République Tunisienne
*****
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
*****
Université de Monastir
*****
Institut Supérieur d'Informatique et de Mathématiques de Monastir
*****
Département Informatique

N° d'ordre : L12-INFO

Mémoire de Projet de Fin d'Etudes

Présenté en vue de l'obtention du

Diplôme National de Licence en Sciences Informatique

Spécialité :

Génie Logiciel et Système d'Information

par

Adem BESBES

Veggo Control Center

Soutenu le 25/06/2025 devant le jury composé de :

Mme : Mariem GZARA	**Président**
Mme : Raghda JOUIROU	**Rapporteur**
Mme : Sana BENZARTI	**Encadrant Pédagogique**
Mr : Oussema GABTNI	**Encadrant Professionnel**

Année Universitaire : 2024 / 2025

Résumé

Ce travail s'inscrit dans le cadre de l'accomplissement de mon Projet de Fin d'Études à l'Institut Supérieur d'Informatique et de Mathématiques de Monastir pour l'année universitaire 2024-2025, en vue de l'obtention du Diplôme National de Licence Fondamentale en Sciences Informatiques. Ce projet a été effectué au sein de la société « Vermeg », avec pour objectif la conception et le développement d'une application web de gestion de Kubernetes, visant à optimiser leur organisation et leur fonctionnement. Pour réaliser ce projet, j'ai utilisé les frameworks Angular et Node.js, ainsi que PostgreSQL pour la gestion de la base de données. J'ai suivi la méthodologie 2TUP pour la conduite du projet. À travers ce document, je décris en détail les différentes étapes de réalisation de ce projet.

Mots clés: Angular, Cloud, Container, DevOps, Docker, Kubernetes, Linux, Node.js, PostgresSQL.

Abstract

This work is part of the accomplishment of my End of Studies Project at the Higher Institute of Computer Science and Mathematics of Monastir for the 2024-2025 academic year, in order to obtain the National Diploma of Fundamental License in Computer Sciences. I carried out this project within the company « Vermeg », with the objective of designing and developing a web application for Kubernetes management, aimed at optimizing its organization and operations. To carry out this project, I used the Angular and Node.js frameworks, along with PostgreSQL for database management. I followed the 2TUP methodology for project management. Through this document, I describe in detail the various stages of the project's implementation.

Keywords: Angular, Cloud, Container, DevOps, Docker, Kubernetes, Linux, Node.js, PostgresSQL.

Dédicace

Tout d'abord, je tiens à remercier DIEU de m'avoir donné la force et le courage de mener à bien ce modeste travail.

Je dédie ce travail :

À mes chers parents

Aucune expression ne saurait traduire l'amour, l'estime, le dévouement et le respect que je vous porte.

Que ce modeste travail soit l'aboutissement de vos vœux les plus chers et le fruit de vos innombrables sacrifices. Puisse Dieu, le Très-Haut, vous accorder santé, bonheur et longue vie.

À ma famille

Votre soutien inconditionnel a été mon pilier.

À mes honorables enseignants

Pour vos enseignements précieux et rigoureux, ainsi que pour vos conseils avisés. Ce travail est le résultat de nos efforts combinés, et je vous en suis profondément reconnaissant.

À tous mes amis

Merci d'avoir partagé avec moi ces années universitaires, faites de joie, de folie et de complicité. Ces moments inoubliables ont enrichi mon parcours.

Ce mémoire est le reflet d'un chemin marqué par la résilience, la discipline et la détermination. Je le dédie avec fierté à celui que je suis devenu : plus fort, plus lucide et plus confiant.

Remerciements

Avant toute chose, je souhaite exprimer ma profonde gratitude à toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce projet de fin d'études.

Je tiens à remercier sincèrement mes encadrants universitaires pour leur accompagnement constant, leurs conseils avisés et leur disponibilité tout au long de ce parcours. Leur expertise et leur soutien ont été précieux et ont grandement enrichi mon apprentissage.

Je souhaite également témoigner ma reconnaissance à toute l'équipe de l'entreprise qui m'a accueilli durant mon stage, pour leur accueil chaleureux, leur encadrement professionnel ainsi que pour les connaissances pratiques et humaines qu'ils m'ont transmises. Travailler à leurs côtés a constitué une expérience des plus enrichissantes.

Enfin, je remercie toutes les personnes, qu'elles soient proches ou éloignées, qui œuvrent chaque jour au progrès de l'humanité, guidées par des valeurs de paix, de science, de progrès et de respect mutuel.

Arrêter la Guerre

En ces temps troublés, je tiens à adresser une pensée sincère à toutes les victimes innocentes des conflits à travers le monde : en Palestine, en Israël, en Ukraine, en Russie, et ailleurs.

La guerre détruit ce que l'humanité a de plus précieux : la vie, la dignité, l'espoir. Tuer un être innocent, c'est porter atteinte à l'ensemble de l'humanité.

Il est temps de faire prévaloir l'esprit d'humanité sur la haine, la paix sur la violence, la solidarité sur la division.

Arrêtez la guerre, entendez les cris,
Sous les bombes, c'est l'enfant qui prie.
Ni drapeau, ni raison, ni gloire,
Ne valent pas une vie, ni même l'espoir.
Choisissons la paix, sans attendre demain,
Offrons l'amour au lieu du chagrin.

Table des Matières

Table des Figures

Liste des Tableaux

Liste des abréviations

- **2TUP :** Two Track Unified Process
- **AOT :** Ahead Of Time
- **AMD-V :** AMD Virtualization
- **API :** Application Programming Interface
- **AWS :** Amazon Web Services
- **AZURE :** Microsoft Azure
- **CLI :** Command Line Interface
- **CNCF :** Cloud Native Computing Foundation
- **CNI :** Container Network Interface
- **CPU :** Central Processing Unit
- **CRI :** Container Runtime Interface
- **CRI-O :** CRI-Open (Container runtime for Kubernetes)
- **CRD :** Custom Resource Definition
- **CSI :** Container Storage Interface
- **DAL :** Data Access Layer
- **DEVOPS :** Development and Operations
- **DNS :** Domain Name System
- **DOM :** Document Object Model
- **ETCD :** *(Distributed key-value store used in Kubernetes)*
- **FPGA :** Field Programmable Gate Array
- **GCP :** Google Cloud Platform
- **GO :** Giga Octet (Go)

- **GPU :** Graphics Processing Unit
- **HTML :** HyperText Markup Language
- **HTTP :** Hypertext Transfer Protocol
- **IHM :** Interface Homme-Machine
- **IDP :** Identity Provider
- **IP :** Internet Protocol
- **IPVS :** IP Virtual Server
- **JSON :** JavaScript Object Notation
- **JS :** JavaScript
- **JWT :** JSON Web Token
- **K8S :** Kubernetes (abréviation utilisant 8 lettres entre le K et le S)
- **KUB :** Kubernetes
- **KVM :** Kernel-based Virtual Machine
- **MCD :** Modèle Conceptuel de Données
- **MLD :** Modèle Logique de Données
- **MVC :** Model-View-Controller
- **NG :** Next Generation
- **NPM :** Node Package Manager
- **OAS :** OpenAPI Specification
- **OCI :** Open Container Initiative
- **OIDC :** OpenID Connect
- **QEMU :** Quick Emulator
- **RAM :** Random Access Memory
- **REST :** Representational State Transfer

- **RBAC :** Role-Based Access Control
- **RUP :** Rational Unified Process
- **SCSS :** Sassy Cascading Style Sheets
- **SPA :** Single Page Application
- **SQL :** Structured Query Language
- **TSC :** TypeScript Compiler
- **TS :** TypeScript
- **UID :** Unique Identifier
- **UML :** Unified Modeling Language
- **URI :** Uniform Resource Identifier
- **URL :** Uniform Resource Locator
- **VM :** Virtual Machine
- **VT-X :** Virtualization Technology for x86
- **WS_ENDPOINT :** Web Service Endpoint
- **YAML :** Yet Another Markup Language

Introduction Générale

Actuellement, chaque entreprise de développement informatique cherche le meilleur moyen pour satisfaire les besoins de ses clients tout en faisant face aux différents conflits qui peuvent être engendrés par un manque de coordination et d'organisation de ses équipes.

En effet, la croissance rapide du développement logiciel et l'émergence continue de nouvelles technologies imposent aux entreprises une adaptation constante, souvent difficile et chronophage. Maîtriser ces outils nécessitent du temps, de l'expertise et une rigueur accrue pour éviter les erreurs de configuration ou de déploiement des applications.

Face à ces défis, mon projet vise à concevoir une application web de contrôle et de gestion des ressources Kubernetes. Cette application a pour objectif d'offrir une interface simple, intuitive et puissante pour administrer efficacement les ressources Kubernetes. Une des principales innovations de mon outil réside dans l'introduction du concept de Teamspaces, qui permet de gérer plusieurs équipes au sein d'un même cluster Kubernetes, en assurant une isolation logique, une meilleure organisation et une meilleure collaboration.

Ce rapport retrace les différentes étapes de la réalisation de mon projet, organisé en quatre chapitres. Le premier chapitre, intitulé **« Contexte et objectifs »**, présente le cadre général du projet, l'idée à l'origine de sa conception, les solutions existantes ainsi que la méthodologie adoptée. Le deuxième chapitre, **« Analyse et spécification des besoins »**, met en lumière les besoins fonctionnels, non fonctionnels et techniques du système. Le troisième chapitre, **« Conception »**, est consacré à la modélisation de la base de données, à l'architecture logicielle de l'application ainsi qu'à la conception des interfaces utilisateur. Le quatrième chapitre, **« Réalisation »**, détaille l'environnement logiciel et matériel employé pour le développement, ainsi que les différentes interfaces de l'application. Enfin, une conclusion générale clôturera ce rapport en résumant les principaux apports du projet et en ouvrant sur des perspectives d'amélioration.

Chapitre 1. Contexte et objectif du projet

Introduction

Ce chapitre introduit le projet en présentant son cadre général ainsi que la problématique ayant conduit l'organisme d'accueil à envisager la réalisation de cette application. Une analyse des solutions existantes sera ensuite proposée, accompagnée de critiques permettant de mettre en lumière leurs limites. Les solutions envisagées pour y remédier seront alors exposées. Enfin, une étude des différentes méthodologies de travail sera menée afin d'identifier celle qui paraît la plus adaptée au contexte du projet.

1. Contexte de Travail

Cette partie est consacrée à la présentation du contexte du projet et à la définition de ses objectifs.

1.1. Cadre du projet

Ce travail s'inscrit dans le cadre du projet de fin d'études, réalisé en vue de l'obtention du diplôme de Licence Fondamentale en Informatique à « L'institut d'informatique et de mathématiques de Monastir ». Le stage, d'une durée de quatre mois, s'est déroulé au sein de la société « Vermeg » du 3 février au 30 juin 2025.

1.2. Présentation de la société d'accueil

« Vermeg » [1] est une entreprise tunisienne spécialisée dans le développement de solutions logicielles destinées aux secteurs de la banque, des marchés financiers et de l'assurance. Fondée en 1993 par Badr Eddine Ouali, Vermeg est reconnue pour son expertise dans la transformation numérique des services financiers.

VERMEG

***Figure 1.1: Logo du « Vermeg »* [1]**

L'entreprise propose une gamme complète de produits et services, notamment :

« SOLIAM », « MEGARA », « SOLIFE », « AGILE Reporter », « COLLINE », « MASSAI » et « Veggo ».

1.3. Plateforme Veggo

Veggo est une plateforme de developpement d'application a faible code (low-code), facilitant la transformation numerique des entreprises, en permettant la creation rapide d'application metier adaptées aux besoins specifiques des utilisateurs.

1.4. Objectif du projet

Ce projet a pour objectif principal la conception et le développement d'une application web intitulée **Veggo Control Center**, conçue pour améliorer l'organisation et la gestion de l'environnement **Kubernetes** au sein de l'entreprise. Cette solution vise à simplifier la configuration et le déploiement d'applications conteneurisées (basées sur Docker) via une interface utilisateur intuitive, offrant une gestion centralisée et rationalisée des ressources à l'échelle d'un cluster Kubernetes. En plus des fonctions classiques d'administration, le projet introduit la notion de **Teamspace** : un concept permettant de segmenter logiquement un cluster en plusieurs espaces de travail dédiés à différentes équipes. Chaque Teamspace dispose de ses propres ressources et droits d'accès. Cette approche favorise une organisation plus structurée, une collaboration sécurisée entre les équipes, ainsi qu'une isolation claire des environnements de travail, tout en conservant une architecture unifiée au sein d'un seul cluster.

2. Étude de l'existant

L'étude de l'existant consiste à auditer les solutions existantes pour s'inspirer et pour raffiner de plus l'idée émergente.

2.1. Solutions existantes

Cette partie présente quelques solutions existantes.

2.1.1. Kubernetes Dashboard

Le tableau de bord kubernetes [3] est une interface graphique permettant aux utilisateurs d'interagir visuellement avec leur cluster. Il simplifie l'administration et rend les opérations plus accessibles, notamment pour les utilisateurs non familiers avec l'interface en ligne de commande (kubectl). Voici ce qu'il permet:

- Déploiement rapide d'applications conteneurisées directement depuis l'interface.
- Diagnostic et résolution de problèmes, en affichant les journaux, les statuts des pods, et les erreurs rencontrées.
- Surveillance centralisée, en donnant une vue globale des ressources en cours d'exécution (pods, services, volumes, etc.).
- Gestion fine des ressources, permettant de créer, modifier ou supprimer des objets kubernetes tels que les déploiement, service, configmaps ou secrets.

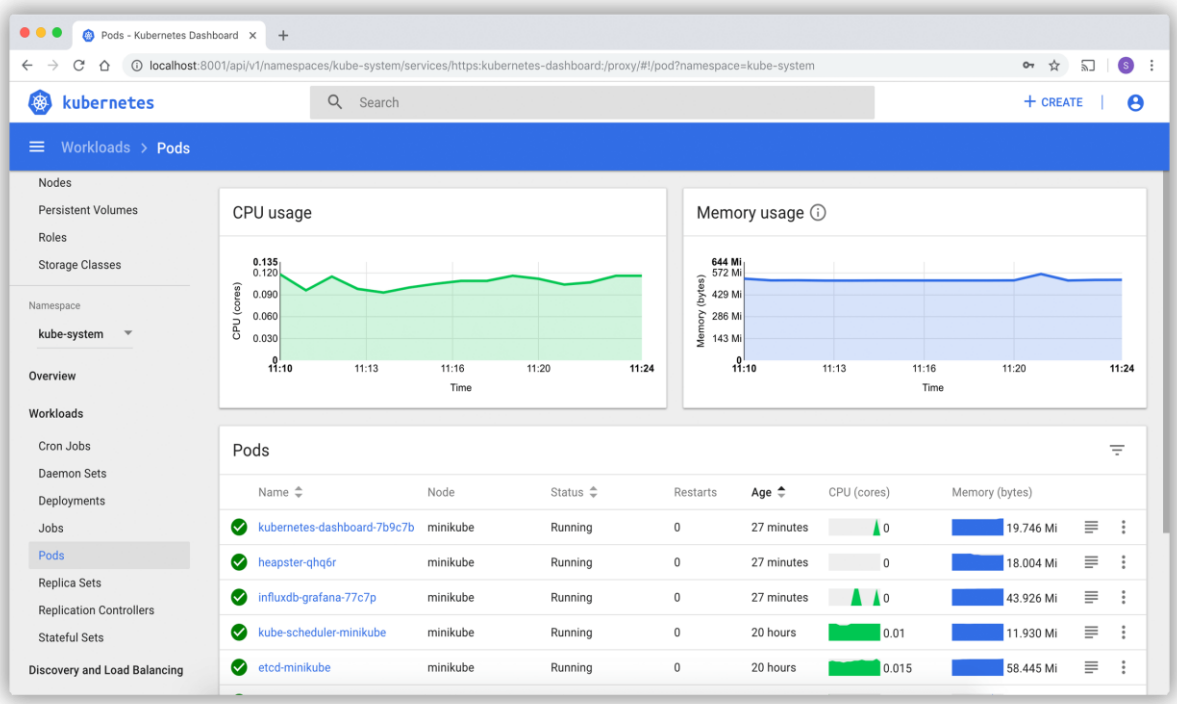

***Figure 1.2: kubernetes Dashboard* [3]**

2.1.2. Lens

Lens [4] est créé par un Team sous le nom « Team Lens » est un groupe d'individus qui a commencé avec une simple aide et maintenant il est maintenu par Mirantis, Société Historiquement spécialisée dans OpenStack, a désormais basculé vers Kubernetes., cette dernière a racheté Docker Entreprise Edition. (Elle a plus qu'un 1 million de client) Son Dashboard offre les actions suivantes :

- Supervision et Observabilité
- Gestion simplifiée des ressources kubernetes
- Amélioration de la collaboration entre équipe
- Développement et test d'application

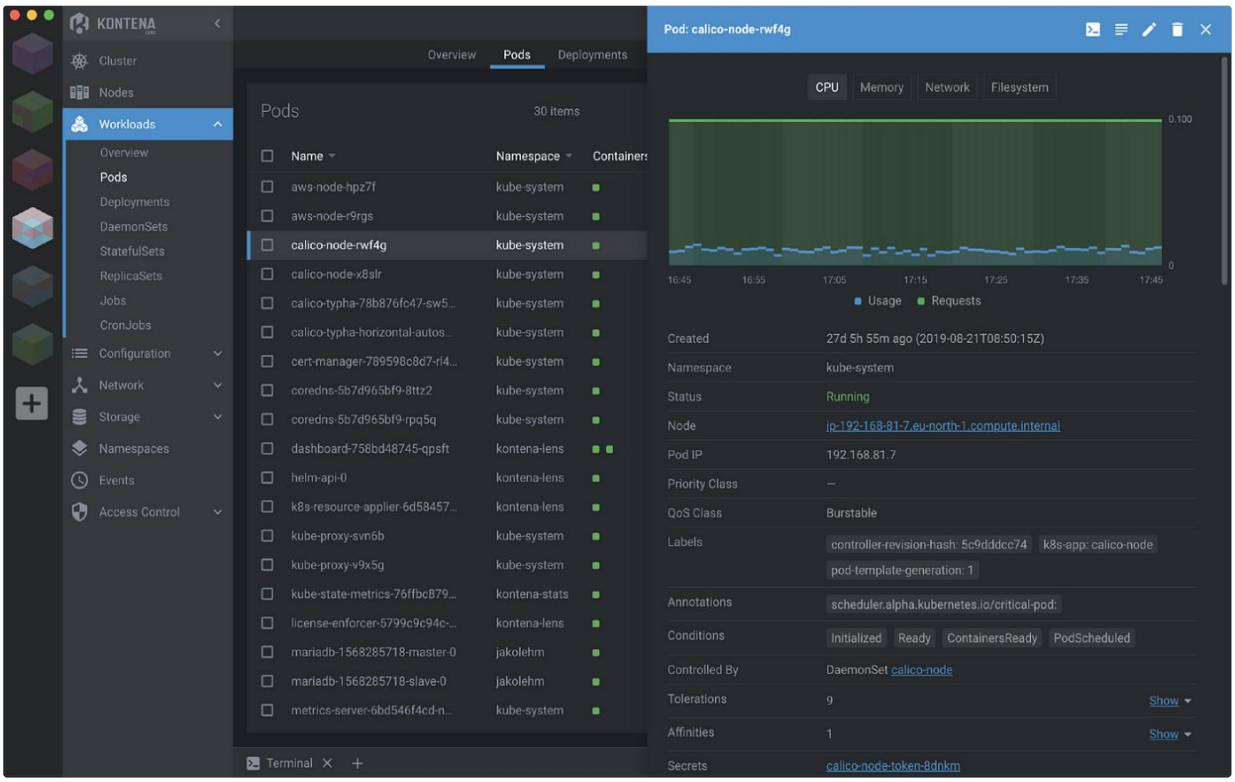

***Figure 1.3: Lens Dashboard* [4]**

2.1.3. Tableau comparatif

Les résultats de l'analyse des applications existantes mentionnées précédemment, comme l'illustre le ***Tableau 1.1***, peuvent être classés selon neuf critères (Cx) pris en considération dans le processus d'évaluation de ces applications :

Espace Group (Teamspace) (c1) : Le développement et le déploiement des applications aujourd'hui se font généralement en équipes, telles que les équipes DevOps ou DevScopes. La notion de Teamspace dans Kubernetes vise à faciliter la gestion de ces groupes dans certaines conditions spécifiques, en rendant le travail collaboratif au sein de Kubernetes plus simple et plus efficace.

Monitoring (c2) : Le système de monitoring de Kubernetes permet la visualisation en temps réel des ressources utilisées, que ce soit au niveau d'un pod ou d'une application spécifique. Il offre une simplification de la surveillance de l'usage des ressources ainsi que des calculs effectués par une application.

Logs (c3) : Les logs ont pour but de faciliter le débogage d'une application ou du système en fournissant un historique complet des événements qui se produisent.

MapVisualization (c4) : Une interface graphique interactive qui offre une représentation visuelle des ressources kubernetes et de leurs relations. Elle aide à comprendre et à concevoir l'architecture d'un système distribué notamment les microservices en mettant en évidence les interactions entre les composants d'un namespace donnée.

Turnkey Production Setup (c5) : Offre aux utilisateurs de Kubernetes la possibilité de déployer des applications préconfigurées, installées via le gestionnaire de paquets Helm à l'aide des Helm Charts.

Test du Vulnérabilités(c6) : Offre la possibilité de tester les applications après leur déploiement dans Kubernetes et d'identifier les vulnérabilités présentes.

Coût du Site (c7) : L'inscription gratuite/payante.

Notification (c8) : Permet d'envoyer des notifications sous certaines conditions concernant une application spécifique, via des requêtes vers le système de gestion des notifications de la plateforme. Cela inclut également la gestion des demandes de rejoindre une équipe et la collaboration entre les membres.

Alert Manager (c9) : Organise le système pour envoyer des alertes lorsque certaines conditions prédéfinies sont détectées.

Tableau 1.1:Tableau comparatif entre les solutions existantes

Critère	Kubernetes Dashboard	Lens
Espace Group (Teamspace) (c1)	Non	Oui
Monitoring (c2)	Oui	Oui
Logs (c3)	Oui	Non
MapVisualization (c4)	Non	Non
Turnkey Production Setup (c5)	Non	Oui
Test du Vulnérabilités(c6)	Non	Oui
Coût du Site (c7)	Gratuit	Payant
Notification (c8)	Oui	Oui
Alert Manager (c9)	Non	Non

2.2. Critique de l'existant

Aucune des applications mentionnées précédemment ne permet réellement de gérer plusieurs équipes DevOps au sein d'un seul cluster. Par exemple, dans le cas de l'application Lens, chaque équipe est confinée à un seul cluster, ce qui ne répond pas à notre besoin. Nous cherchons une solution capable de gérer plusieurs équipes dans un seul cluster tout en offrant un contrôle précis sur les applications exposées à l'extérieur. De plus, il est essentiel pour nous de pouvoir personnaliser la gestion des permissions au sein des équipes.

2.3. Travail à réaliser

De ce fait, il est essentiel de présenter une solution novatrice et pratique qui facilite le travail avec kubernetes dans un environnement organisationnel et cloud dans la notion DevOps et d'être plus actif.

