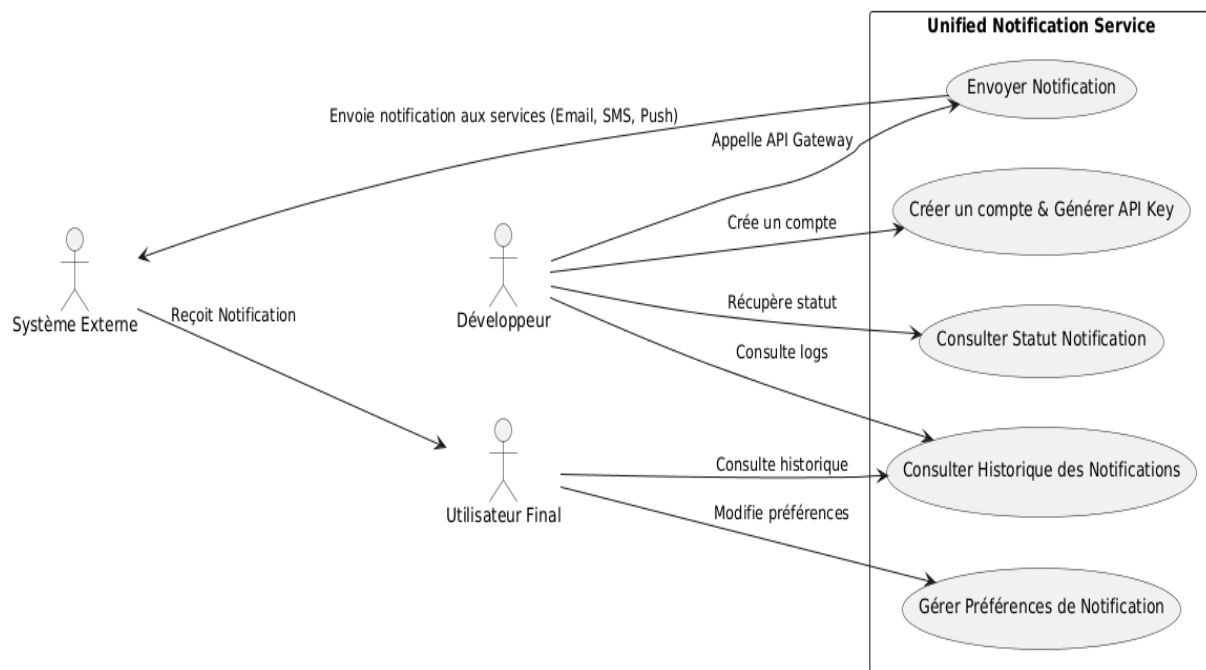


Figure 2:cas de utilisation

## 2.3Modélisation Fonctionnelle des Fonctionnalités du Système

### 2.3.1 Sprint 1

Le premier sprint est une phase essentielle dans le développement du **Unified Notification Service**. Un sprint est une période de travail définie pendant laquelle un incrément du produit est réalisé. Les sprints sont de durée fixe et ne se chevauchent pas. Avant de débiter un sprint, l'équipe doit définir un objectif clair et estimer la charge de travail pour chaque tâche. Le tableau ci-dessous présente les user stories du backlog du sprint :

Id	Histoires	Priorité
1	Un utilisateur peut recevoir des notifications	Élevée
2	Un administrateur peut configurer les canaux	Élevée
3	Un utilisateur peut gérer ses préférences de notification	Moyenne