Ainsi, les fonctionnalités principales que notre application doit proposer sont les suivantes :

- Pour le Développeur :
  - ✓ La simplicité de déploiement des applications kubernetes.
  - ✓ De se connecter avec des "applications déployer" avec un Terminal Emulator.
  - ✓ La configuration des ressources kubernetes.
  - ✓ Vue des applications par charge de travail. Cette vue présente l'ensemble des applications actuellement déployées dans le *namespace* sélectionné, classées par type de *workloads* (par exemple : Deployment, ReplicaSets, StatefulSets, etc.). Chaque catégorie peut être consultée indépendamment, ce qui permet une exploration ciblée des ressources. Chaque liste fournit des informations opérationnelles clés sur les *workloads*, telles que:
    - Le nombre de Pods prêts pour un ReplicaSet.
    - L'utilisation mémoire actuelle pour un Pod.
    - Le statut général de chaque objet.
  - ✓ Les vues détaillées offrent un aperçu complet de la configuration et de l'état d'un objet donné, tout en illustrant les relations entre ressources. Par exemple :
    - Les Pods gèrent par un ReplicaSet.
    - Les ReplicaSets créés par un Deployment.
    - Les liens avec des objets associés comme les HPA.
- Pour le Team Manager :
  - ✓ Gérer les rôles de développeur dans le contexte du travail.
  - ✓ Diagnostiquer les applications contre les vulnérabilités.
  - ✓ Présentation des ressources réseau utilisées pour exposer les services à l'extérieur du cluster et assurer leur découverte interne entre les composants. Les vues Services et Ingress mettent en évidence :
    - Les Pods ciblés par ces ressources.
    - Les points de terminaison internes, permettant la communication à l'intérieur du cluster.
    - Et les points de terminaison externes, accessibles aux utilisateurs ou systèmes situés hors du cluster.

- ✓ La vue de stockage affiche quant à elle les Persistent Volume Claims (PVC) associés aux applications. Ces volumes sont utilisés pour conserver les données de manière durable, même si les pods sont redémarrés ou remplacés.

➢ Pour l'administrateur :

- ✓ Affichage des nœuds, des namespaces et des volumes persistants avec accès à des vues détaillées pour chacun. La vue Liste des nœuds fournit un aperçu des niveaux d'utilisation du CPU et de la mémoire, agrégés pour l'ensemble des nœuds du cluster
- ✓ Gérer et configurer les services et l'infrastructure d'être accessible aux clients.
- ✓ Configurer les ressources comme le networking et le Stockage etc. pour certaines équipes.
- ✓ Configurer Alertmanager avec des règles d'alerte pour s'assurer que certaines applications ne dépassent pas certaines limites.
- ✓ Affichage des ressources Kubernetes utilisées pour la configuration dynamique des applications en cours d'exécution dans le cluster. Cette vue permet de consulter, modifier et gérer les objets de configuration, tout en offrant la possibilité d'afficher les secrets, masqués par défaut pour des raisons de sécurité

3. Méthodologie de travail

Avant la réalisation de chaque projet, il est essentiel de définir une méthodologie de travail permettant d'aboutir à un logiciel fiable, adaptable et efficace. Pour identifier la méthodologie la plus appropriée, une comparaison a été effectuée entre les méthodologies 2TUP et Agile Scrum.

3.1. Méthodologie 2TUP: Tow Tracks Unified Process

Le processus de développement logiciel 2TUP [5], qui s'inspire du processus unifié (RUP), propose une approche itérative et incrémentale. Il se décompose en trois branches distinctes :

- **Branche fonctionnelle** : Cette branche capitalise sur la connaissance métier de l'entreprise en capturant les besoins fonctionnels des utilisateurs finaux. Elle se concentre sur la création d'un modèle centré sur ces besoins.

- **Branche technique** : Ici, le savoir-faire technique et les contraintes techniques sont pris en compte. Les solutions techniques développées sont indépendantes des fonctions à réaliser et visent à répondre aux défis spécifiques du système.

- **Branche de réalisation :** Cette branche vise à concrétiser les deux branches précédentes en menant une conception applicative et en livrant une solution adaptée aux besoins identifiés.

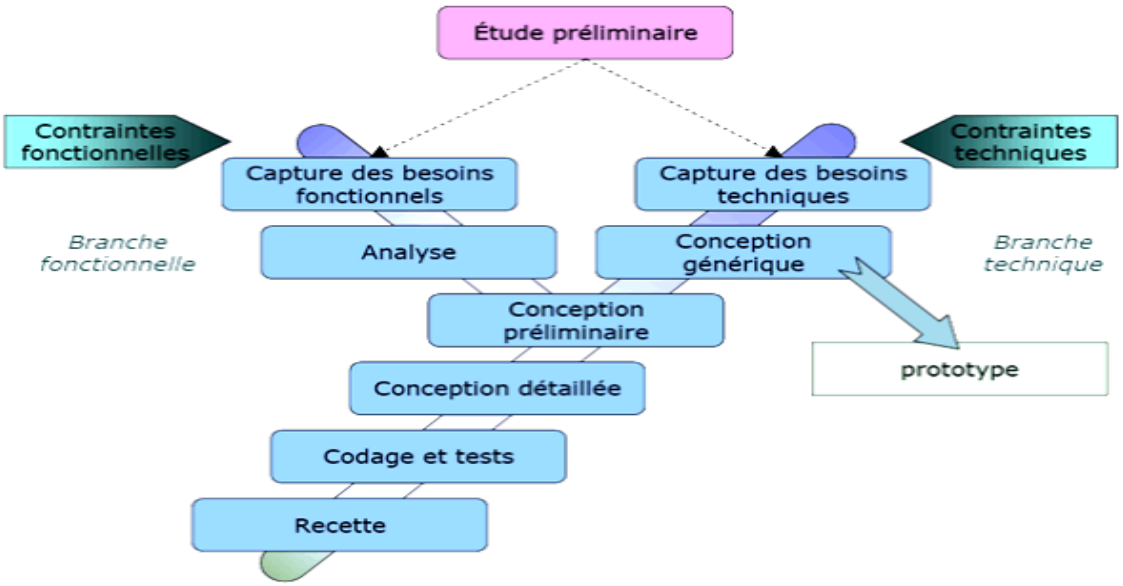

***Figure 1.4: Représentation de la méthodologie 2TUP* [5]**

3.2. Méthodologie agile Scrum

Scrum [5], baptisée ainsi en référence au rugby, prône une approche dynamique de la gestion de projet. Les rôles clés incluent :

- **Le Scrum Master :** Responsable de veiller à l'application correcte des principes de Scrum, ce rôle agit comme un serviteur-leader pour l'équipe, sans être un chef de projet à proprement parler.
- **Le Product Owner :** Faisant le lien entre les besoins métier et techniques, ce rôle est chargé de maintenir le Product Backlog à jour et de rédiger les user stories en collaboration avec les clients.
- **L'équipe de développement :** Composée de professionnels aux compétences variées, cette équipe est chargée de transformer les besoins en fonctionnalités concrètes et utilisables.

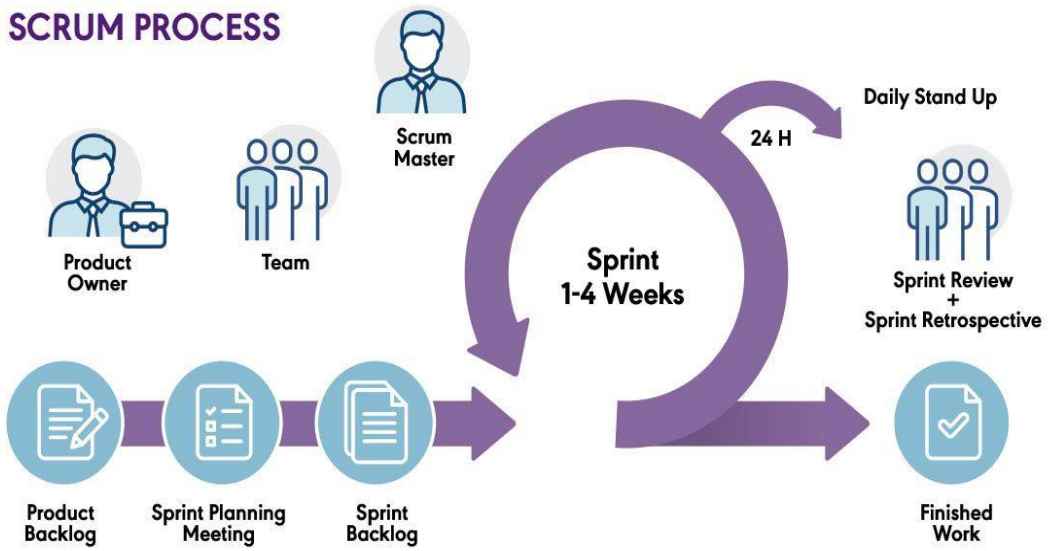

Figure 1.5: Représentation de la méthodologie Scrum **[5]**

3.3. Méthodologie adoptée

Après une analyse approfondie des différentes approches méthodologiques, le choix s'est porté sur le processus **2TUP** pour encadrer le déroulement du projet. Cette méthodologie s'est avérée particulièrement adaptée au contexte du travail, car elle propose un cadre de développement structuré, allant de l'identification des besoins fonctionnels jusqu'à la conception et l'implémentation de la solution. De plus, le 2TUP répond efficacement aux exigences liées à la portée et à la durée du projet.

Conclusion

En conclusion, ce premier chapitre a permis de poser les bases du projet en présentant son cadre global, ainsi que d'expliquer les objectifs spécifiques de l'application dans un environnement fédéré. J'ai également abordé l'importance de mettre en place une plateforme centralisée et intuitive, visant à simplifier la gestion et le déploiement de Kubernetes. Dans le chapitre suivant, je me concentrerai sur la spécification détaillée des besoins fonctionnels, ainsi que sur les techniques que l'application devra intégrer afin de répondre efficacement aux objectifs de simplification et de réduction de la complexité du développement.

Chapitre 2. Analyse et spécification des besoins

Introduction

L'étape d'analyse et de spécification des besoins constitue une phase essentielle pour définir avec précision les fonctionnalités attendues du système. Ce chapitre a pour objectif de présenter les principales fonctionnalités envisagées, ainsi que les différents acteurs impliqués dans l'utilisation de l'application. Les besoins seront ensuite formalisés à travers des diagrammes de cas d'utilisation et des diagrammes de séquence, permettant de représenter de manière claire les interactions entre les utilisateurs et le système, ainsi que le comportement attendu de ce dernier.

1. Analyse et identification des besoins

Cette section a pour objectif d'identifier les différents acteurs du système ainsi que les rôles qu'ils y jouent. Elle s'intéresse ensuite à l'analyse des besoins fonctionnels et non fonctionnels que le système doit satisfaire, afin de garantir une réponse adaptée aux attentes des utilisateurs et aux exigences techniques du projet.

1.1. Identification des acteurs

Un acteur est une entité externe qui définit le rôle d'un utilisateur humain ou non humain, qui interagit avec un système interactif. Notre plateforme comporte les acteurs suivants :

- **Développeur :** C'est un développeur qui travaille chez Vermeg ou dans une autre entreprise utilisant la plateforme Veggo.
- **Chef d'équipe (Team Manager)** : C'est un chef d'équipe travaillant chez Vermeg ou dans une autre entreprise, responsable de l'organisation du flux de projet et de la coordination des équipes DevOps, composées de développeurs.
- **Administrateur :** C'est un administrateur travaillant chez Vermeg, maîtrisant Kubernetes.

2. Modèle informationnel de contexte

Cette section présente le modèle informationnel du contexte, décrivant les principales entités, leurs caractéristiques et les relations qui structurent les informations essentielles du domaine étudié.

2.1. Diagramme du contexte

Le diagramme de contexte est une représentation graphique simplifiée du système à modéliser, illustré sous forme d'une boîte noire. Il met en évidence les interactions entre le système et son environnement externe, en identifiant les acteurs ou entités externes qui échangent des informations avec lui. Ce diagramme permet de délimiter clairement les frontières du système, en précisant les flux d'informations entrants et sortants. Il constitue une étape fondamentale dans le processus de modélisation, car il offre une vue d'ensemble du système sans en exposer les détails internes, facilitant ainsi la compréhension globale de son fonctionnement.

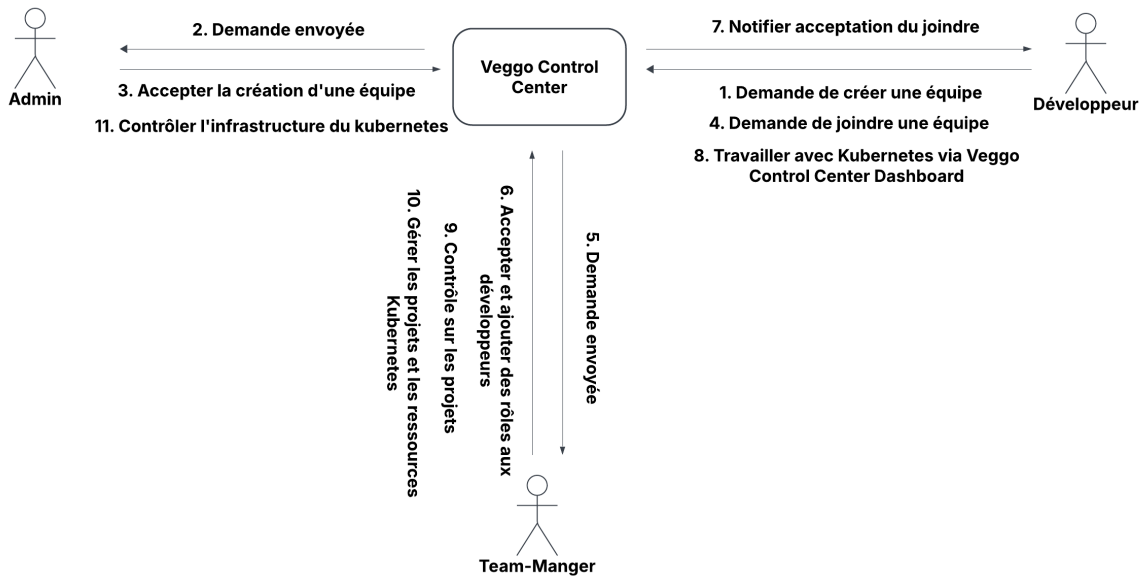

Figure 2.1: Diagramme de contexte

2.2. Besoins Fonctionnels

Les besoins fonctionnels correspondent aux services et fonctionnalités que le système doit fournir afin de répondre aux attentes des différents utilisateurs. Ces besoins sont définis en fonction des rôles spécifiques identifiés au sein de l'application, à savoir : Utilisateur, Développeur, Chef d'équipe et Administrateur.

- L'application doit permettre à l'utilisateur de :
  - ✓ **S'inscrire :** L'utilisateur a la possibilité de créer un profil sur le Veggo Control Center afin d'accéder aux fonctionnalités réservées aux développeurs.
- L'application doit permettre au Développeur de :
  - ✓ **S'authentifier :** après avoir créé un compte, le Développeur peut se connecter à son profil.
  - ✓ **Demande de Créer une équipe :** Le développeur peut faire une demande de création d'une équipe.
  - ✓ **Rejoindre une équipe :** le Développeur peut soumettre une demande pour rejoindre une équipe existante.
  - ✓ **Recherche et déployer des applications avec Helm Charts :** le Développeur peut rechercher des packages applicatifs préconfigurés (Helm Charts), les configurer et les déployer dans une Namespace sélectionnée.
  - ✓ **Se connecter à un Pod (application) :** le Développeur peut ouvrir un terminal pour accéder à un Pod et exécuter des scripts Shell afin d'interagir avec l'application.
  - ✓ **Créer des ressources Kubernetes :** le Développeur peut créer des ressources telles que des Pods, Deployments, Services, etc., via un formulaire, un éditeur web, ou en important un fichier YAML ou JSON.
  - ✓ **Consulter les logs :** le Développeur peut consulter les logs des applications pour diagnostiquer d'éventuelles erreurs ou comportements anormaux.
  - ✓ **Visualiser les Ressources :** le Développeur peut voir toutes les ressources Kubernetes disponibles dans une Namespace spécifique.
  - ✓ **Adapter la scalabilité des ressources :** le Développeur peut augmenter ou réduire le nombre d'instances d'une ressource (ReplicaSet) en fonction des besoins.
- L'application doit permettre au Chef d'équipe de :
  - ✓ **Gérer les membres de l'équipe :** le Chef d'équipe peut accepter, ajouter ou supprimer des développeurs de son équipe.
  - ✓ **Attribuer des rôles aux membres :** le Chef d'équipe peut définir les rôles et permissions de chaque développeur.
  - ✓ **Surveiller l'état des applications et scanner les vulnérabilités :** le Chef d'équipe peut analyser les applications déployées à l'aide de l'outil Trivy,

identifier les failles de sécurité potentielles et télécharger un rapport de diagnostic.

- ✓ **Visualiser l'architecture Kubernetes :** le Chef d'équipe peut consulter une représentation graphique de l'architecture Kubernetes montrant les relations et dépendances entre les différentes ressources.

- L'application doit permettre à l'administrateur de :
  - ✓ **S'authentifier :** l'administrateur peut se connecter à l'espace d'administration via un identifiant et un mot de passe.
  - ✓ **Gérer les équipes :** l'Administrateur peut valider ou supprimer la création d'équipes.
  - ✓ **Administrer l'architecture du cluster Kubernetes :** il s'assure que les conditions nécessaires sont en place pour que les équipes puissent travailler efficacement avec Kubernetes.
  - ✓ **Mettre en œuvre les bonnes pratiques de déploiement :** l'Administrateur peut ajouter des ressources complexes (Network Policies, Load Balancer, Control Plane, Persistent Volume, etc.) non accessibles directement aux développeurs.
  - ✓ **Intégrer les outils de monitoring :** l'Administrateur est chargé de la configuration des serveurs de surveillance tels que Prometheus et Grafana.
  - ✓ **Configurer l'Alert Manager :** il définit les conditions d'alerte dans Prometheus et Grafana, et configure l'envoi automatique d'alertes par e-mail.
  - ✓ **Gérer les utilisateurs :** il peut gérer les autorisations et les jetons d'accès des utilisateurs, en fonction de leur rôle dans le cluster.

2.3. Besoins Non Fonctionnels

Les exigences non-fonctionnelles encadrent les conditions de bon fonctionnement du système et l'optimisation des services fournis à l'utilisateur. Notre application doit respecter les critères suivants :

- ✓ Performance : la plateforme doit être en mesure de prendre en charge simultanément un grand nombre d'utilisateurs, les temps de réponse, la disponibilité du système, la robustesse.

- ✓ Caching : le système doit intégrer un mécanisme de mise en cache afin d'optimiser les performances, réduire les temps de chargement des ressources fréquemment utilisées et limiter les requêtes répétitives vers le backend.
- ✓ Fiabilité : la disponibilité constante de la plateforme est essentielle assurant ainsi une expérience d'équipe fiable et cohérence en permanence.
- ✓ Sécurité : Il est impératif de garantir la confidentialité des données et de mettre en place des mesures de protection contre les attaques externes afin de préserver l'intégrité du système.
- ✓ Evolutivité : la plateforme doit pouvoir évoluer en fonction de la demande des membres et de l'administration. Il doit être capable de gérer de manière efficiente une croissance rapide du nombre d'utilisateurs ainsi que le volume de données stockées sur la plateforme.

3. Spécification des besoins fonctionnels

Cette étape vise à contextualiser le système en comprenant son environnement. Cela implique l'identification des fonctionnalités clés, des acteurs pertinents, la clarification des risques critiques, et la reconnaissance des premiers cas d'utilisation.

3.1. Diagramme de cas d'utilisation général

Un modèle de cas d'utilisation décrit les fonctionnalités proposées pour un nouveau système. Il représente une unité d'interaction discrète entre un utilisateur (humain ou machine) et le système. Cette interaction constitue une unité de travail significative. Chaque cas d'utilisation décrit les fonctionnalités à intégrer au système proposé, qui peuvent inclure celles d'un autre cas d'utilisation ou étendre un autre cas d'utilisation avec son propre comportement.

Afin d'alléger le diagramme des cas d'utilisation, les fonctionnalités techniques transversales telles que l'authentification et la vérification des droits d'accès ne sont pas représentées individuellement. Il est supposé que le chef d'équipe et le développeur doivent être authentifiés et autorisés avant de pouvoir exécuter les cas d'utilisation décrits. La **Figure 2.2** présente le diagramme général des cas d'utilisation.

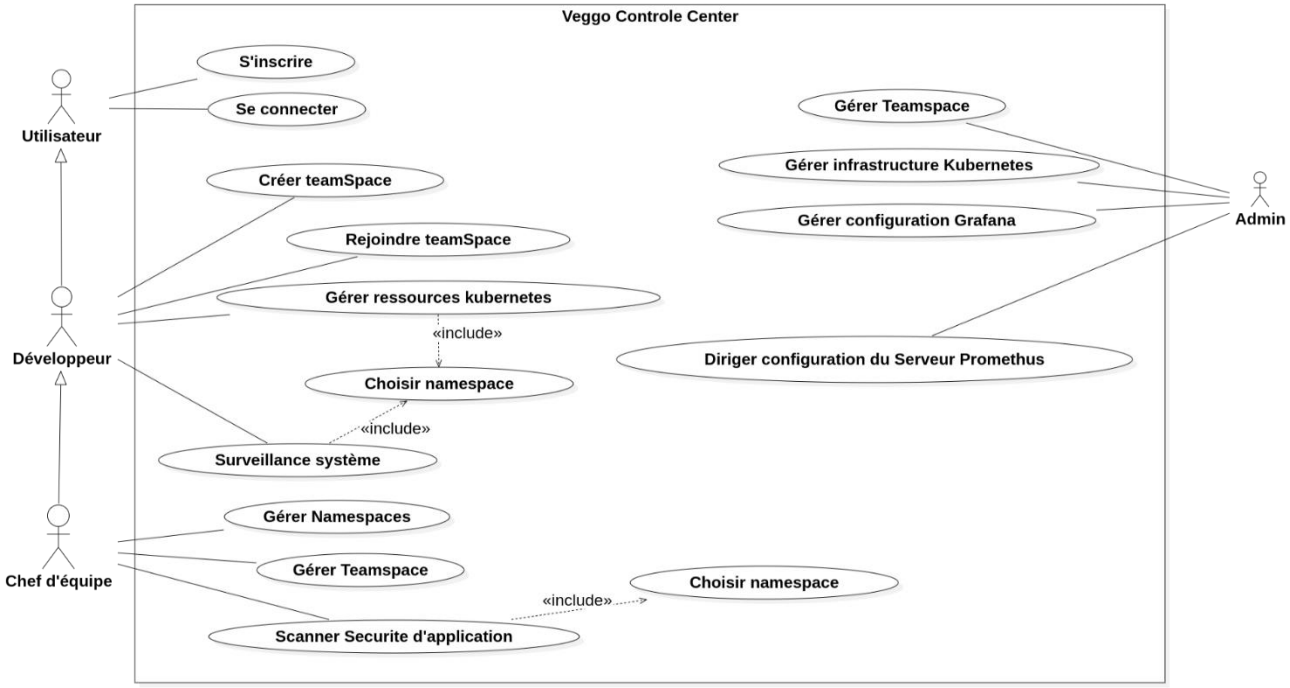

Figure 2.2: Diagramme de cas d'utilisation général

3.2. Diagrammes de cas d'utilisation détaillés

Dans cette section, les rôles et responsabilités de chaque acteur seront détaillés.

3.2.1. Diagramme de cas d'utilisation détaillés de l'acteur « Développeur »

La **Figure 2.3** présente les fonctionnalités accessibles à l'acteur « Développeur » une fois qu'il est authentifié et dispose des droits d'accès nécessaires.

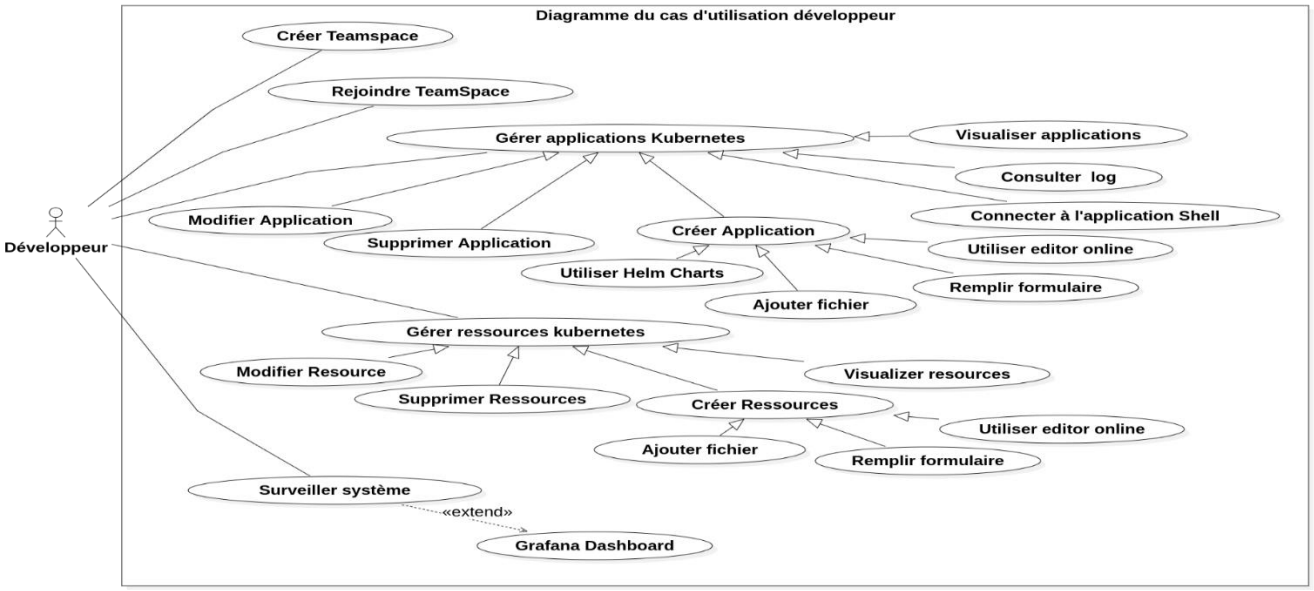

Figure 2.3: Diagramme de cas d'utilisation de l'acteur « Développeur »

3.2.2. Diagramme de cas d'utilisation détaillés de l'acteur « Chef d'équipe »

La **Figure 2.4** présente les fonctionnalités accessibles à l'acteur « Chef d'équipe » après authentification et vérification de son rôle.

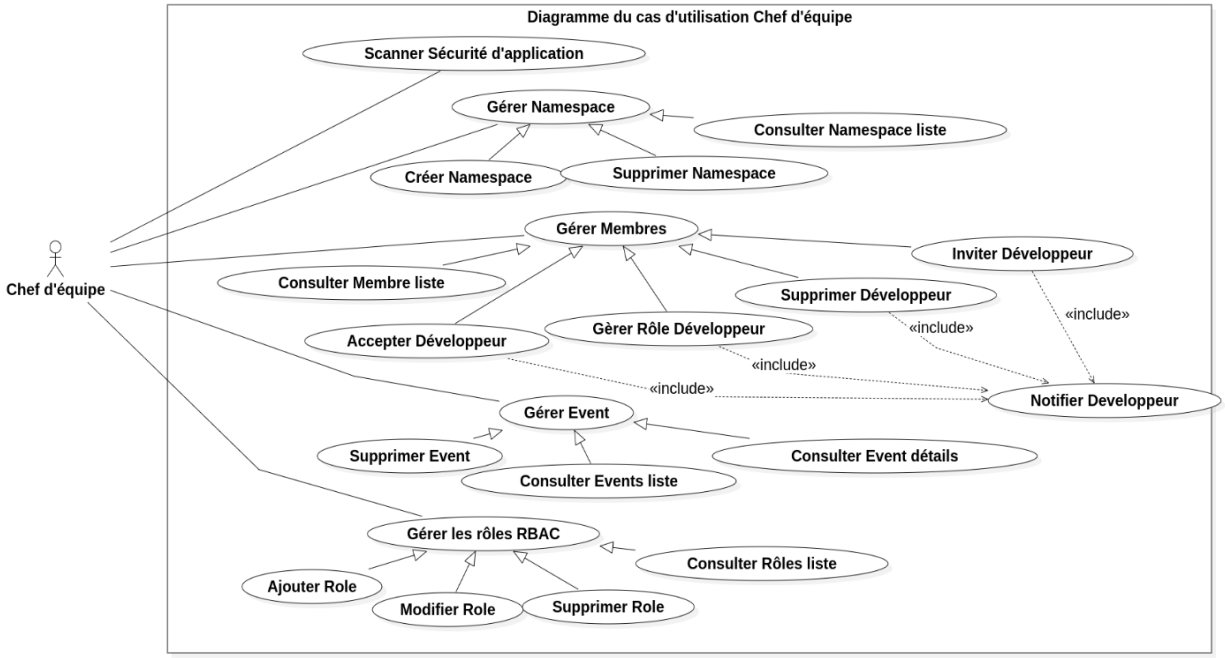

Figure 2.4: Diagramme de cas d'utilisation de l'acteur « Chef d'équipe »

3.2.3. Diagramme de cas d'utilisation détaillés de l'acteur « Administrateur »

La **Figure 2.5** présente les fonctionnalités accessibles à l'acteur « Administrateur » après authentification.

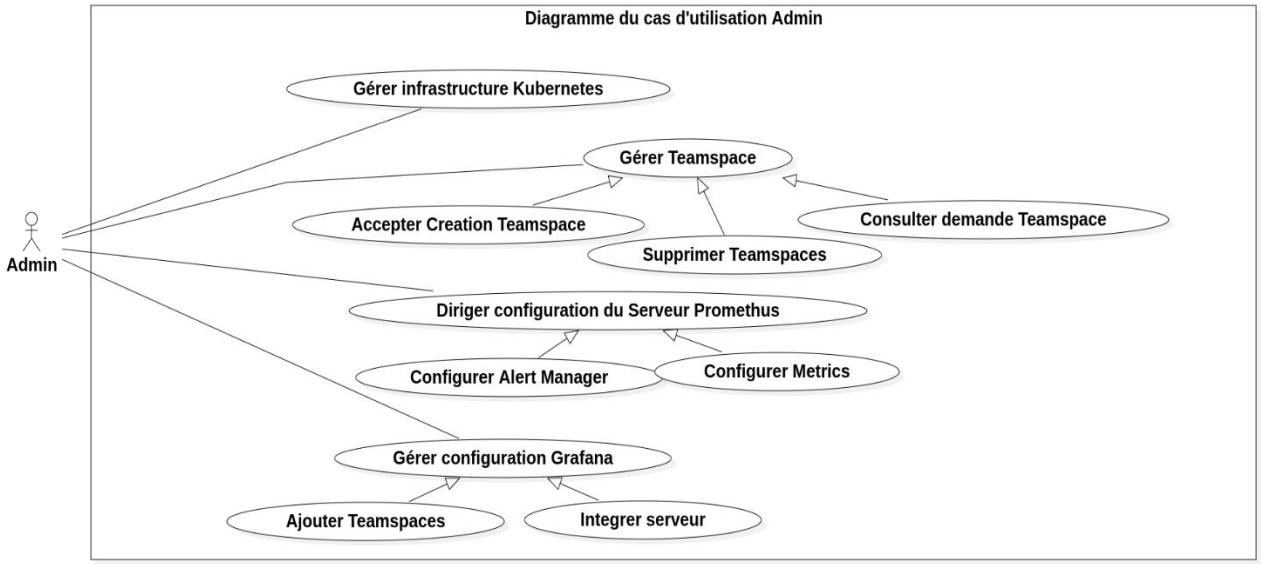

Figure 2.5: Diagramme de cas d'utilisation de l'acteur « Admin »

3.3. Diagramme de séquence acteur-système

Les diagrammes de séquence permettent de visualiser les interactions entre les utilisateurs, les écrans, les objets et les entités du système. Ils fournissent une cartographie séquentielle des messages échangés entre les objets au fil du temps. Ces diagrammes sont souvent placés sous les cas d'utilisation du modèle afin d'illustrer le scénario d'utilisation.

3.3.1. Diagramme de séquence « Créer kubernetes Ressources »

Le ***Tableau 2.1*** et la **Figure 2.6** présentent respectivement la description et la représentation du diagramme de séquence intitulé « Créer Kubernetes Ressources ».

Tableau 2.1:Description du Cas d'Utilisation « Créer kubernetes Ressources »

Cas d'utilisation	Créer kubernetes Ressources
Acteur	Développeur ou Chef d'équipe ou admin
Intérêt	Permet aux acteurs mentionnés de créer des ressources Kubernetes (Pods, Deployments, Services, etc.).
Précondition	L'utilisateur (Développeur ou Chef d'équipe ou admin) doit être authentifié et disposer des autorisations nécessaires pour créer des ressources Kubernetes.
Scénario nominal	- Le développeur demande l'accès à la page de gestion des ressources kubernetes. -Le système vérifie les droits d'accès : ➢ **Si l'utilisateur n'est pas autorisé :** ✓ Un message d'erreur s'affiche, indiquant que l'accès est refusé. ➢ **Si l'utilisateur est autorisé** : ✓ Le système lui propose de choisir une méthode de création des ressources : -L'utilisateur choisit l'une des trois méthodes suivantes :

	a) Utiliser un formulaire interactif - L'utilisateur remplit les champs requis - Le système valide les données : ➢ Si le fichier n'est pas conforme (mauvais format ou erreur de syntaxe) : ✓ Un message d'erreur apparaît, précisant la nature du problème détecté. ➢ Si le fichier est valide : ✓ La ressource est créée, et un message de confirmation s'affiche. c) Importer un fichier de configuration - L'utilisateur téléverse un fichier YAML ou JSON. - Le système vérifie le format et le contenu : ➢ Si le fichier n'est pas conforme (mauvais format ou erreur de syntaxe) : ✓ Un message d'erreur apparaît, précisant la nature du problème détecté. ➢ Si le fichier est valide : ✓ La ressource est créée, et un message de confirmation s'affiche. c) Utiliser un éditeur web - L'utilisateur saisit manuellement la configuration YAML dans un éditeur intégré. - Le système vérifie la validité du contenu : ➢ En cas d'erreurs (syntaxe YAML ou ressources invalides): ✓ Un message d'erreur s'affiche. ➢ Si tout est correct:

	✓ La ressource est créée, et un message de succès est affiché.
Post Condition	Une ressource Kubernetes est créée avec succès.
Exception	➢ Si la configuration ne respecte pas les règles Kubernetes ou la syntaxe YAML : ✓ Le système affiche un message d'erreur précisant la nature du problème. ➢ Si un champ requis dans le formulaire est vide ou invalide : ✓ Une erreur est affichée avec les détails. ➢ Si un fichier importé ne respecte pas la structure attendue : ✓ Une erreur est générée après l'analyse (parsing) du fichier. ➢ Autorisation insuffisante : Si l'utilisateur ne dispose pas des droits nécessaires pour gérer les ressources Kubernetes, le système bloque l'accès et affiche un message d'erreur indiquant que l'opération est interdite.

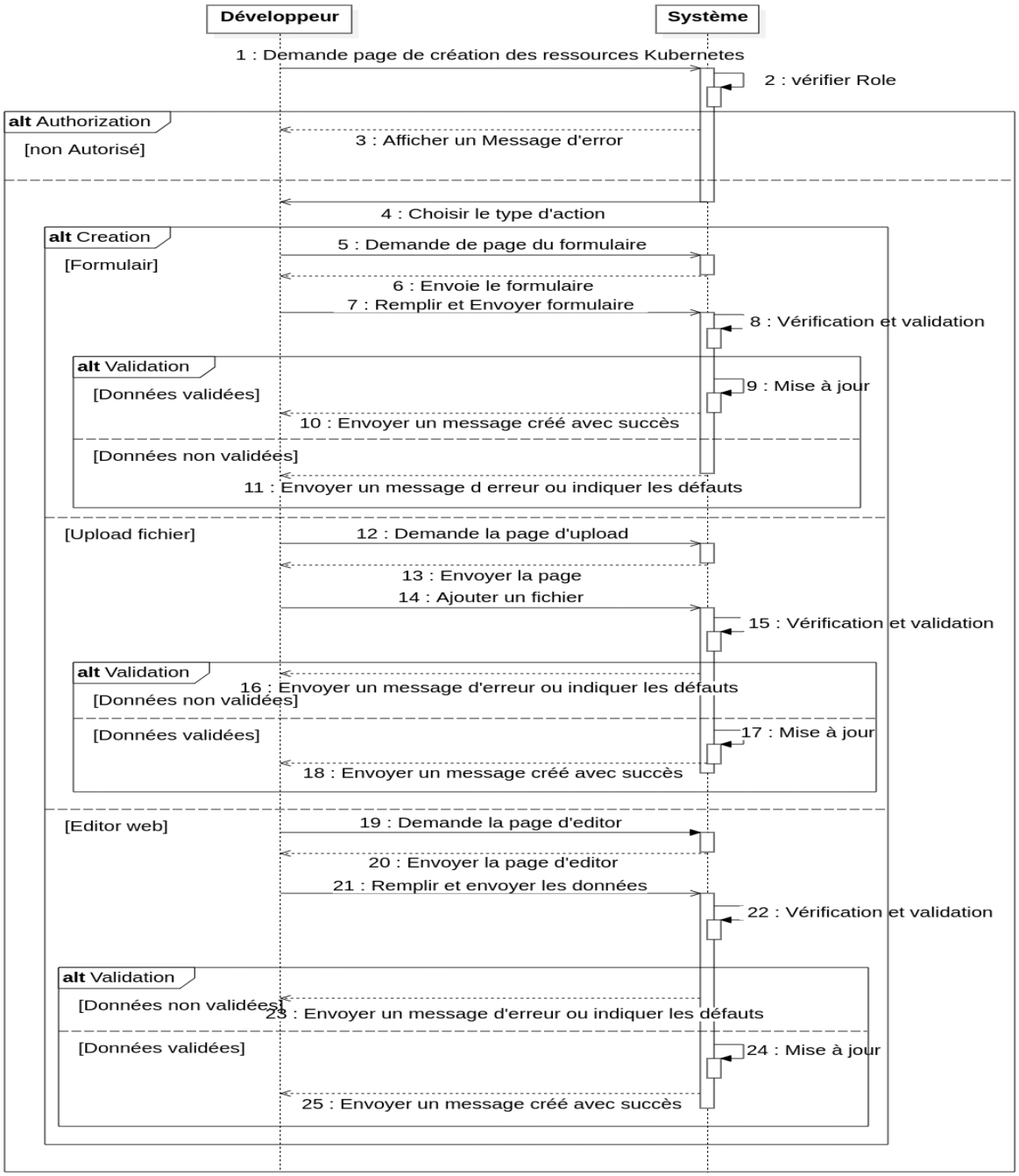

Figure 2.6: Diagramme de séquence « Créer Ressources kubernetes »

3.3.2. Diagramme de séquence « Gérer Namespace »

Le ***Tableau 2.2*** et la **Figure 2.7** présentent respectivement la description et la représentation du diagramme de séquence intitulé « Gérer Namespace ».

Tableau 2.2: Description du Cas d'Utilisation «Gérer Namespace»

Cas d'utilisation	Gérer les Namespace

Acteur	Chef d'équipe
Intérêt	Permet au Chef d'équipe d'organiser et gérer les *Namespaces* au sein de son *Teamspace*, facilitant la séparation logique des ressources Kubernetes.
Précondition	Le Chef d'équipe doit être authentifié.
Scénario Nominal	- Le chef d'équipe accède à l'interface de gestion des *Namespaces.* - Le système affiche la liste des namespaces actuellement associés à l'équipe. ➢ Ajout d'un Namespace : • Le Chef d'équipe demande la création d'un nouveau Namespace. • Le système affiche la page contenant le formulaire de création. • Le Chef d'équipe remplit les champs nécessaires et soumet le formulaire. • Le système contrôle la validité des données saisies ✓ Si les données sont valides : le Namespace est créé, et la liste est mise à jour ✓ Un message de confirmation est affiché ➢ Suppression d'un Namespace : • Le Chef d'équipe sélectionne un Namespace existant et demande sa suppression. • Le système vérifie si des ressources sont encore présentes dans le Namespace. ✓ Si oui : un message de confirmation est affiché pour prévenir des conséquences de la suppression. • Après confirmation, le système supprime le Namespace et met à jour la liste.

Post Condition	Un Namespace est soit créé, soit supprimé avec succès.
Exception	• **Nom invalide** : Si le nom du Namespace ne respecte pas les conventions de nommage de Kubernetes (caractères interdits, format incorrect, nom déjà existant, etc.), Un message d'erreur apparaît, précisant la nature du problème détecté. • **Champ obligatoire manquant** : Si le formulaire de création contient des champs vides ou mal remplis, une erreur est affichée avec les détails des champs à corriger. • **Ressources présentes dans le Namespace :** Lors de la tentative de suppression, si des ressources sont encore présentes dans le Namespace, le système demande une confirmation avant de procéder. Si l'utilisateur annule, la suppression est abandonnée. • **Droits insuffisants :** Si le Chef d'équipe ne possède pas les autorisations nécessaires pour gérer les Namespaces, l'accès à cette fonctionnalité est refusé, avec un message d'erreur approprié.

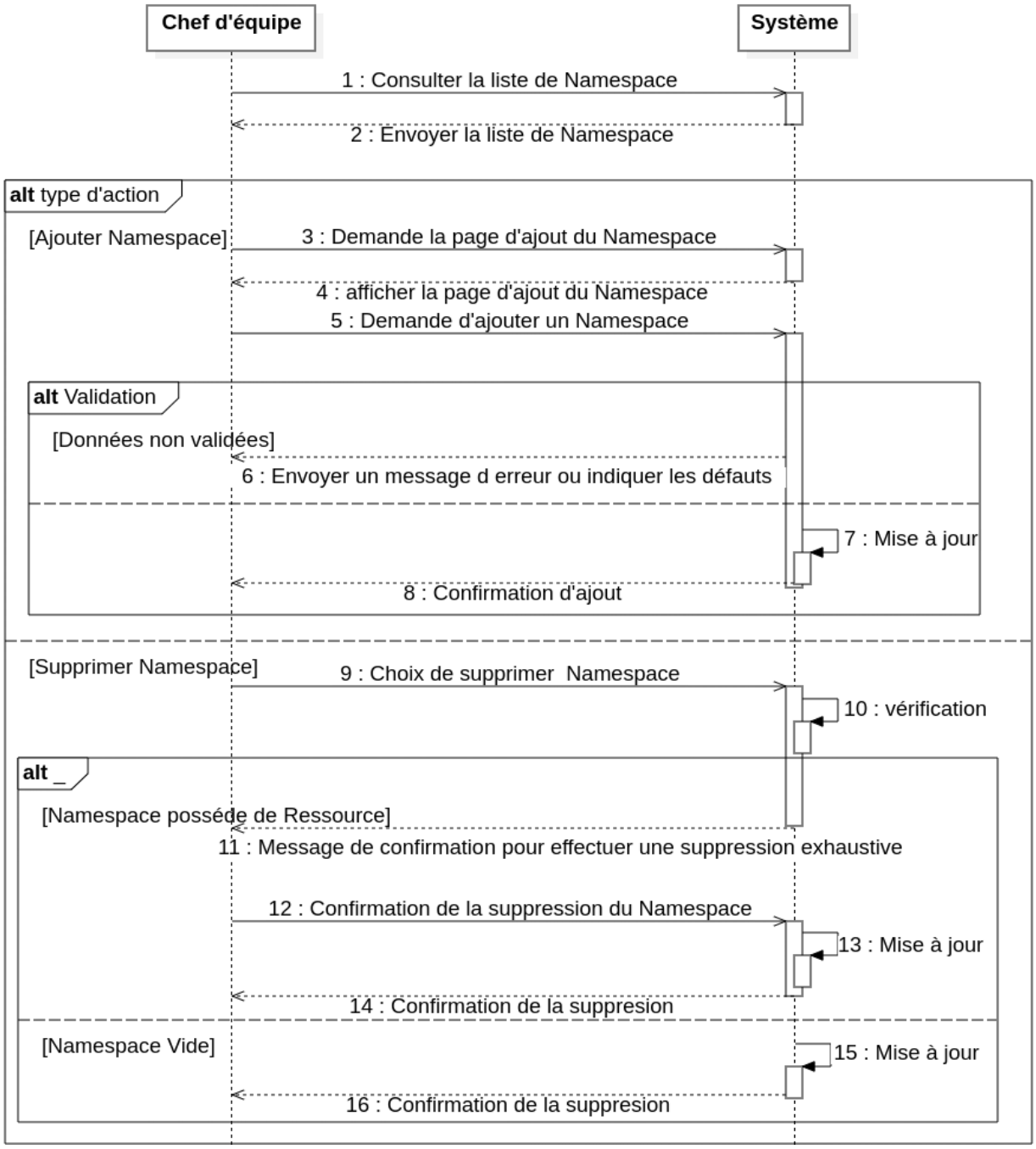

Figure 2.7: Diagramme de séquence « Gérer Namespace»

3.3.3. Diagramme de séquence « Gèrer Membres »

Le ***Tableau 2.3*** et la **Figure 2.8** présentent respectivement la description et la représentation du diagramme de séquence intitulé « Gérer Membres »

Tableau 2.3: Description du Cas d'Utilisation «Gérer Membres»

Cas d'utilisation	Gérer les Membres

Acteur	Chef d'équipe
Intérêt	Permet au Chef d'équipe de gérer les membres de son *Teamspace* : inviter des développeurs, attribuer ou modifier des rôles, et retirer des membres si nécessaire.
Précondition	Le Chef d'équipe doit être authentifié
Scénario nominal	- Le Chef d'équipe accède à l'interface de gestion des membres. - Le système affiche la liste des membres actuellement associés à l'équipe. ➢ Envoyer une invitation : • Le Chef d'équipe demande la liste des développeurs disponibles (non encore membres de l'équipe). • Le système affiche la liste des développeurs potentiels. • Le Chef d'équipe sélectionne un ou plusieurs développeurs à inviter. • Le système envoie une notification d'invitation aux développeurs sélectionnés ➢ Réception de la demande : • Le Chef d'équipe reçoit les demandes et peut les accepter ou les refuser. • Une fois acceptée, le développeur est ajouté à l'équipe. ➢ Gérer les rôles ou les membres : • Le Chef d'équipe peut modifier le rôle d'un membre existant. • Il peut également retirer un membre de l'équipe si nécessaire. • Le système met à jour automatiquement la liste des membres et les rôles.

Post Condition	Un membre est ajouté, supprimé, son rôle est modifié, ou une invitation a été envoyée avec succès.
Exception	• **Erreur de sélection :** Si le Chef d'équipe tente d'inviter un développeur déjà membre de l'équipe, une erreur est affichée • Ou si le chef accepte un Membre après que sa date soit expirée.