### Diagramme de Séquence Système "Envoyer une Notification"



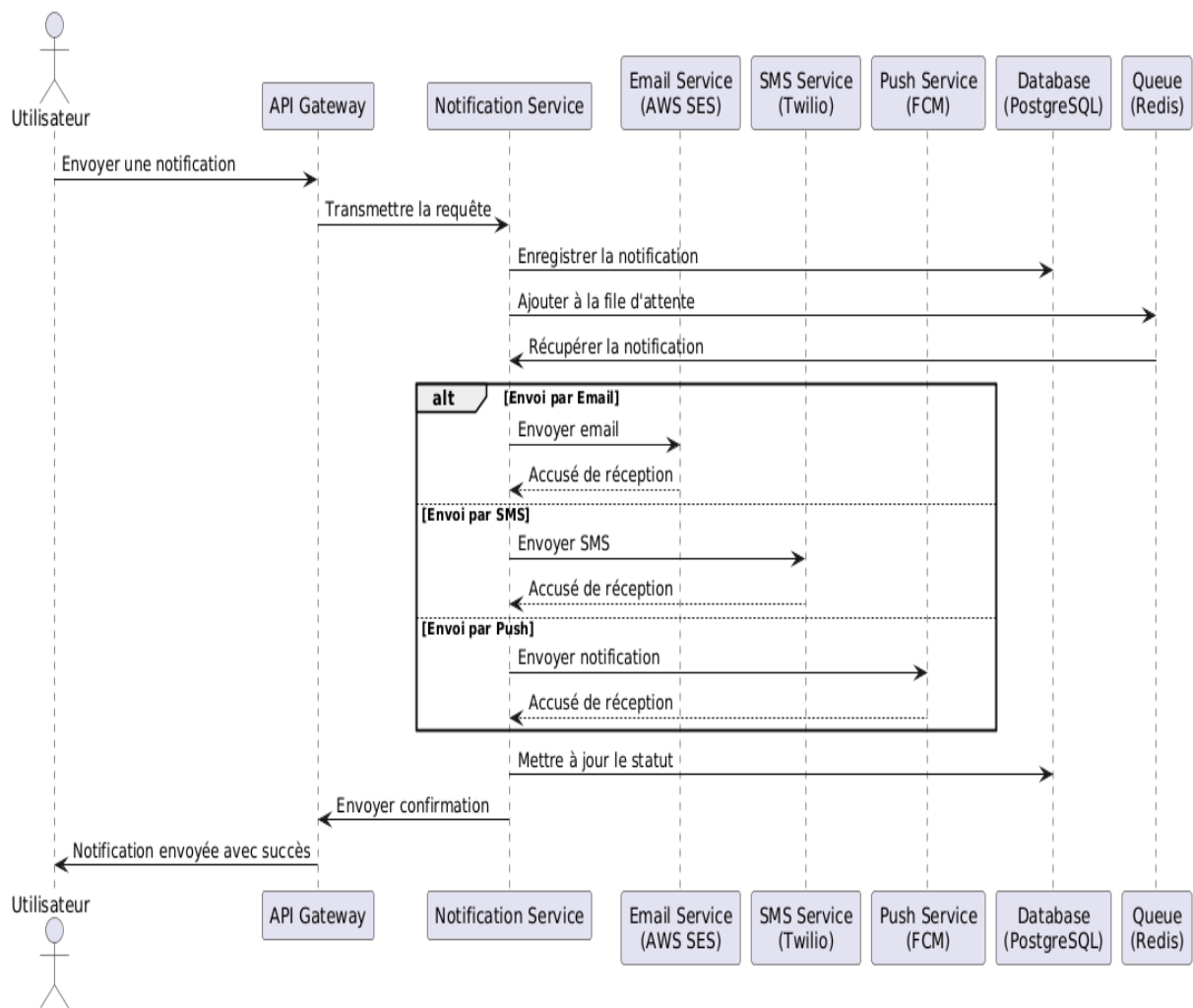


Figure 3:Diagramme de Séquence

La figure suivante illustre le diagramme de séquence du système pour l'envoi d'une notification. Ce processus met en évidence les interactions entre les acteurs et le système.