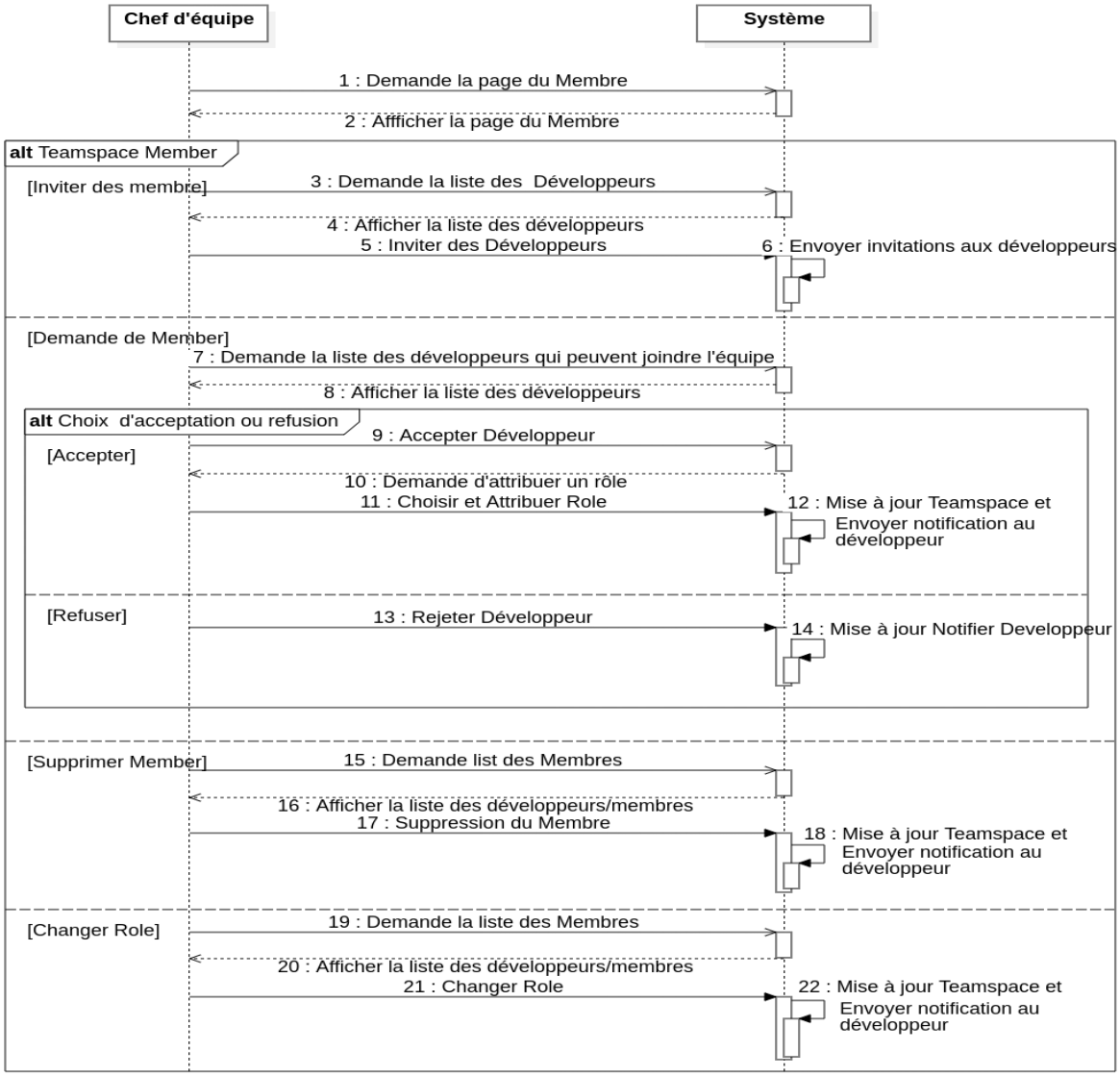

Figure 2.8: Diagramme de séquence « Gérer Membres»

4. Choix Technique

Pour le côté front-end, j'utiliserai la bibliothèque Angular, et le framework Express pour le côté back-end.

4.1. Framework Angular

Angular est un framework open source développé par Google depuis 2016. Conçu en TypeScript, il offre une solution complète pour le développement d'applications web robustes, modulaires et évolutives. Grâce à son architecture basée sur les composants et l'injection de dépendances, il se prête particulièrement bien à la création d'applications monopage (SPA). Le choix du framework Angular repose sur les raisons suivantes :

• **Structure complète** : Angular fournit une architecture claire et complète qui intègre tous les outils nécessaires au développement (routing, formulaires, requêtes HTTP, etc.), ce qui facilite la gestion de projets de grande envergure.

• **TypeScript** : En étant basé sur TypeScript, Angular permet un développement plus sûr, avec une meilleure gestion des types, une auto-complétion et une détection précoce des erreurs.

• **Performance et modularité** : Grâce à son système de modules et son compilation AOT (Ahead-Of-Time), Angular garantit de bonnes performances même sur des applications complexes.

4.2. Framework Express

Express est un framework web léger et flexible pour Node.js. Il a été créé en 2010 par TJ Holowaychuk et est largement utilisé pour développer des applications web et des API. Express facilite la gestion des routes, des requêtes HTTP et des middlewares dans une application Node.js, tout en laissant une grande liberté de structure au développeur.

Le choix du framework Express repose sur les raisons suivantes :

• **Légèreté et flexibilité** : Express n'impose pas de structure rigide, ce qui permet d'adapter l'architecture de l'application selon les besoins du projet.

• **Performance** : En étant construit sur Node.js, Express bénéficie de son modèle événementiel non-bloquant, ce qui le rend adapté aux applications nécessitant de hautes performances.

Conclusion

Dans ce chapitre, j'ai défini les acteurs ainsi que les besoins fonctionnels et non fonctionnels de l'application à l'aide des diagrammes de cas d'utilisation et des diagrammes de séquence acteur-système. J'ai également présenté la sélection technique des frameworks à utiliser. Le chapitre suivant sera dédié à la conception de l'application.

Chapitre 3. Conception

Introduction

Dans ce chapitre, j'aborde une phase essentielle du développement d'une application, qui fait le lien entre la spécification et la réalisation. Je commence par présenter l'architecture globale de l'application, avant de passer à la conception générale et détaillée, incluant les vues statiques à travers des diagrammes de classes. Enfin, je conclus ce chapitre en mettant en lumière certaines fonctionnalités de l'application à l'aide de maquettes.

1. Modèle architectural

Après avoir opté pour la méthodologie 2TUP, la démarche de conception s'aligne sur l'architecture de l'application. Avant de passer au développement, il est crucial de choisir un modèle de conception (design pattern). Parmi les patrons les plus couramment utilisés, on retrouve l'architecture en trois tiers (3-tiers) ainsi que le modèle Modèle-Vue-Contrôleur (MVC).

1.1. Architecture 3-tiers :

Ce modèle d'architecture, décrit par la **Figure 3.1**, se décompose en trois niveaux logiques bien distincts qui ont chacun un rôle bien défini :

- **Couche de présentation:** Elle représente l'interface par laquelle l'utilisateur interagit avec l'application. Elle transmet les requêtes de l'utilisateur à la couche de traitement, puis affiche les résultats générés. Elle constitue la partie visible et interactive du système, souvent désignée sous le terme interface homme machine (IHM).
- **Couche de traitement (Métier):** Elle correspond au cœur fonctionnel de l'application. Elle implémente la logique métier en définissant les opérations à effectuer sur les données, en réponse aux requêtes transmises par la couche de

présentation. C'est dans cette couche que sont appliquées les règles de gestion et les mécanismes de contrôle du système.

- **Couche Bases de données:** Cette couche est chargée de la gestion et de l'accès aux données utilisées par l'application. Les données peuvent être internes à l'application ou bien provenir d'un système externe. Quelle que soit leur origine, la couche de traitement y accède de manière uniforme, sans avoir à se préoccuper des spécificités de stockage ou de provenance, grâce à l'abstraction fournie par cette couche.

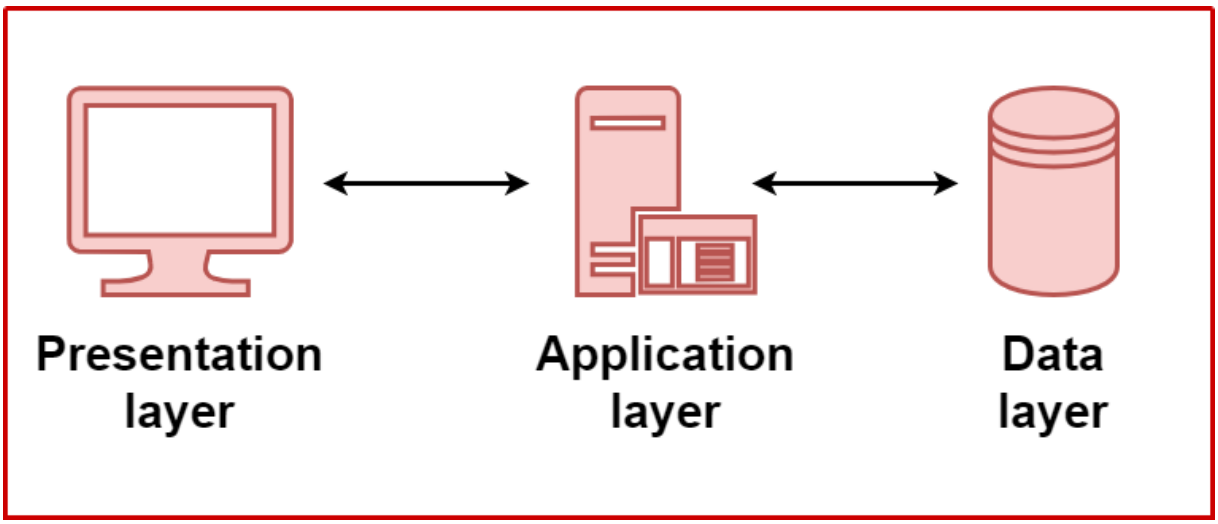

Figure 3.1: Architecture 3-tiers **[5]**

1.2. Modèle MVC

L'architecture **Modèle-Vue-Contrôleur (MVC)** est une méthode d'organisation de l'interface graphique d'un programme. Elle repose sur la séparation en trois entités distinctes : le modèle, la vue et le contrôleur, chacune ayant un rôle spécifique dans la gestion de l'interface. La **Figure 3.2** décrit l'architecture MVC.

- **Modèle:** Composant qui contient les données ainsi que la logique associée, telle que la validation, la lecture et l'enregistrement des données. Il peut s'agir d'une simple valeur ou d'une structure de données complexe. Le modèle représente le domaine fonctionnel dans lequel l'application évolue.
- **Vue**: Il s'agit de la partie visible de l'interface graphique. Elle utilise les données fournies par le modèle pour les afficher. La vue peut prendre la forme d'un formulaire, d'un tableau, de boutons ou de tout autre élément visuel. Elle contient également la logique nécessaire pour représenter correctement ces données à l'écran.

- **Contrôleur :** cette partie est chargée de la synchronisation du modèle et de la vue. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur demande au modèle les données, les analyser, prendre des décisions et finalement les déléguer à la vue.

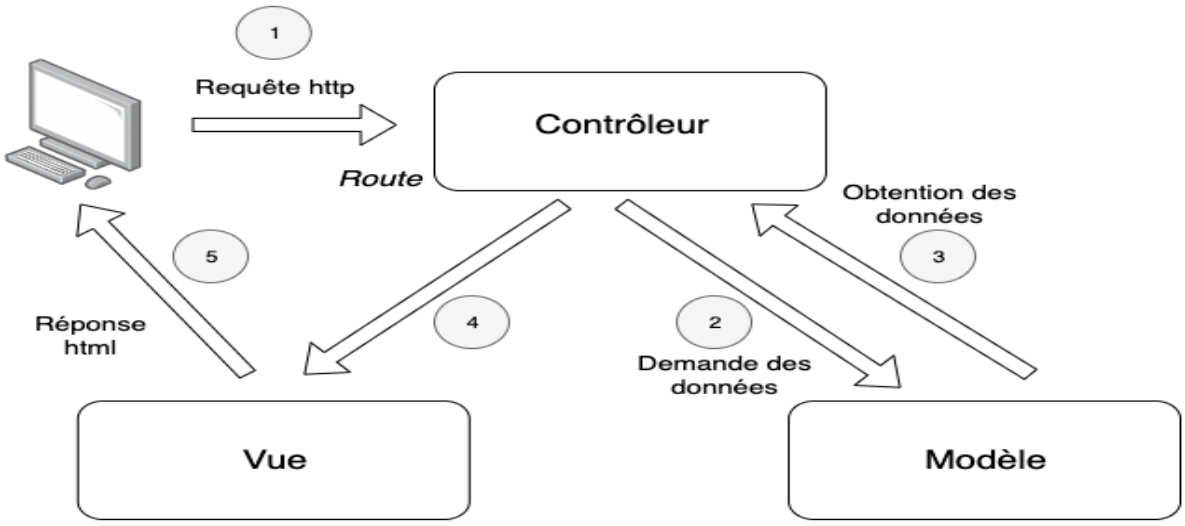

***Figure 3.2: Architecture du modèle MVC* [5]**

1.3. Architecture adoptée

L'architecture générale de cette application s'inspire de l'architecture 3-tier. Elle est illustrée par la **Figure 3.3**.

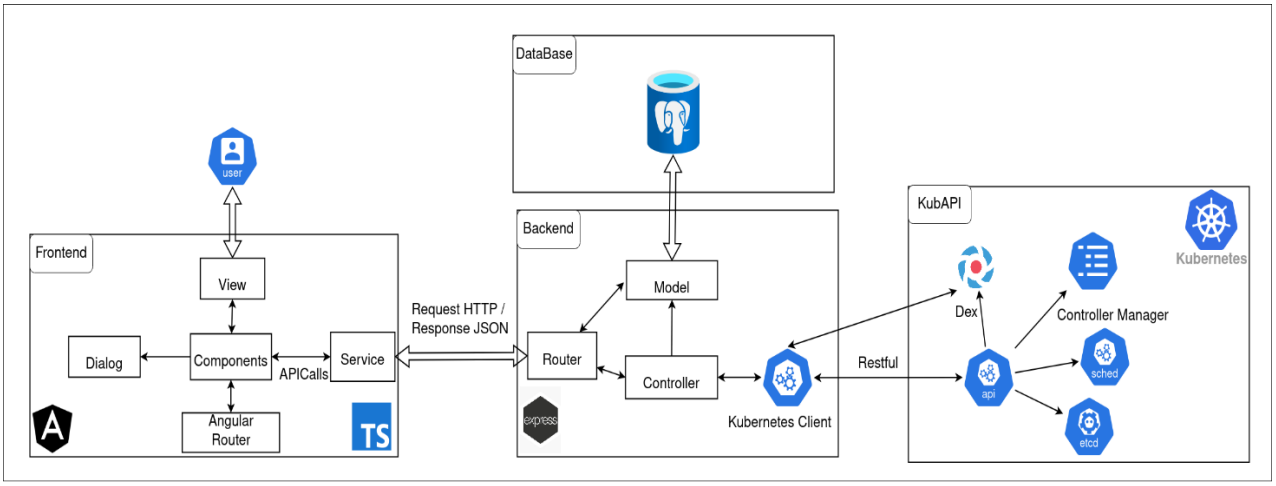

Figure 3.3 L'architecture de mon application

Le front-end et le back-end constituent deux piliers fondamentaux de l'architecture 3-tiers, chacun jouant un rôle complémentaire dans le fonctionnement global de l'application.

Le front-end englobe la partie interface de l'application et se divise en deux couches principales: la couche **Composant**, responsable de la représentation graphique des interfaces utilisateur ainsi que du traitement et de la validation des interactions avec l'interface ; et la couche **Service**, qui permet de structurer les traitements côté client et d'interagir avec le serveur, offrant ainsi une expérience utilisateur fluide et efficace.

D'un autre côté, le back-end prend en charge les aspects métier et l'accès aux données. Il se compose, dans sa première partie, de trois couches fondamentales : la couche **Routeur**, qui dirige les requêtes entrantes vers les contrôleurs appropriés ; la couche **Contrôleur**, chargée de recevoir les événements de l'utilisateur et de déclencher les actions nécessaires ; et la couche **Métier**, qui contient la logique applicative propre à l'application.

Le troisième tiers de l'architecture concerne l'accès aux données et l'interaction avec l'infrastructure. Il comprend deux couches : la couche **Kubernetes APIs**, responsable de l'interconnexion avec l'infrastructure Kubernetes. Elle permet de gérer les services et les ressources Kubernetes, d'orchestrer les déploiements, et d'administrer les droits d'accès des différents utilisateurs ; et la couche **Modèle**, qui représente les entités métier manipulées par l'application.

Cette structure assure une séparation claire des responsabilités, tout en garantissant une gestion efficace des processus métier, des données et de l'infrastructure.

2. Conception de base de données :

La conception d'une base de données vise à structurer les données de manière cohérente, en définissant la façon dont elles seront stockées, liées entre elles et exploitées. Elle commence par l'élaboration du **Modèle Conceptuel de Données (MCD)**, qui permet de représenter les entités, leurs attributs ainsi que les relations entre elles. Cette étape est suivie de la transformation du MCD en **Modèle Logique de Données (MLD)**, une représentation plus technique et plus proche de la structure finale de la base, prête à être traduite en schéma physique pour l'implémentation.

2.1. Modèle conceptuel de données (MCD)

Le Modèle Conceptuel de Données permet de représenter les différentes entités du système ainsi que les relations qui les unissent. Il constitue une étape essentielle dans la structuration

logique des informations. La ***Figure 3.4*** présente le MCD de notre base de données, tandis que le ***Tableau 3.1*** détaille les caractéristiques de chaque entité modélisée.

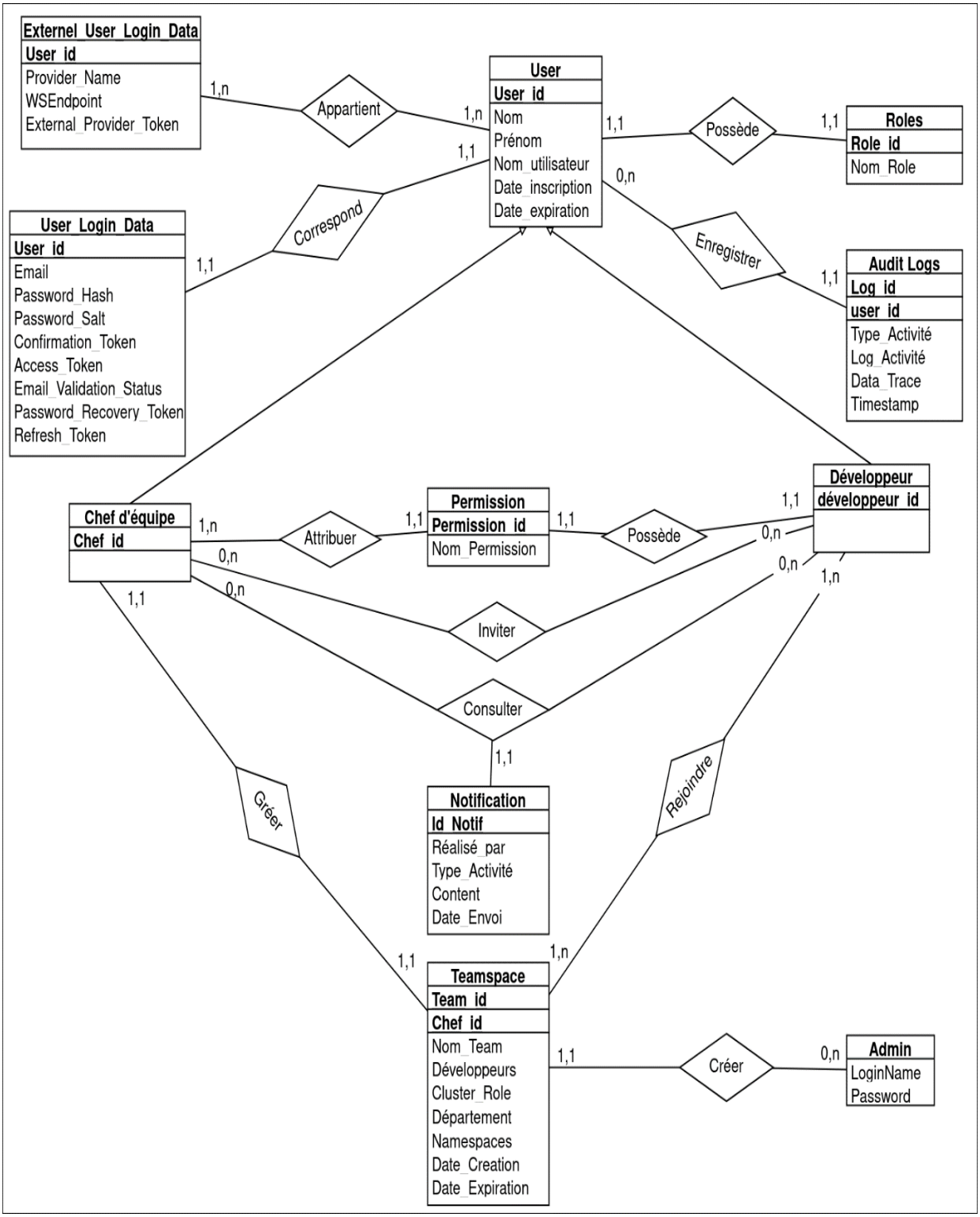

Figure 3.4: Modèle Conceptuel de données (MCD)

Tableau 3.1: Les entités du modèle conceptuel de données

Entités	**Description**	**Attributs**
User	Représente tout utilisateur de la plateforme.	User_Id : String Nom : String Prénom : String Nom_utilisateur : String Date_inscription : Date Date_expiration : Date
Chef d'équipe (hérite de User)	Utilisateur ayant le rôle de chef d'équipe.	Hérite de User. Pas d'attribut spécifique.
Développeur (hérite de User)	Utilisateur ayant le rôle de Développeur	Hérite de User. Pas d'attribut spécifique.
Teamspace	Représente une équipe DevOps sur la plateforme.	Team_Id : String Nom_Team : String Développeur : Array<Développeur> Cluster_Role : String Department : String Namespaces : Array<String> Date_Creation : Date Date_Expiration : Date
Notification	Message informatif envoyé à un utilisateur pour signaler une erreur, une action réussie ou un événement important.	Notification_Id : String Réalisé_par : String Type_Activité : String Content : String

		TeamId? : String(optional) Date_Envoi : Date
Audit Logs	Journal d'activité lié à un utilisateur, retraçant ses action sur les ressources Kubernetes.	Log_id : String Type_Activité : String Log_Activité : String Data_Trace : String Timestamp : Date
User_Login_Data	Contient les informations relatives à l'authentification et aux droits d'accès de l'utilisateur .	User_id : String Email : String Password_Hash : String Password_Salt : String Confirmation_Token : String Access_Token : String email_Validation_Status : boolean Password_Recovery_Token: String Refresh_Token : String
Rôle	Définit le rôle de l'utilisateur (développeur, chef d'équipe)	Role_id : String Nom_Role : String
Permission	Définit le type d'autorisation d'un utilisateur pour interagir avec les ressources Kubernetes.	Permission_id : String Nom_Permission : String
Admin	Entité unique représentant l'administrateur global de la plateforme.	LoginName : String Password : String

External_User_Login_Data	Décrit l'intégration avec un fournisseur d'accès externe (ex: OAuth, OpenId).	Provider_Name : String WSEndpoint : String External_Provider_Token : String

2.2. Modèle Logique de données (MLD)

Dans ce projet, le modèle MCD a été organisé en suivant le modèle relationnel, qui répond de manière adéquate aux besoins non fonctionnels. En appliquant les règles de transformation du modèle Entité/Association vers un modèle relationnel, le schéma relationnel suivant a été élaboré, comportant 9 relations.

Chef_ Équipe (chef_id, Nom, Prénom, Nom_utilisateur, Date_inscription, Date_expiration)

Développeur (developpeur_id, Nom, Prénom, Nom_utilisateur, Date_inscription, Date_expiration)

Teamspace (Team_Id, Nom_Team, Développeur, Cluster_Role, Département, Namespaces, Date_Creation, Date_Expiration, #chef_id)

Notification (Notification_id, Réalisé_par, Type_Activité, Content, Date_Envoi, #Team_Id?, #id_utilisateur)

AuditLogs (Log_Id, Type_Activité, Log_Activité, Data_Trace, Timestamp, #User_id)

User_Login_Data (User_id, Email, Password_Hash, Password_Salt, Confirmation_Token, Access_Token, email_Validation_Status, Password_Recovery_Token, Refresh_Token, #User_id)

Rôles (Role_Id, Nom_Role, #User_id)

Permission (Permission_Id, Nom_permission)

External_User_Login_Data (Provider__Name, WSEndpoint, External_Provider_Token, #User_id)

3. Conception logicielle détaillée

La conception logicielle détaillée a pour objectif de décrire précisément la structure et le comportement du système. Elle comprend une **vue statique**, représentée à l'aide d'un

diagramme de classes, illustrant les entités du système et leurs relations, ainsi qu'une **vue dynamique**, modélisée par un **diagramme de séquence**, mettant en évidence les interactions entre les objets au cours de l'exécution.

3.1. Vue statique : Diagramme de classe

Dans la section dédiée au modèle architectural adopté, les différentes couches logicielles de l'application ont été identifiées. La suite de cette partie vise à détailler les **composants logiciels** associés à chacune de ces couches. Comme le montre la **Figure 3.5**, les classes de l'application y sont organisées selon leur répartition par couche.

- Vue : cette couche comporte les composants « Components » qui correspondent aux interfaces graphiques par lesquelles l'utilisateur interagit avec le système et déclenche des évènements. Dans la couche Vue je définis les classes suivantes : « Teamspace », « User », « Notification » «Dashboard » « Job », « StatefulSet », « Pod », « Deployment », « DaemonSet », « ReplicationController », « CronJob », « PersistentVolume », « StorageClass », « PersistentVolumeClaim », « RessourceQuotas », « HorizontalPodAutoScalers », « ConfigMap », « Secret », « LimitRanges », « ServiceAccount », « Role », « RoleBinding », « ClusterRole », « ClusterRoleBinding », « NetworkPolicy », « IngressClass », « ReplicaSet », « Service », « Endpoint », « Ingress », « Nodes », « Namespace », « Event », « Helm », « Terminal », « Logs », « Metric » .
- Service : cette couche comporte les classes « Service » qui permettent de structurer les traitements et interagir avec le serveur. Dans la couche Service, je définis les classes suivantes : « TeamManagerService », « TeamService », « UserService » « PermissionService », « RoleService », « NotificationService », « ResourceEndpointService », « AuthorizationService », « ResourceService », « NamespacedResourceService ».
- Routing : La couche de routage, basée sur Express, permet de rediriger les requêtes HTTP vers les contrôleurs appropriés. Par exemple ; « Namespace.routing » oriente la requête vers le contrôleur « NamespaceControlleur ».
- Controller : cette couche contient les contrôleurs responsables de gérer les évènements générés par l'utilisateur et de lancer les actions correspondantes. Dans la couche Controller, je définis les classes suivantes : « TeamManagerController », « TeamController », « UserController » « PermissionController »,

« RoleController », « NotificationController », « ResourceEndpointController », « AuthorizationController », « ResourceController », « NamespacedResourceController ».

- La couche Modèle contient les entités métier utilisées par l'application. Chaque classe correspond soit à une entité stockée dans la base de données relationnelle, soit à une ressource kubernetes accessible via l'API.
  - Modèles internes à l'application : User, Notification, AuditLog, Role, Permission, Développeur, Chef_Équipe, Teamspace
  - Modèles Kubernetes (via API) : Ces ressources ne sont pas conservées dans la base de données, mais sont récupérées en temps réel par l'intermédiaire de l'API **Kubernetes (Kub-API)**. Pour simplifier la lecture du diagramme, elles sont regroupées sous un package nommé Kub-API.

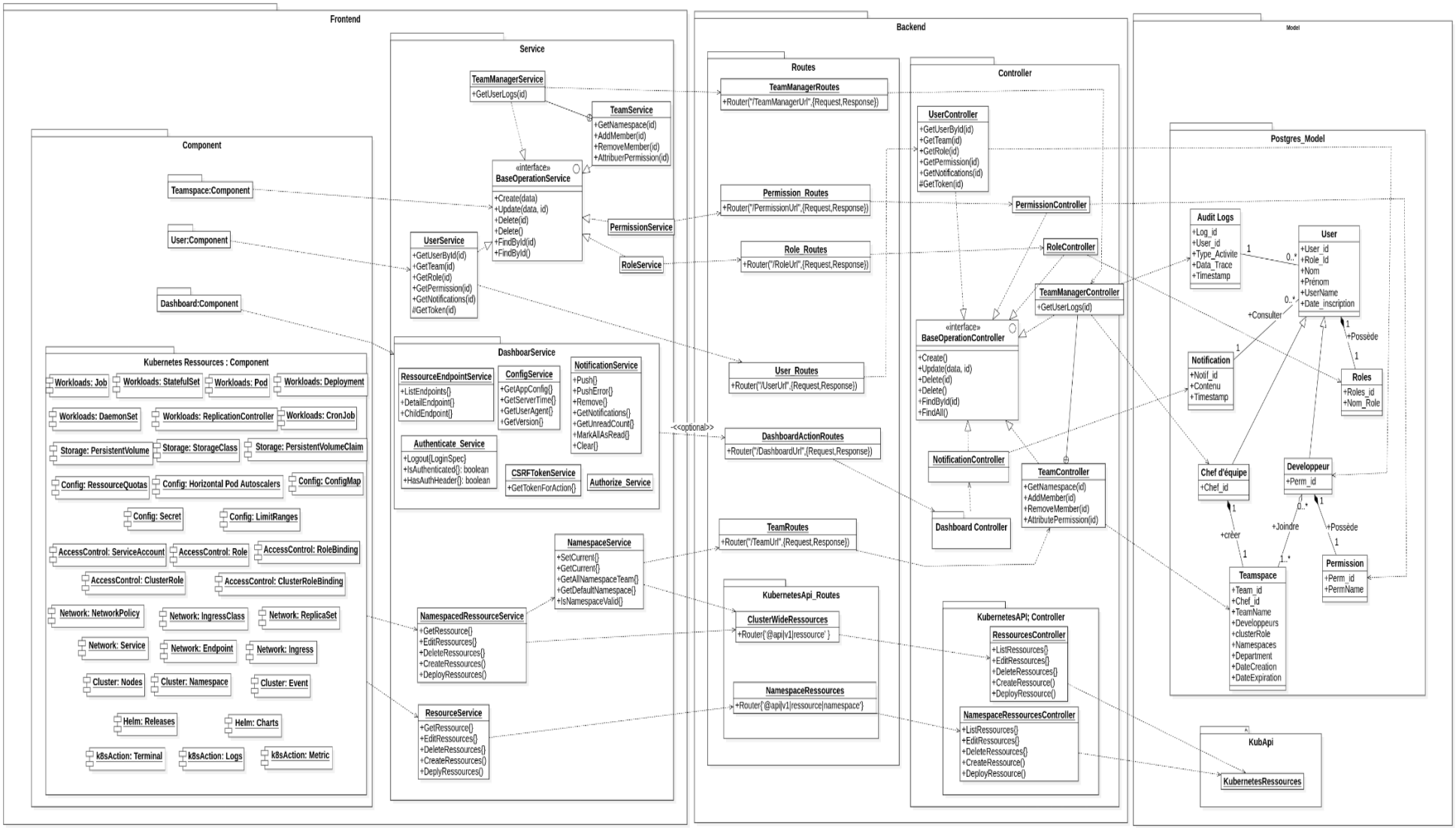

Figure 3.5: Diagramme de classe « Architecture 3-tier »

3.1.1. Diagramme de classe « kubernetes Ressources »

La **Figure 3.6** présente le contexte des ressources kubernetes à travers un diagramme de classe d'analyse, illustrant de manière claire la représentation et l'organisation des différentes données.

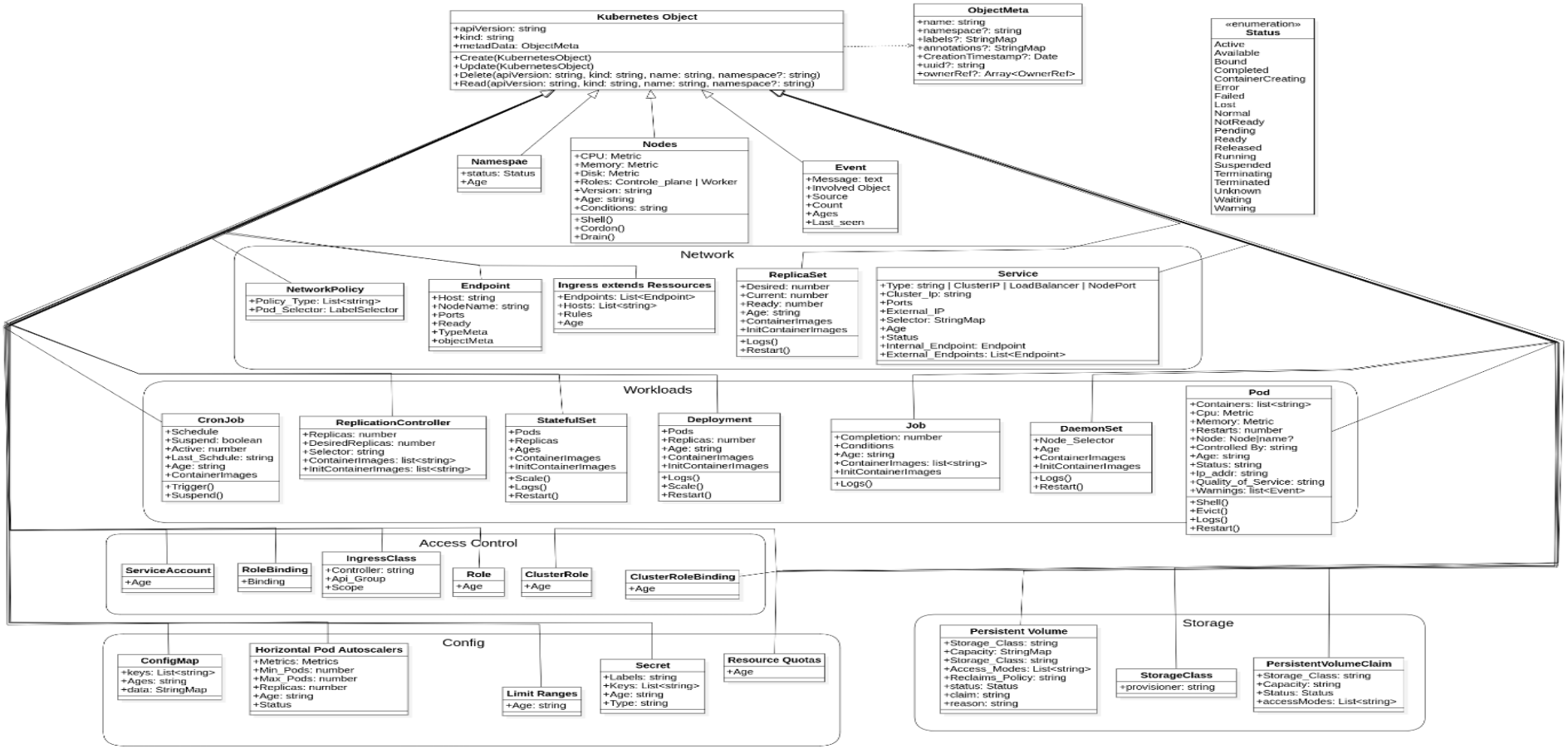

Figure 3.6: Diagramme du classe « Kubernetes Ressources »

3.2. Vue dynamique : Diagramme de séquence

Cette section présente le déroulement des traitements ainsi que les interactions entre les différentes couches de l'application, à travers l'utilisation de diagrammes de séquence. Ces diagrammes permettent de visualiser la chronologie des échanges entre les composants du système lors de l'exécution d'un scénario fonctionnel.

3.2.1. Diagramme de séquence général

La présente section offre une vue d'ensemble des interactions entre les différentes couches logicielles de l'application. Comme illustré dans la **Figure 3.7**, chaque composant joue un rôle précis : il intercepte une requête émise par l'utilisateur, puis invoque le service approprié afin d'établir une communication avec le back-end. Une fois cette connexion établie, le framework **Express** transmet la requête au contrôleur concerné. Ce dernier s'appuie sur la **couche Modèle** pour traiter la requête. Cette couche regroupe les entités métier de l'application, qu'il s'agisse de données stockées dans la base de données relationnelle ou de ressources Kubernetes récupérées dynamiquement via l'API Kubernetes.

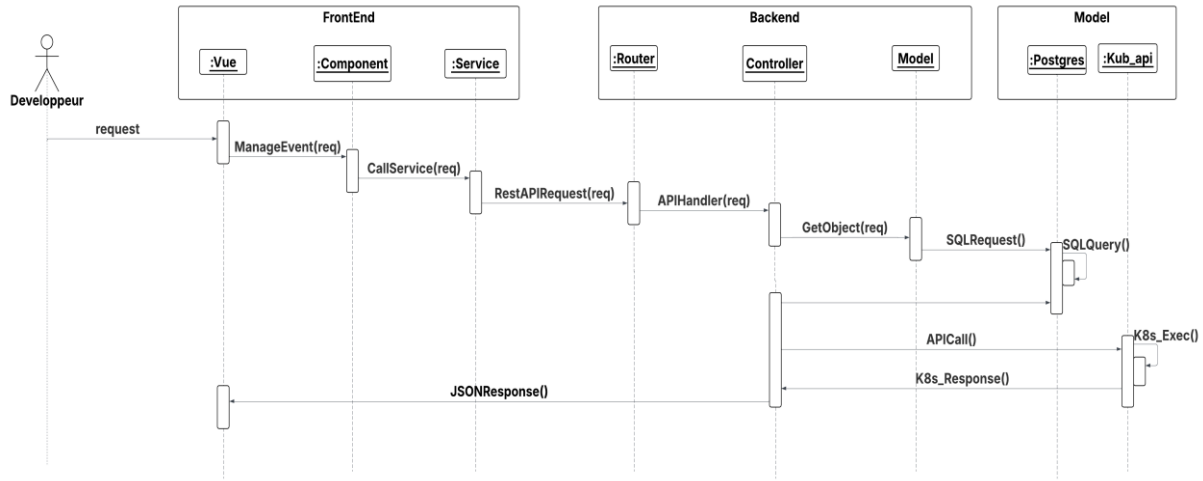

Figure 3.7: Diagramme de séquence général

3.2.2. Diagramme de séquence « Acceptation de la création du Team »

La **Figure 3.8** présente le diagramme de séquence détaillé illustrant le processus d'acceptation de la création d'un teamspace. Ce diagramme s'appuie sur le modèle d'interaction générale exposé dans la section précédente, « Diagramme de séquence général ». Côté front-end, le composant utilisé est le **TeamspaceComponent**, qui fait appel au service **TeamspaceService**.

Côté back-end, la requête est ensuite dirigée vers le contrôleur **TeamspaceController**, lequel interagit avec l'entité **Teamspace** pour finaliser l'opération.

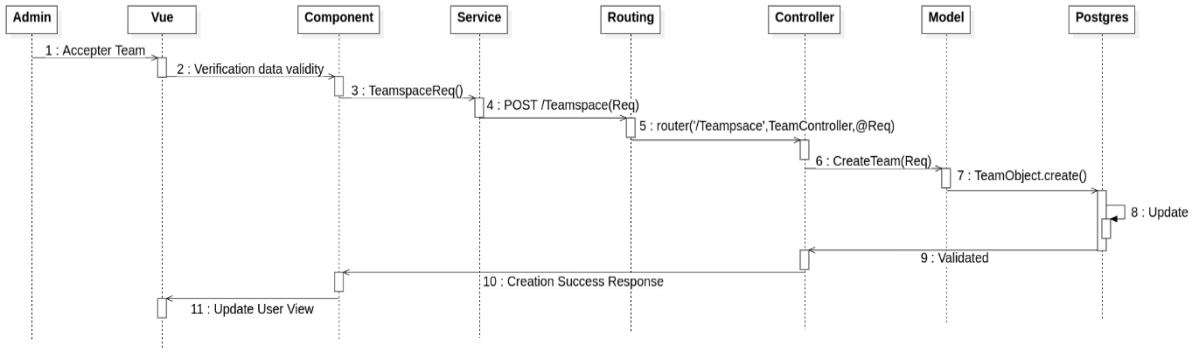

Figure 3.8: Diagramme de séquence « Acceptation de la création du Team »

3.2.3. Diagramme de séquence « Connexion à une application »

La **Figure 3.9** présente le diagramme de séquence détaillé illustrant la connexion et l'interaction avec une application en cours d'exécution dans Kubernetes. Ce diagramme suit le modèle d'interaction générale décrit dans la section précédente, « Diagramme de séquence général ». Côté front-end, le composant concerné est le **ShellComponent**, qui sollicite le service **ShellService**. Côté back-end, la requête est acheminée vers le contrôleur **ExecShellController**, lequel fait appel au module **WebSocket** ainsi qu'à l'API **kubernetes-client** pour assurer la communication avec l'application.

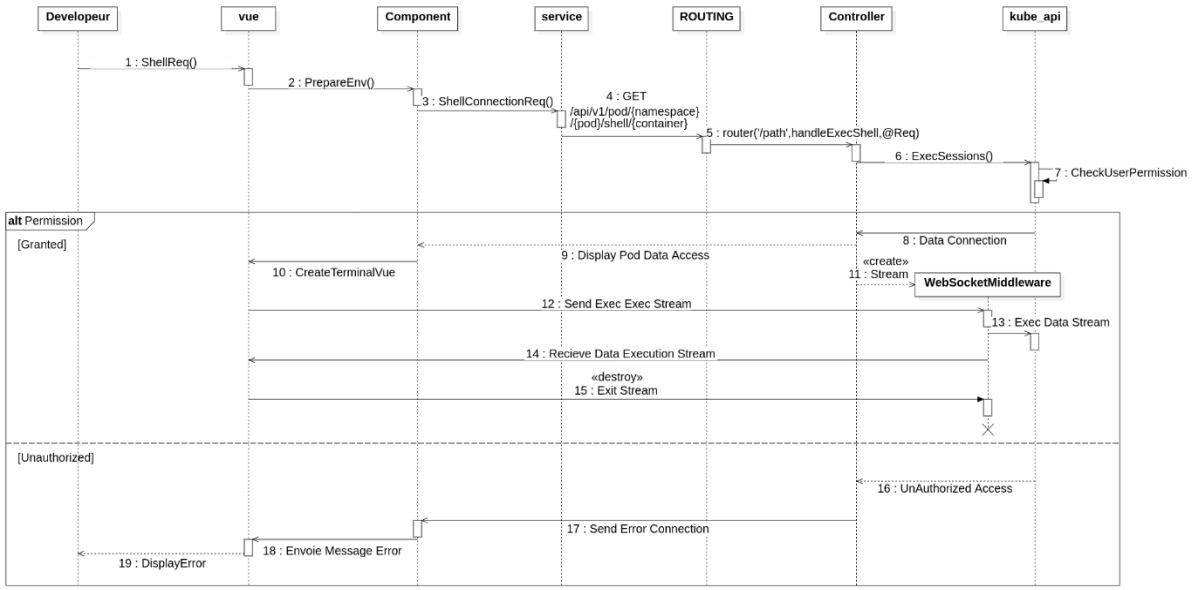

Figure 3.9: Diagramme de séquence « Connexion à une application »

3.2.4. Diagramme de séquence « Création d'une ressource kubernetes »

La **Figure 3.10** illustre le diagramme de séquence détaillé correspondant à la création d'une ressource Kubernetes. Ce diagramme suit le modèle d'interaction générale présenté dans la section « Diagramme de séquence général ». Côté front-end, le composant concerné est le **RessourceComponent**, qui fait appel au service **RessourceService**. Côté back-end, la requête est transmise au contrôleur **RessourceController**, lequel utilise l'API **kubernetes-client** pour effectuer la création de la ressource.

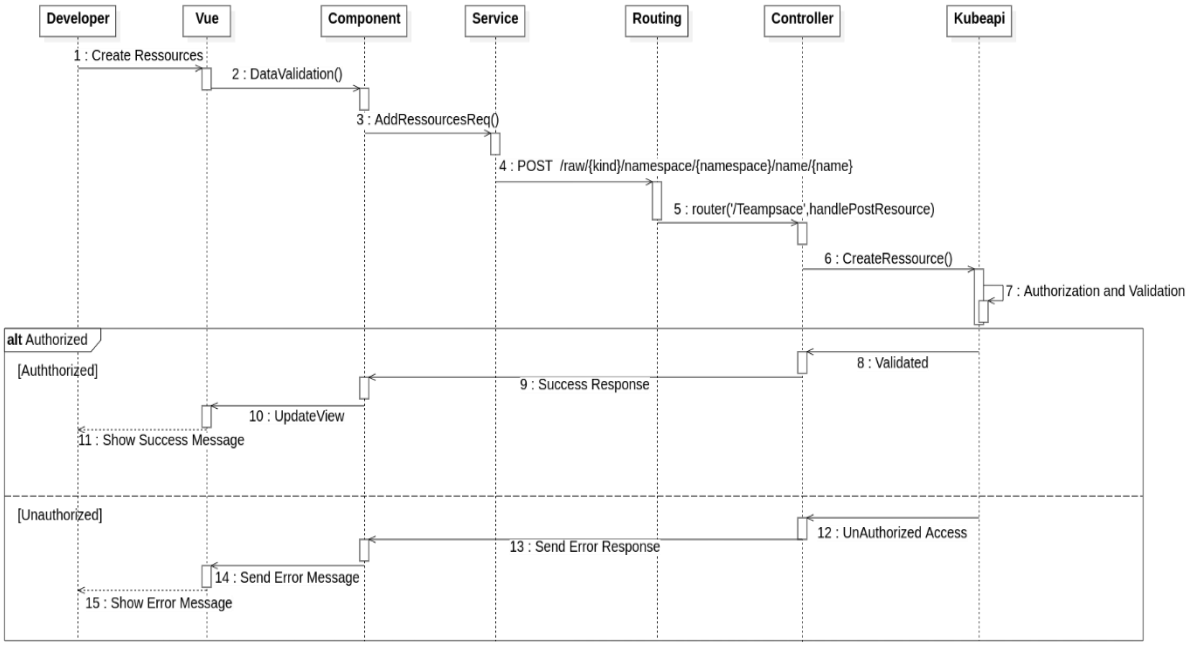

Figure 3.10: Diagramme de séquence « Création d'une ressource kubernetes »

3.3. Conception des interfaces graphiques

Cette section présente la conception des interfaces graphiques de l'application, en mettant en évidence l'organisation des vues et les liens de navigation entre elles.

3.3.1. Diagramme de navigation

Le diagramme de navigation sert à représenter les différentes interfaces utilisateur ainsi que les liens de navigation qui les relient. La **Figure 3.11** présente le diagramme de navigation de notre application, mettant en évidence les parcours possibles entre les écrans.