### Description du Cas d'Utilisation "Envoyer une Notification"

Le tableau suivant décrit le cas d'utilisation "Envoyer une Notification" :

Titre	Envoyer une Notification
Acteurs	
Principal	Système de notification
Secondaires	API Tiers (Firebase, Twilio, AWS SES)
Résumé	Ce cas d'utilisation permet au système d'envoyer une notification via un canal spécifique
Préconditions	L'utilisateur est inscrit et a défini ses préférences
Post condition	La notification est envoyée et confirmée

### Scénario nominal :

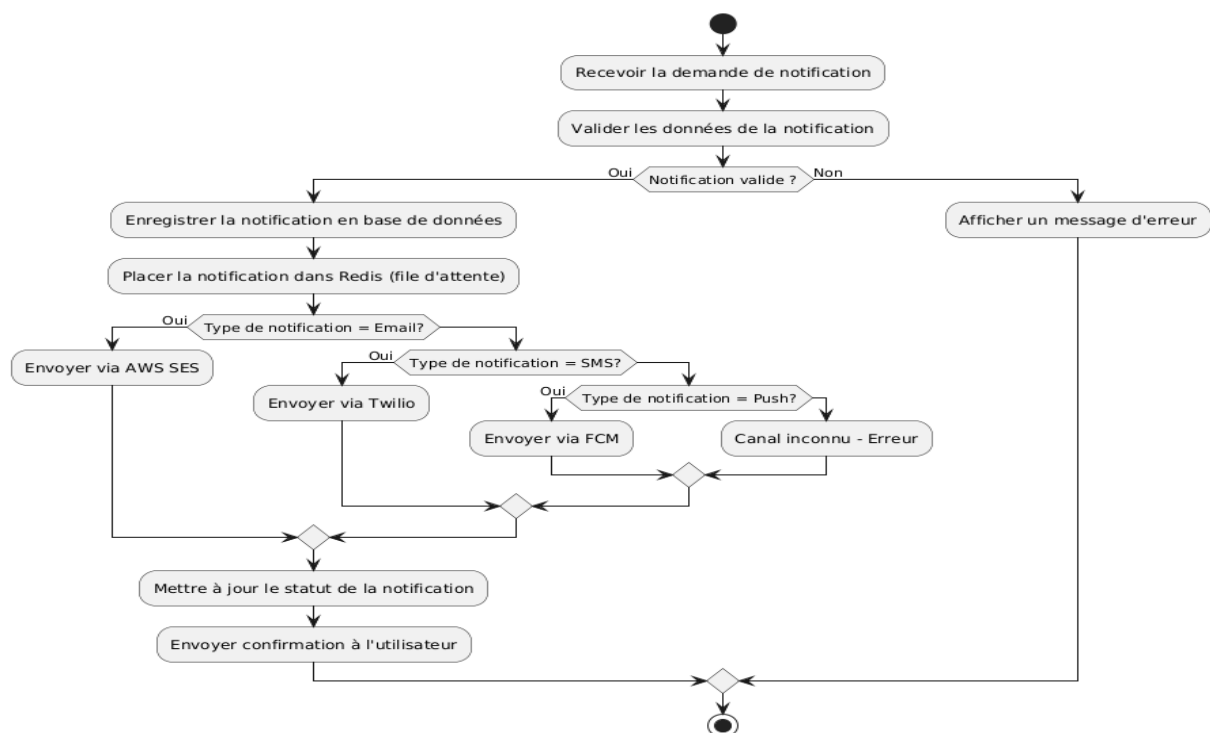
1. Le service reçoit une requête d'envoi de notification.
2. Le système valide les préférences de l'utilisateur.
3. Le système sélectionne le canal approprié (email, SMS, push).
4. Le système envoie la notification via l'API correspondante.
5. Le système reçoit une confirmation de succès.
6. Le système enregistre l'événement dans les logs.

### Scénarios d'exception :

1. L'utilisateur n'a pas défini de canal de réception.
2. Le service tiers ne répond pas.
3. Le message est rejeté par l'API externe.

### Diagramme d'Activité "Gérer les Notifications"

Ce diagramme montre les différentes étapes du processus de gestion des notifications, de la réception de la demande à la confirmation de la livraison.



### Conclusion

Ce chapitre a permis de définir les besoins fonctionnels et non fonctionnels du système, ainsi que les acteurs impliqués. Nous avons également illustré les interactions à travers un diagramme de cas d'utilisation. La prochaine étape consiste à approfondir l'architecture technique et la conception des différents composants du système.