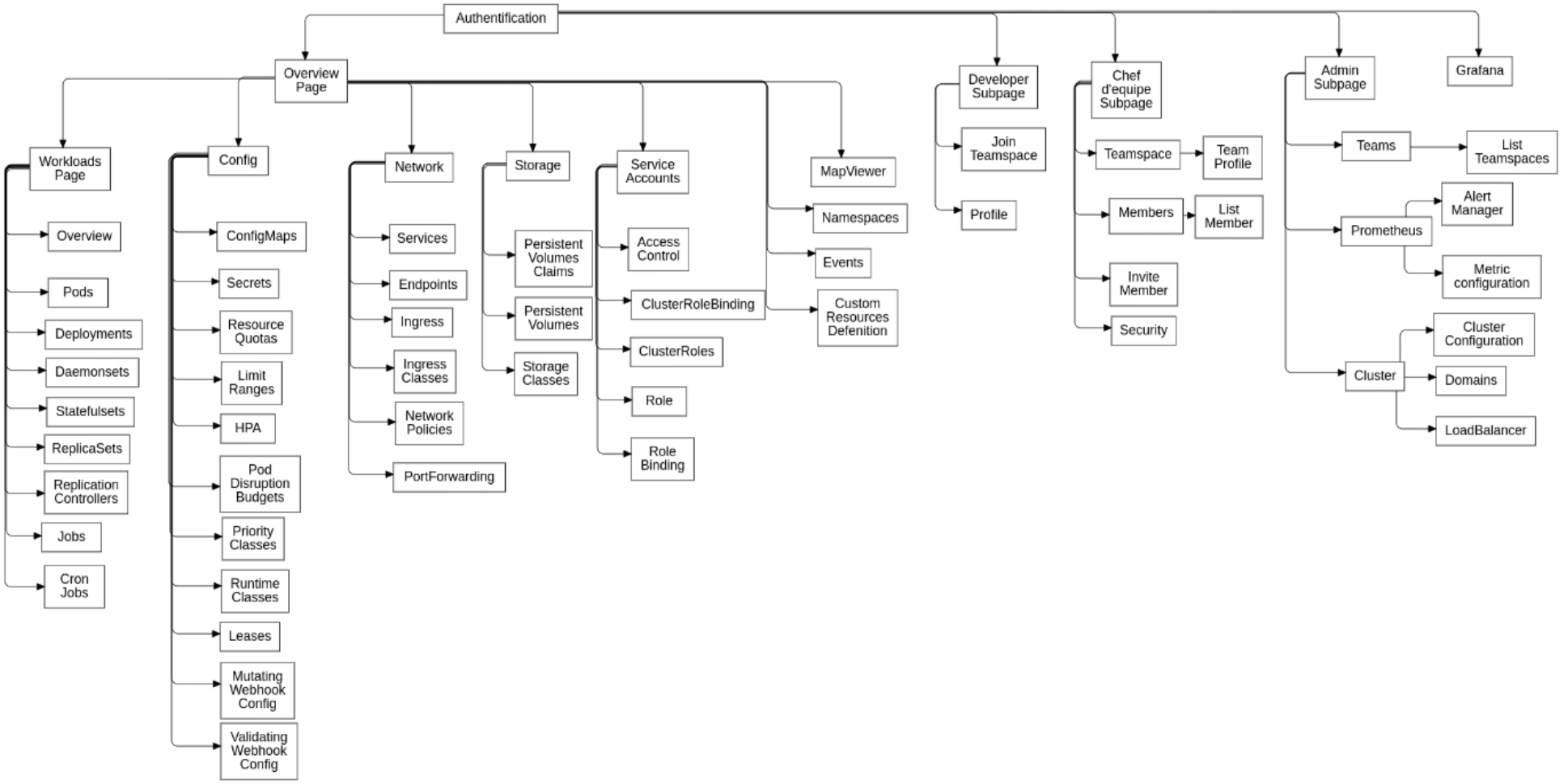

Figure 3.11: Diagramme de navigation

4. Conception Graphique

La phase de conception graphique joue un rôle essentiel dans le processus de développement, car elle permet de visualiser la structure et l'apparence des différentes interfaces de l'application avant leur implémentation. À travers des maquettes, elle offre une représentation concrète de l'expérience utilisateur attendue. Dans cette section, je présente quelques maquettes illustrant l'interface et la navigation prévues pour mon application.

4.1. Maquettage :

Le maquettage est une étape indispensable, car il permet de donner une vision claire du site et de ses différentes interfaces sous forme de schémas. Dans ce qui suit, je vais présenter quelques exemples de maquettes de mon application.

4.1.1. Interface du cas d'utilisation « Signup » de l'acteur « Développeur »

Lorsqu'un développeur s'inscrit sur la plateforme, il doit suivre un processus bien défini. Après avoir créé correctement son compte, il doit choisir de rejoindre une ou plusieurs équipes. Il peut également soumettre une demande de création d'une nouvelle équipe. Ensuite, il doit configurer son profil. Enfin, un formulaire récapitulatif permet de sauvegarder les actions effectuées au cours de l'inscription.

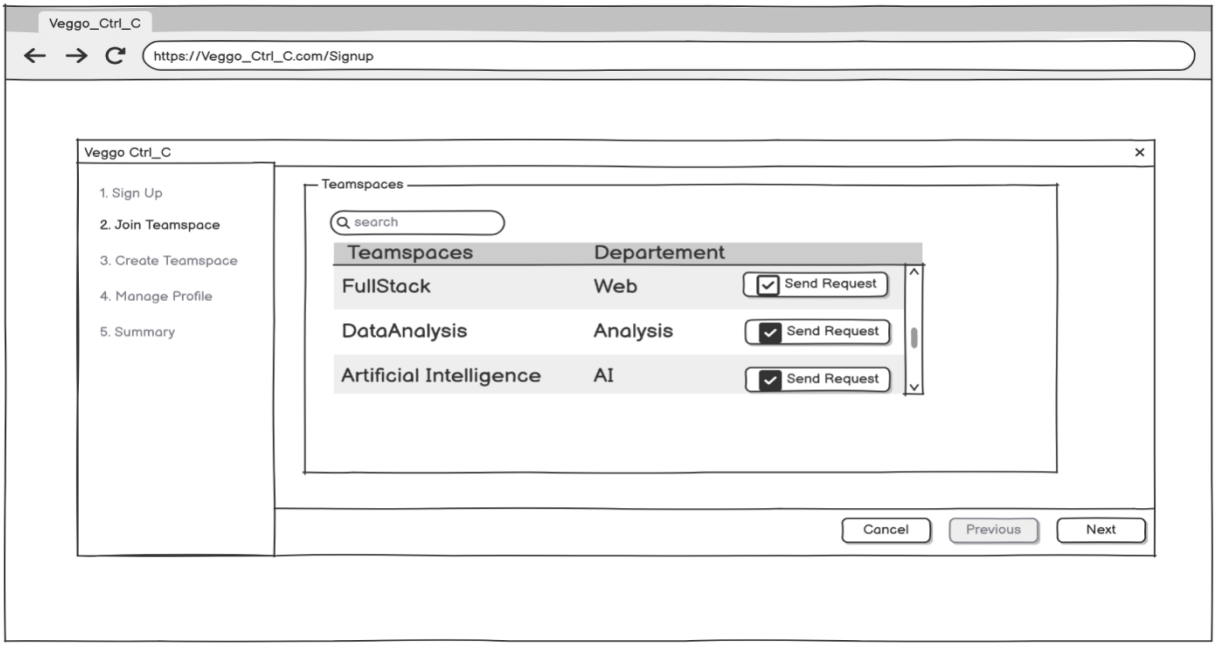

Figure 3.12: Interface « Signup »

4.1.2. Interface du cas d'utilisation « Gérer la demande du teamspace » de l'acteur « Admin »

Cette interface permet à l'administrateur de consulter, valider ou refuser les demandes de création de teamspaces soumises par les développeurs. Elle présente la liste des demandes en attente, accompagnée des informations nécessaires à la prise de décision, telles que le demandeur, la date de soumission et une description éventuelle. L'administrateur peut, pour chaque demande, visualiser les détails, approuver la création du teamspace ou rejeter la demande avec un motif. Cette interface joue un rôle clé dans le contrôle et la gestion des espaces collaboratifs au sein de la plateforme.

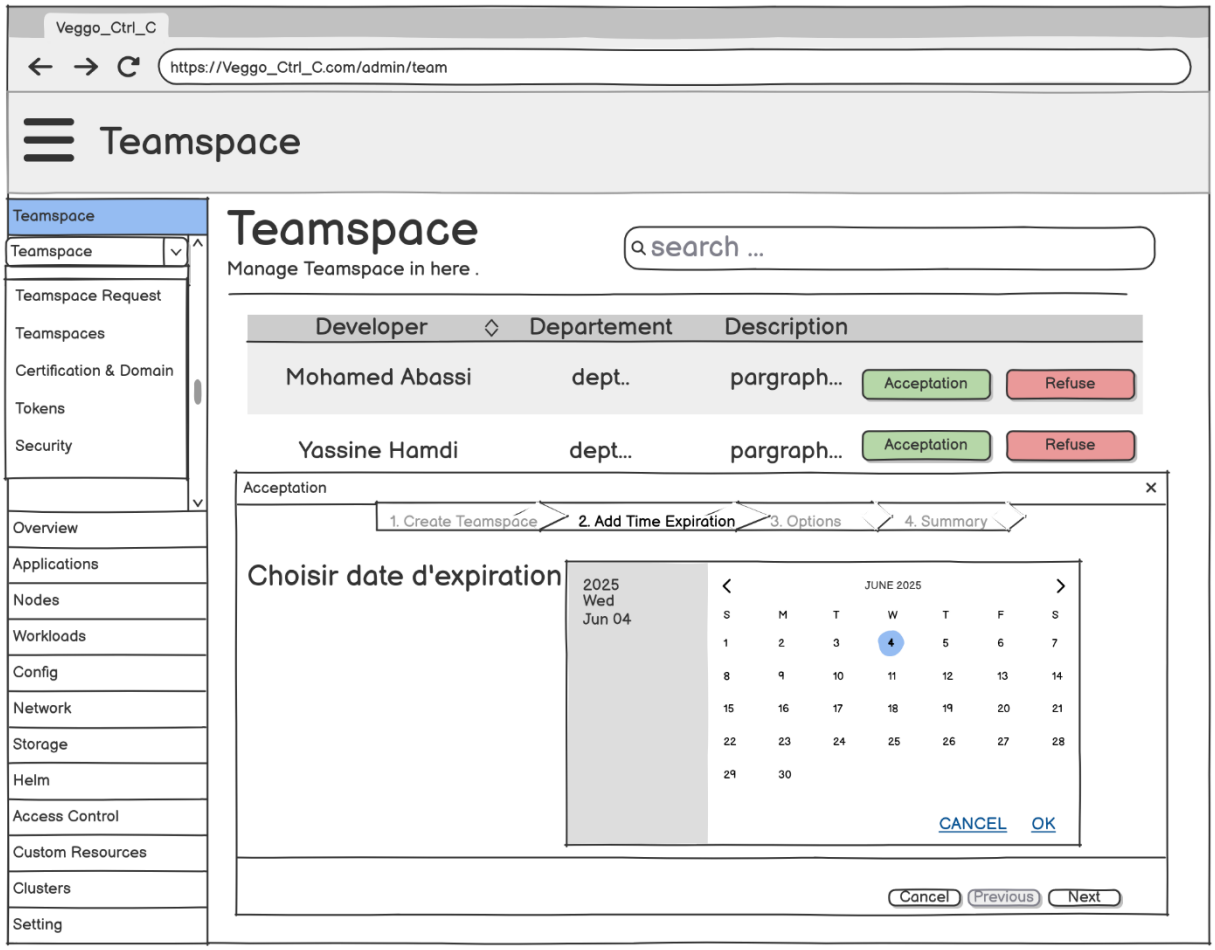

Figure 3.13: Interface « Gérer teamspace »

4.1.3. Interface du cas d'utilisation « Gérer Membre » de l'acteur « Chef d'équipe »

Cette interface permet au chef d'équipe de gérer les membres de son équipe. Il peut consulter la liste des membres actuels, ajouter de nouveaux membres, ou retirer des membres existants.

L'interface offre également la possibilité de modifier les rôles des membres au sein de l'équipe, afin d'adapter les responsabilités selon les besoins du projet. Enfin, le chef d'équipe peut envoyer des invitations aux développeurs.

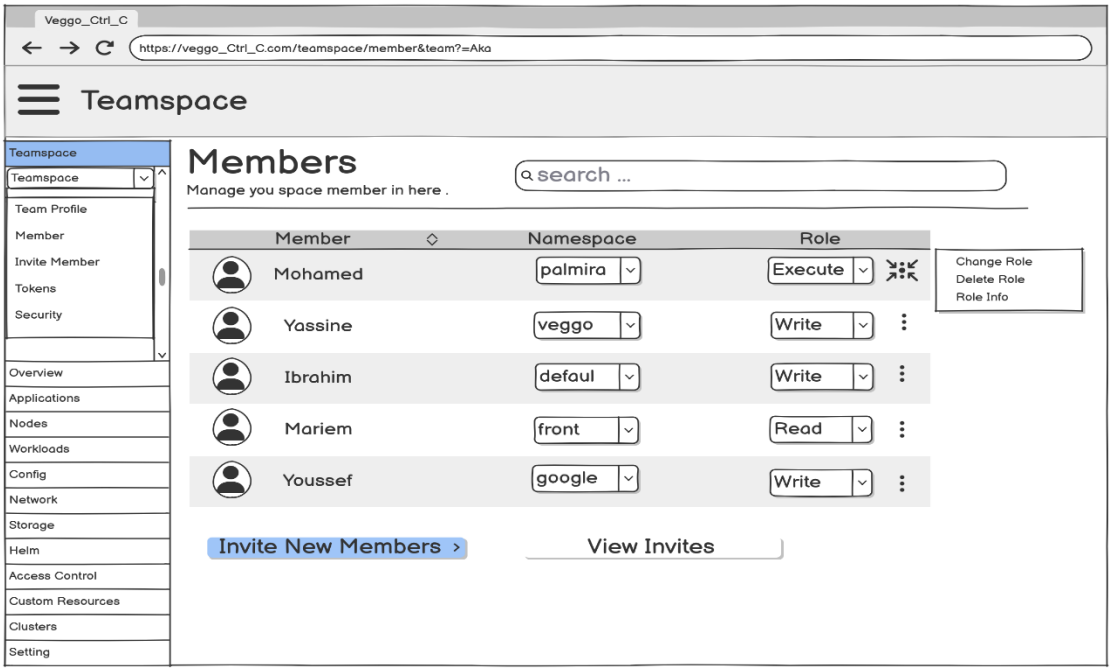

Figure 3.14 Interface « Gérer Membre »

Conclusion

Dans ce chapitre, les étapes clés de la conception du projet ont été exposées en détail. La présentation a commencé par l'architecture globale de l'application, avant d'aborder la conception de la base de données via le Modèle Conceptuel de Données (MCD) et sa conversion en modèle relationnel. La conception logicielle a ensuite été développée, avec une vue statique représentée par un diagramme de classes, complétée par des diagrammes de séquence illustrant la dynamique des interactions au sein du système. Pour conclure, la conception graphique des interfaces utilisateur a été présentée, mettant en lumière les différents liens de navigation entre les écrans.

Chapitre 4. Réalisation et implémentation

Introduction

Après avoir exprimé les différentes fonctionnalités envisagées par l'application, ainsi que sa conception, on va présenter dans ce chapitre la réalisation informatique de ses composantes. Il s'agit de la mise en œuvre des principales fonctions proposées pour tester le fonctionnement de l'application ainsi quelques interfaces de l'application.

1. Environnement de développement

Dans cette partie, je vous présente les différents outils matériels et logiciels utilisés pour l'implémentation de notre application.

1.1. Environnement Matériel

Pour le développement de cette application, on peut utiliser une machine qui a le minimum de caractéristiques 8 CPU, 8 Go de RAM et 100 Go ou plus d'espace disque (le processeur doit être compatible avec la virtualisation (VT-x ou AMD-V)).

1.2. Environnement Logiciel

Dans cette section, je vais détailler l'environnement logiciel ainsi que les outils que j'ai utilisés pour développer cette application.

1.2.1. Labo pour travailler avec kubernetes

Kubernetes est incontestablement devenu le standard en matière de déploiement d'applications cloud-native. Pour mettre en place un laboratoire expérimental et travailler avec Kubernetes sur des machines simples, comme un ordinateur personnel, j'ai préparé les outils suivants (d'autres outils existent également) : Docker, kubectl et Minikube.

1.2.2. Minikube

Minikube [6] est un outil destiné à exécuter un cluster kubernetes localement. Il permet aux développeurs de tester et d'explorer les fonctionnalités de kubernetes sans avoir à déployer une infrastructure complexe. En réalité, Minikube lance un cluster kubernetes a un seul nœud au sein d'une machine virtuelle (VM) directement sur l'ordinateur de l'utilisateur, mais il est possible d'augmenter le nombre de nœuds pour simuler un cluster multi-nœuds. Cela nécessite cependant une machine plus puissante, car chaque nœud consomme des ressources supplémentaires (CPU, mémoire, stockage). Il est compatible avec les systèmes Linux, Windows et macOS, et peut s'appuyer sur différents hyperviseurs pour créer sa VM, tels que VirtualBox, Docker, KVM (via QEMU), entre autres.

1.2.3. Docker

Docker [7] est une plateforme ouverte qui permet de développer, expédier et exécuter des applications dans des environnements isolés appelés conteneurs. Cette technologie offre la possibilité de séparer vos applications de votre infrastructure, ce qui vous permet de livrer des logiciels rapidement et de manière cohérente.

1.2.4. Kubectl

Kubectl [3] est le client CLI de Kubernetes. Il te permet de gérer les ressources Kubernetes, telles que les Pods, Deployments, Services, etc., directement depuis ton terminal.

1.2.5. Environnement et technologies de développement

Dans ce qui suit, on dresse la liste des logiciels et des outils qui ont été utilisés pour le développement et l'implémentation de l'application, ainsi que pour élaborer les diagrammes de ce rapport.

Tableau 4.1: Environnements et technologies de développement

Technologie	Description
neovim	**Neovim** [8] est un éditeur de texte avancé basé sur Vim, conçu pour être plus modulaire, extensible et moderne.

	**StarUML** [9] est un outil de modélisation qui permet de concevoir et de gérer les principaux types de diagrammes définis par la norme UML. Il offre également la possibilité d'ajouter des extensions, ce qui permet de prendre en charge des diagrammes avancés ou spécifiques, adaptés à des besoins plus complexes comme SysML, ERD, ou encore des diagrammes orientés vers des technologies particulières.
	**Balsamiq Mockups** [10] est un outil permettant de créer rapidement des maquettes d'interfaces utilisateur (IHM), facilitant la conception de prototypes avant le développement. Bien qu'il s'agisse d'un logiciel payant, il propose une période d'essai gratuite de 30 jours, ce qui permet de le tester librement avant de s'engager.
	**Dex** [11] est un service d'authentification open source qui sert de fournisseur d'identité **(IdP)** pour Kubernetes. Il fait office d'intermédiaire en gérant l'authentification des utilisateurs, leur permettant ainsi d'accéder à un cluster kubernetes via des protocoles standards comme **OAuth 2.0** et **OpenID Connect (OIDC).**
	**OAuth 2.0** [12] est un protocole d'autorisation qui permet à une application d'accéder de manière limitée aux ressources d'un utilisateur hébergé sur un autre service, sans nécessiter la transmission des identifiants de cet utilisateur. Dans le contexte de kubernetes, Oauth 2.0 facilite la gestion sécurisée des accès aux API en déléguant l'authentification à un fournisseur tiers, tel que Dex, qui agit comme intermédiaire pour gérer les identités et les jetons d'accès.
	**Helm** [13] est le gestionnaire de paquets officiel pour **Kubernetes**, qui facilite le déploiement, la configuration et la gestion d'applications conteneurisées sur un cluster Kubernetes. Helm utilise des paquets appelés **charts**, qui contiennent toutes les ressources Kubernetes nécessaires (déploiements, services, configurations, etc.) pour installer une application.

	**Grafana** [14] est une plateforme open source dédiée à la visualisation et à la supervision des données. Elle est principalement utilisée pour surveiller les systèmes, applications et infrastructures cloud. Grafana permet de connecter diverses sources de données telles que **Prometheus**, **InfluxDB**, **Elasticsearch** ou des **bases SQL** afin de créer des tableaux de bord interactifs affichant en temps réel des métriques sous forme de graphiques, jauges, alertes, et plus encore.
	**Prometheus** [15] est un système open source de **monitoring** et d'**alerte** conçu pour collecter, stocker et analyser des **métriques temporelles** issues d'applications, de services ou d'infrastructures. Développé à l'origine par SoundCloud, Prometheus est désormais un projet de la **Cloud Native Computing Foundation (CNCF)**, tout comme Kubernetes.
	**Trivy** [16] est un **scanner de sécurité open source** développé par **Aqua Security**, utilisé pour détecter les **vulnérabilités** dans les **images de conteneurs**, mais aussi dans les **dépendances applicatives**, les **fichiers de configuration** (comme Dockerfile ou Kubernetes YAML), et les **infrastructures-as-code**.
	Le **JavaScript Kubernetes Client** [17] est une bibliothèque officielle qui permet aux développeurs Node.js d'interagir directement avec l'API kubernetes en utilisant du code JavaScript ou TypeScript. Elle offre une interface programmée pour accéder, gérer et surveiller les ressources kubernetes.
	**Express TS** [18] fait référence à l'utilisation du framework web Express.js, l'un des plus populaires dans l'écosystème Node.js, en combinaison avec le langage TyepScript. Cette approche permet de conserver la simplicité et la légèreté d'Express, tout en profitant des avantages du typage statique de TypeScript, tels que la sécurité, la robustesse et une meilleure lisibilité du code.

	**Angular** [19] est un framework web open source développé par Google, destiné à la création d'applications web dynamiques côté front-end. Construit sur TypeScript, il permet de développer des interfaces utilisateur riches, performantes et bien organisées, en s'appuyant sur des concepts modernes comme l'architecture par composants, l'injection de dépendances et la liaison de données bidirectionnelle (data binding).
	**Swagger** [20] est un ensemble d'outils open source conçu pour décrire, documenter, tester et consommer des API REST. Il s'appuie sur une spécification standard appelée **OpenAPI Specification (OAS)**, qui permet de définir avec précision les points de terminaison (endpoints), les méthodes, les paramètres, les types de données, les réponse attendues ainsi que les mécanismes d'authentification d'une API.
	**Postman** [21] est un outil permettant de tester des API afin de s'assurer qu'elles répondent aux attentes en termes de fonctionnalité, de fiabilité, de performance et de sécurité. Au-delà du simple test, Postman propose des fonctionnalités de collaboration, facilitant le partage de collection d'API, la gestion d'environnements communs et le travail en équipe sur le développement, la documentation et le débogage des API de façon centralisée et efficace.
	**Minikube** [6] est un outil open source qui permet de déployer un cluster kubernetes local sur une seule machine, facilitant ainsi le développement , les test et l'apprentissage de kubernetes dans un environnement isolé.
	**Docker** [7] est une plateforme open source qui permet de créer, déployer et exécuter des applications au sein de conteneurs. Un conteneur Docker est une unité légère, portable et isolée qui embarque une application ainsi que toutes ses dépendances, assurant ainsi un fonctionnement cohérent quel que soit l'environnement d'exécution.

2. Bibliothèque Angular

L'application cliente est développée à l'aide du framework Angular, qui permet de concevoir des interfaces utilisateur dynamiques et de structurer des applications web mono-page à travers un système de composants interconnectés.

Pour démarrer un projet Angular, il faut d'abord installer Node.js, l'environnement d'exécution JavaScript, en le téléchargeant depuis son site officiel. Ensuite, l'**Angular CLI** (outil en ligne de commande ng) doit être installée via npm, le gestionnaire de paquets de Node.js à l'aide de commande suivant [19]:

```
npm install -g @angular/cli
```

Une fois l'environnement correctement installé, il est possible de démarrer la création de l'application. Pour cela, il suffit d'exécuter la commande suivante dans le terminal afin de générer un nouveau projet Angular :

```
ng new Veggo-Ctrl-C
```

Ensuite, une fois l'installation est terminée, on peut ouvrir le dossier de projet :

```
cd Veggo-Ctrl-C
```

Et lancer le serveur :

```
ng serve
```

Il n'existe pas de structure de projet imposée par défaut, mais l'approche la plus couramment adoptée consiste à organiser les fichiers SCSS, TypeScript(TS), HTML et les fichiers de test dans un même dossier, regroupés par fonctionnalité. Cette organisation modulaire est illustrée dans la **Figure 4.1**.

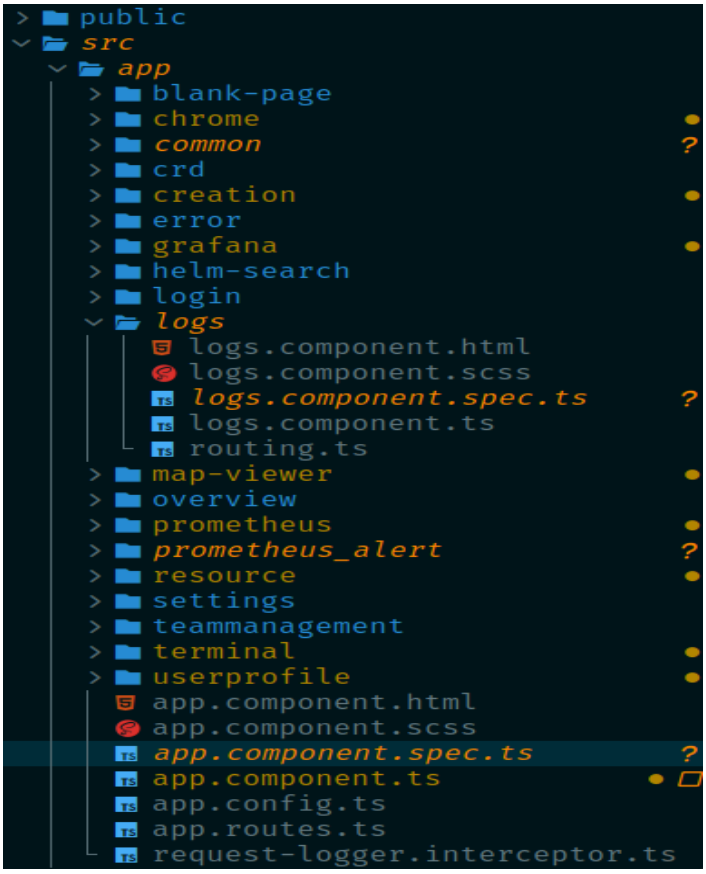

Figure 4.1: Structure de composant angular

2.1. Cycle de vie d'un composant Angular

Le cycle de vie d'un composant Angular désigne l'ensemble des étapes que traverse un composant, depuis sa création jusqu'à sa destruction. Angular propose plusieurs méthodes de cycle de vie qui ne servent de points d'ancrage permettant d'exécuter du code a des moments précis du cycle de vie de composant.

Voici un aperçu des principales étapes de ce cycle :

- **Création du composant :** le composant est instancié par Angular.
- **Injection des dépendances :** les services et autres dépendances nécessaires sont injecté dans le composant.
- **Réception des données d'entrée :** le composant parent transmet des données au composant enfant via les propriétés d'entrée.
- **Initialisation des données d'entrée :** le composant enfant initialise ses variables à partir des données reçues.

- **Compilation et rendu du template :** le template associé au composant est compilé et affiché dans le DOM.
- **Exécution active :** le composant est pleinement opérationnel et peut interagir avec l'utilisateur et le reste de l'application.
- **Gestion des interactions et nettoyage :** lorsque le composant est détruit, il effectue le nettoyage des ressources utilisées ; comme la désinscription des observables ou la libération d'autre ressource.

3. Framework Express

Express est un framework web simple et puissant pour Node.js, conçu pour simplifier la création de serveurs et d'API. Grâce à sa gestion efficace des routes, middlewares et requêtes HTTP, il permet de développer rapidement des applications web modulaires et performantes.

3.1. Express avec TypeScript

Par défaut, Express [22] est utilisé avec JavaScript. Cependant, afin de bénéficier d'une meilleure vérification des types et de faciliter la maintenance, notamment pour l'intégration avec l'API Kubernetes, j'ai décidé de migrer le projet vers TypeScript. Cette transition nécessite la configuration du compilateur TypeScript (tsc).

Pour créer un projet Express en TypeScript, nous commençons par créer un dossier dédié, puis nous nous y déplaçons en tant que répertoire courant avant d'exécuter la commande suivante pour initialiser un projet npm :

```
npm init -y
```

Ensuite, nous installons le framework Express avec la commande :

```
npm install express
```

Enfin, nous ajoutons et configurons les librairies nécessaires pour TypeScript et le développement avec les commandes suivantes :

```
npm install --save-dev typescript ts-node @types/express
@types/node nodemon
```

Après avoir installé les dépendances, il faut initialiser le fichier de configuration TypeScript en exécutant :

```
npx tsc --init
```

Cela génère un fichier `tsconfig.json` à la racine du projet.

On peut organiser le projet sous la structure suivante :

```
/src
  ├── index.ts
  ├── routes/
  ├── controllers/
  ├── models/
  └── middlewares/
```

4. Travail réalisé

Dans cette section, je vous présente quelques interfaces graphiques de l'application, illustrant les différents cas d'utilisation.

4.1. Interface « Signup »

Initialement, le développeur doit créer un compte en suivant les différentes étapes d'inscription, afin d'être intégré dans un espace d'équipe (*teamspace*) ou d'en créer un nouveau.

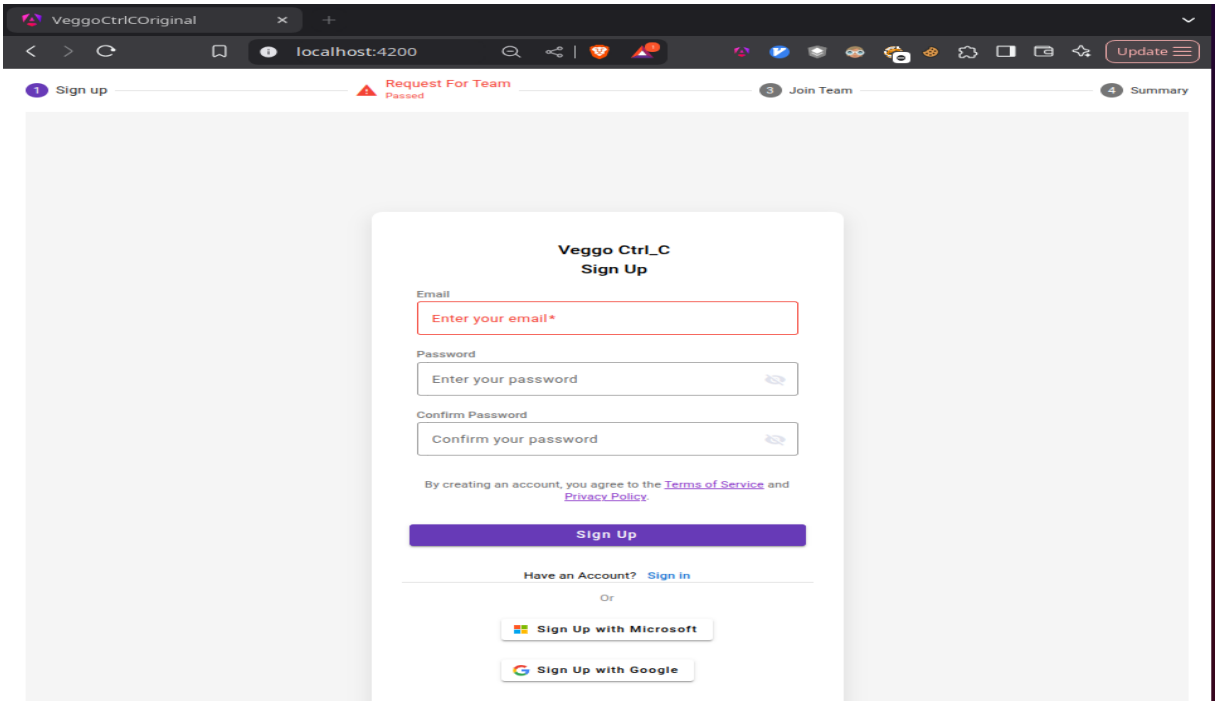

Figure 4.2: Interface « Signup »

4.2. Interface « Connexion au système »

Cette interface a pour objectif de permettre à l'administrateur de se connecter au système déployant l'infrastructure Kubernetes. Elle offre également un accès direct au système d'exploitation, facilitant ainsi l'interconnexion et la gestion avancée de Kubernetes, pour les fonctionnalités non prises en charge par notre interface.

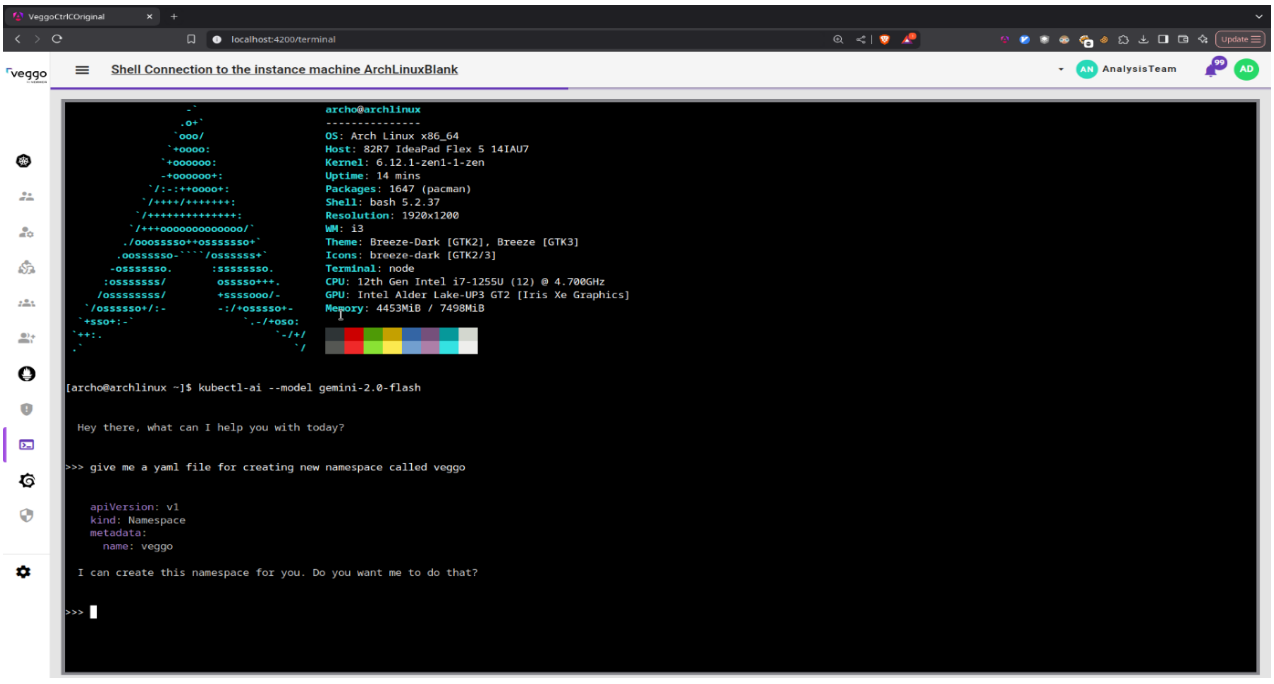

Figure 4.3: Interface « Connexion au système »

4.3. Interface « Alert Manager »

Cette interface permet à l'administrateur de définir des conditions d'alerte afin que certaines ressources ne dépassent pas les seuils prédéfinis.

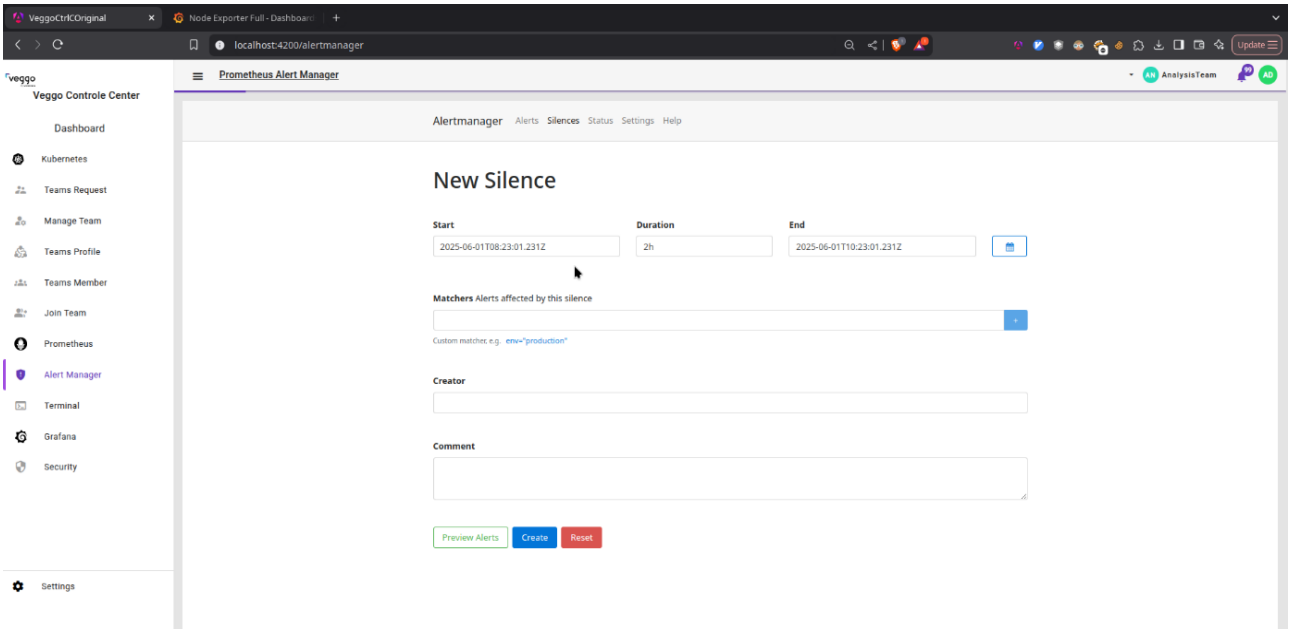

Figure 4.4: Interface « Alert Manager »

4.4. Interface « Prometheus Server »

Cette interface permet à l'administrateur de gérer les métriques de Kubernetes afin qu'elles puissent être analysées par le serveur Prometheus. Elle offre également d'autres fonctionnalités telles que l'exécution de requêtes sur les ressources Kubernetes, la visualisation des métriques et la gestion de diverses configurations.

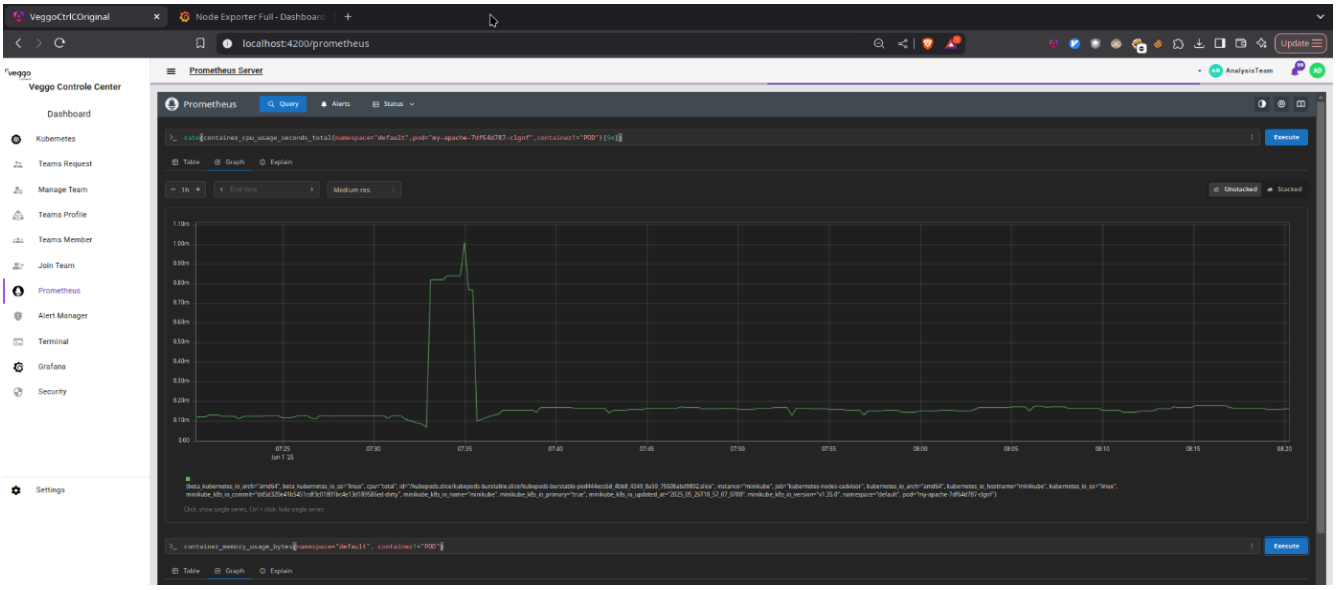

Figure 4.5: Interface « Prometheus Server »

4.5. Interface « Grafana »

Cette interface a pour objectif de visualiser les métriques récupérées par Prometheus sous forme de tableaux de bord de surveillance (*monitoring dashboards*), pouvant être créés ou importés. Grafana permet en outre de structurer les utilisateurs en équipes et de gérer leurs rôles respectifs. De plus, il est possible de configurer notre application pour qu'elle s'intègre à Grafana, évitant ainsi de recréer manuellement les équipes et les membres, tout en conservant leurs rôles et autorisations.

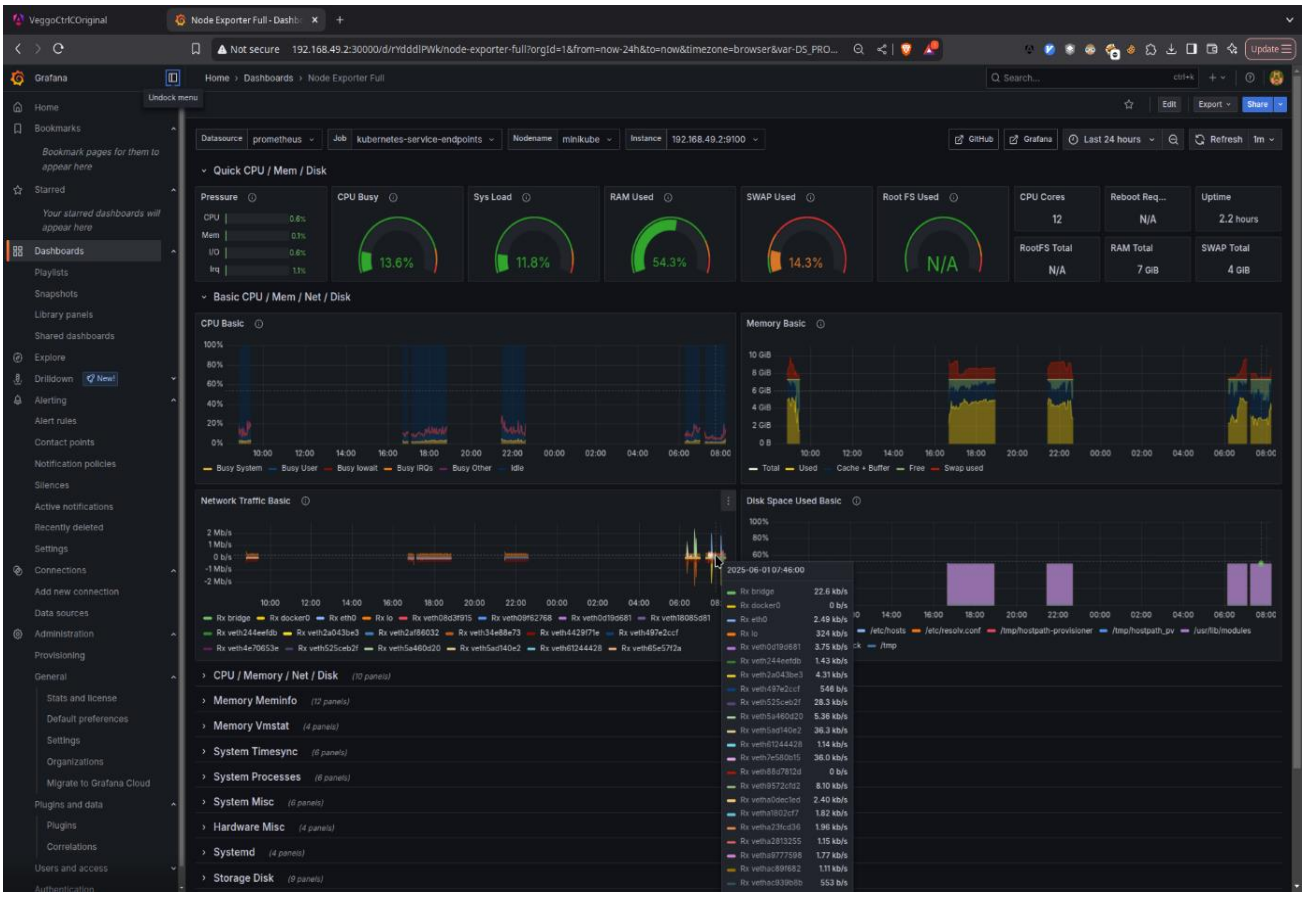

Figure 4.6: Interface « Grafana »

4.6. Interface « Map Viewer »

Cette interface permet au chef d'équipe de visualiser l'état de l'interconnexion des ressources Kubernetes dans les différents *namespaces*. Les icônes affichées permettent d'interpréter rapidement l'état de chaque ressource : par exemple, la couleur verte indique que la ressource fonctionne correctement, le rouge signale que l'application est arrêtée, et le jaune signifie que la ressource est suspendue. De plus, les icônes utilisées sont issues des standards officiels de Kubernetes, ce qui facilite l'identification visuelle de chaque type de ressource.

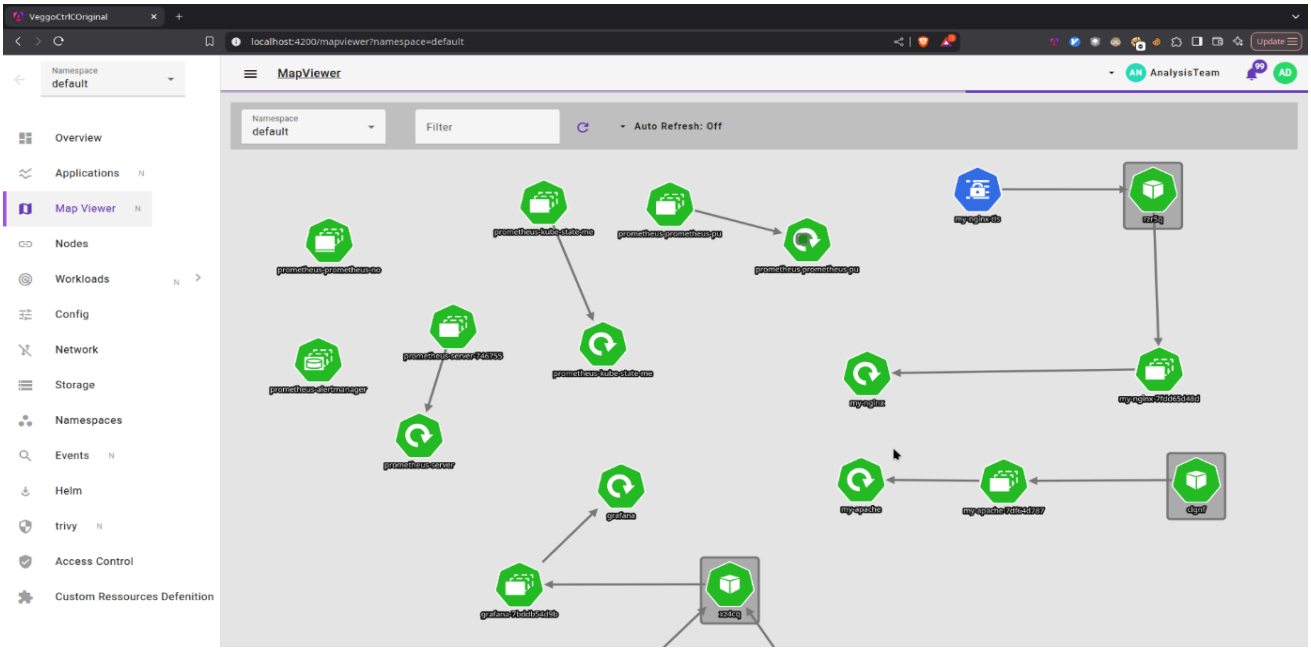

Figure 4.7: Interface « MapViewer »

4.6.1. Interface Dialog « Resource Information »

Après avoir visualisé l'interface, le chef d'équipe peut interagir avec la vue en cliquant sur les icônes. Lorsqu'une icône est sélectionnée, une fenêtre de dialogue s'ouvre pour afficher les informations correspondantes à la ressource cliquée.

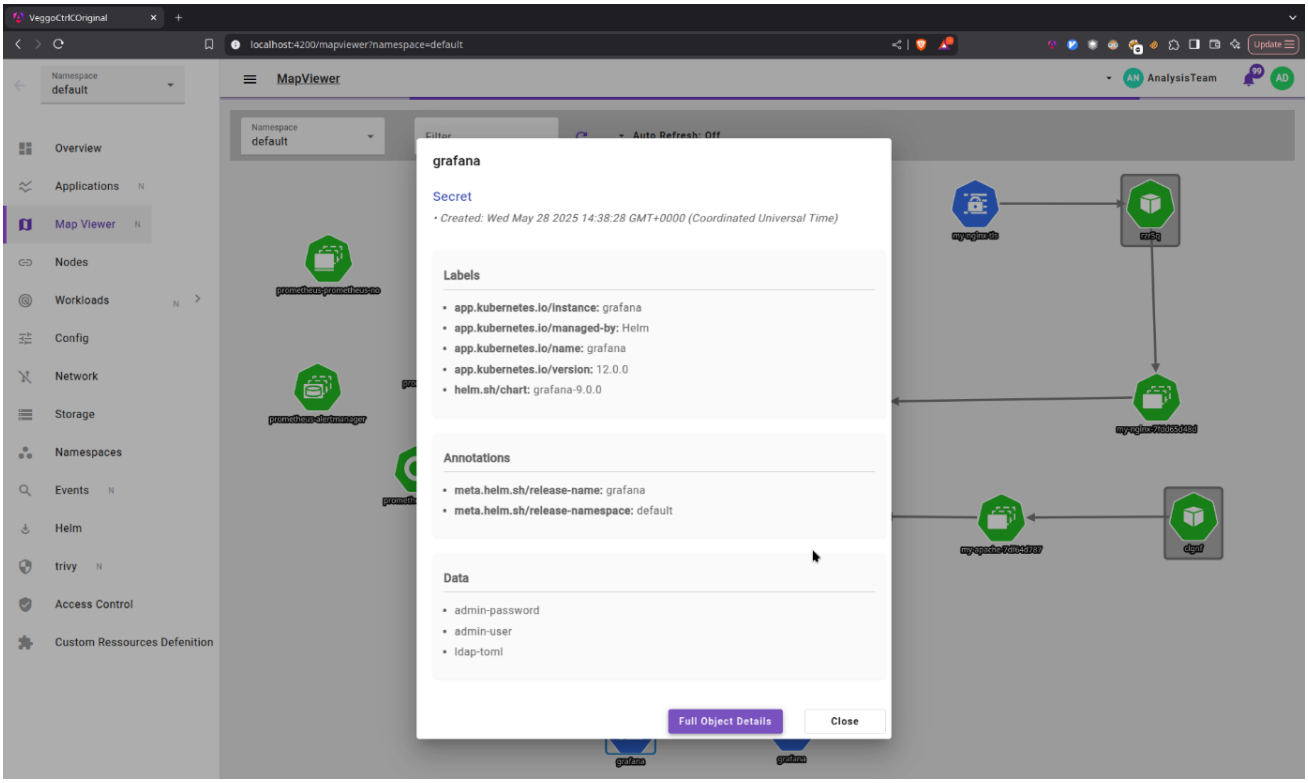

Figure 4.8: Interface « Resource Information »

4.6.2. Interface Dialog « Modifier Resource – YAML »

Suite à l'action précédente, le chef d'équipe peut modifier la configuration des ressources, avec la possibilité de choisir entre les formats YAML ou JSON. Cette fenêtre de modification n'est pas accessible uniquement depuis l'interface de visualisation, mais peut également être invoquée depuis d'autres sections dédiées à la gestion des ressources. Elle est accessible au chef d'équipe, ainsi qu'au développeur s'il dispose des permissions nécessaires.

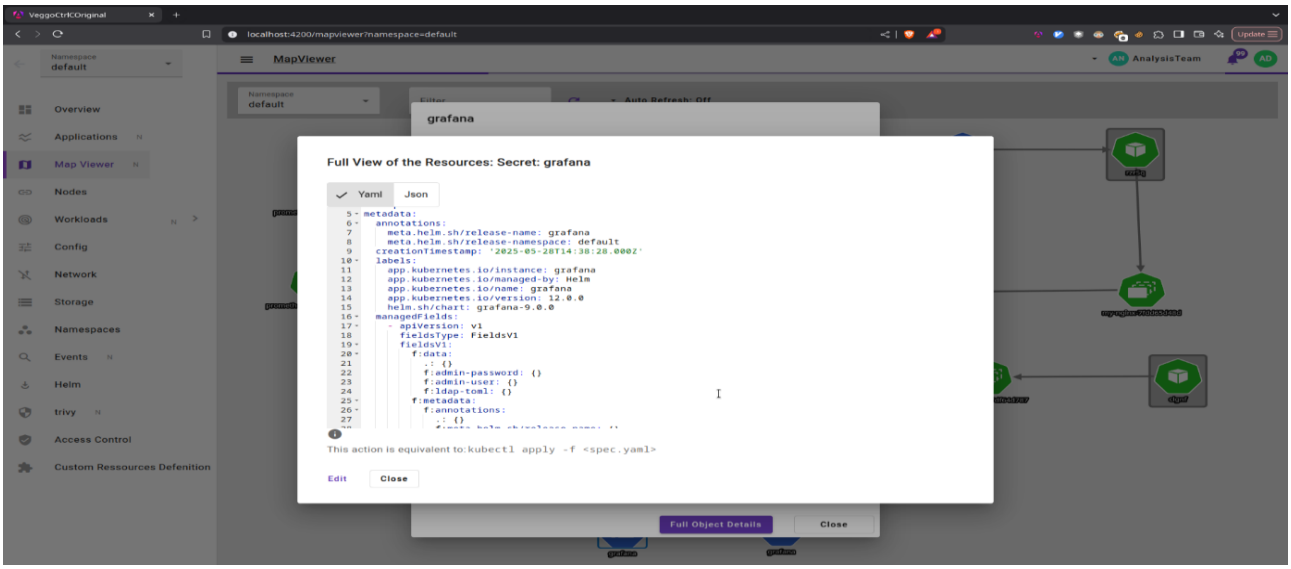

Figure 4.9 Interface « Resource Modification – YAML »

4.6.3. Interface Dialog « Modifier Resource –JSON »

Il s'agit de la même interface que précédemment, mais permettant la modification de la ressource spécifiquement au format JSON.

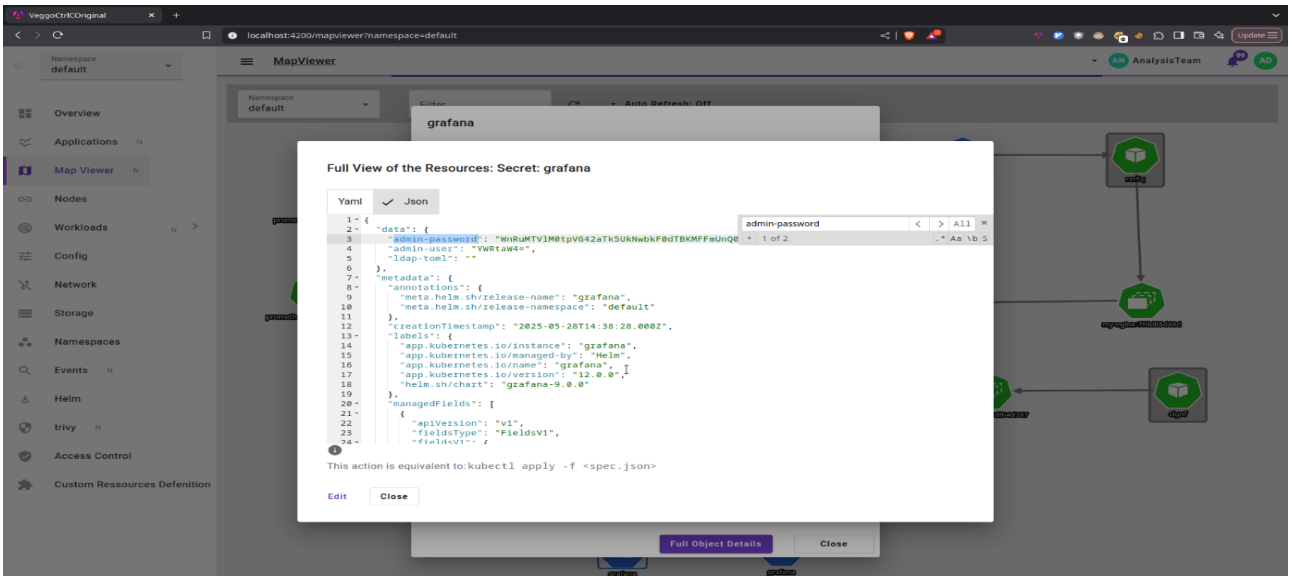

Figure 4.10: Interface « Resource Modification – JSON »

4.7. Interface « Workloads »

Cette interface affiche l'état et le nombre de ressources dans la section *Workloads* pour un *namespace* choisi. Elle inclut les types suivants : *Pod*, *CronJob*, *DaemonSet*, *Deployment*, *Job*, *ReplicaSet*, *ReplicationController* et *StatefulSet*. Elle est accessible à tous les utilisateurs. Toutefois, si un développeur ne dispose pas des permissions nécessaires pour visualiser ou modifier certaines ressources, un message indiquant "aucune ressource trouvée " sera affiché à la place.

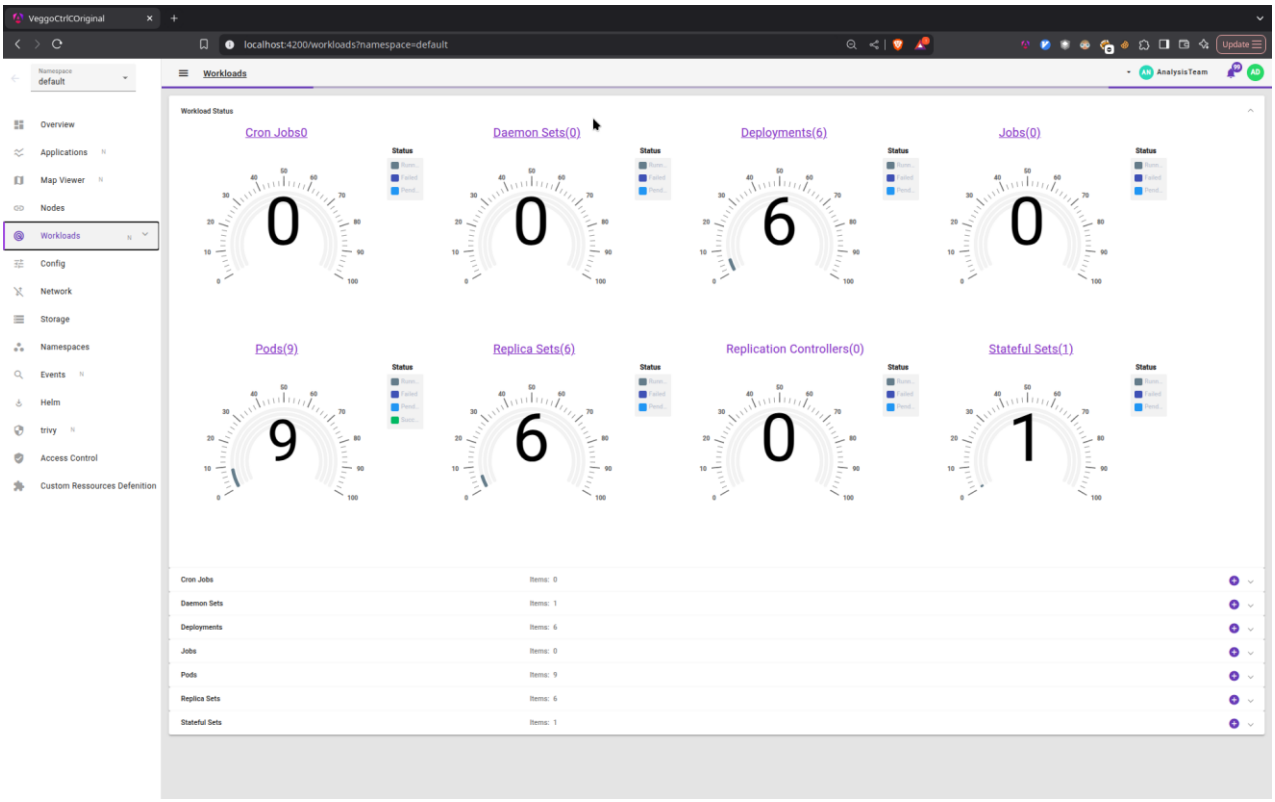

Figure 4.11: Interface « Workloads »

4.8. Interface « List Resource »

Cette interface permet à l'utilisateur de visualiser, par exemple, tous les *Pods* d'un *namespace* sélectionné et d'interagir avec les ressources. Il peut afficher les détails d'un *Pod*, en créer, modifier, filtrer, télécharger ou supprimer. L'utilisateur peut également consulter les journaux (*logs*) et se connecter au conteneur du *Pod*. Toutes ces actions sont disponibles pour l'administrateur et le chef d'équipe. En revanche, un développeur doit d'abord obtenir les droits d'accès nécessaires pour pouvoir effectuer ces opérations. Cette interface est commune à l'ensemble des ressources Kubernetes, avec des fonctionnalités spécifiques à chaque type. Par exemple, seul le *Pod* permet une connexion directe à son conteneur.

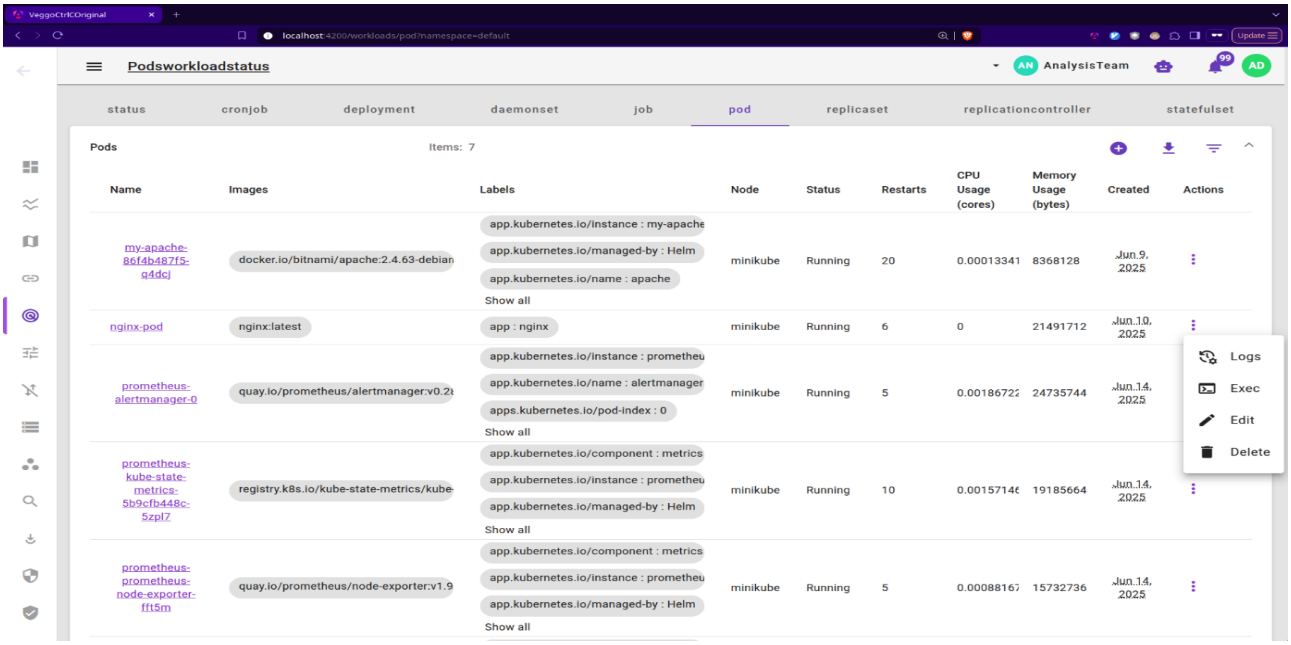

Figure 4.12: Interface « Resource List »

4.8.1. Interface « Création de Ressource »

Lorsqu'un utilisateur choisit de créer une ressource, une nouvelle interface s'affiche en bas de la liste actuelle. En fonction du type de ressource sélectionné, un éditeur web adapté est présenté, accompagné d'indications sur la structure à respecter pour créer correctement la ressource. L'utilisateur peut également choisir parmi plusieurs modes de création : via un formulaire interactif, via un éditeur de code, ou en téléversant un fichier existant (YAML ou JSON). Cette action est disponible pour l'administrateur et le chef d'équipe. Elle peut également être effectuée par un développeur, à condition qu'il dispose des autorisations nécessaires.

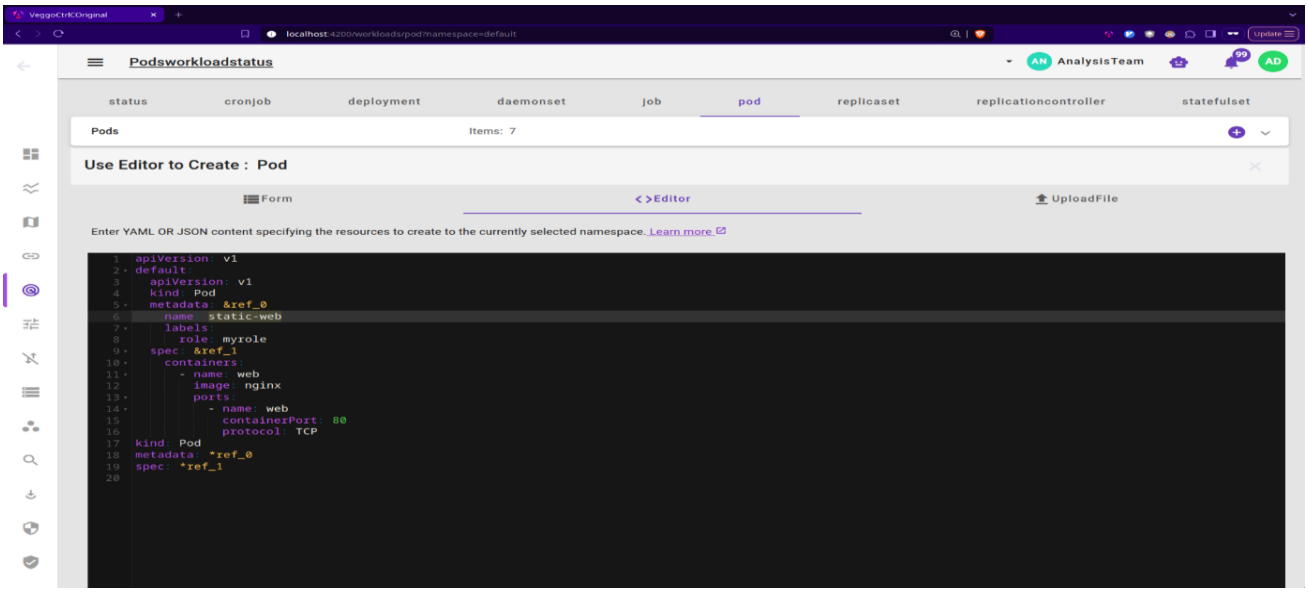

Figure 4.13: Interface « Resource Creation »

4.8.2. Interface « Resource Detail »

L'interface **Resource Detail** est affichée lorsqu'un utilisateur clique sur une ressource dans la liste. Elle permet de consulter l'ensemble des informations associées à cette ressource spécifique. Si la ressource dépend d'autres ressources ou possède des intégrations avec des ressources d'un type différent, ces relations seront également présentées de manière claire dans cette interface. Cela permet à l'utilisateur d'avoir une vue complète de la structure et des dépendances de la ressource sélectionnée.

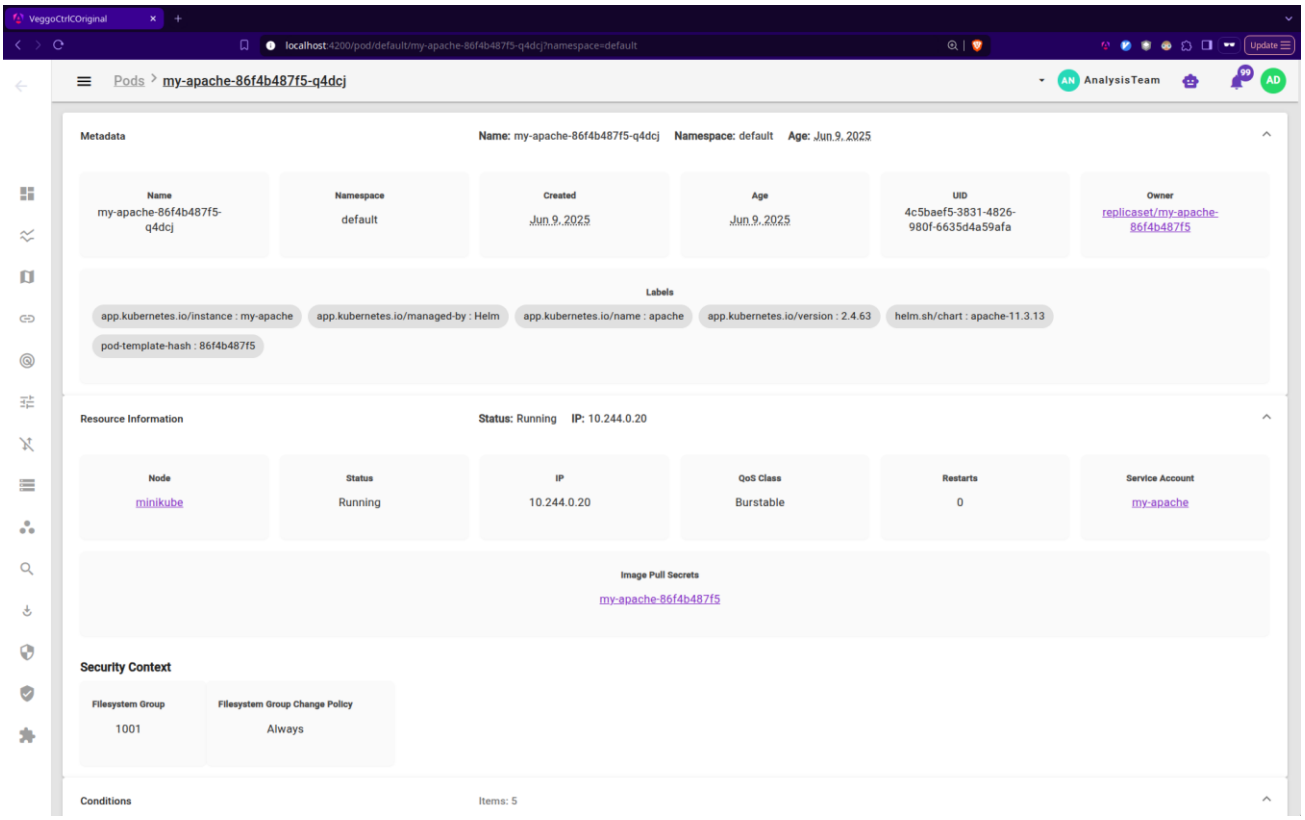

Figure 4.14: Interface « Resource Detail »

4.8.3. Interface « Journaux d'application »

L'interface d'affichage des logs d'une application sur Kubernetes permet aux utilisateurs de consulter en temps réel ou de manière historique les messages générés par les conteneurs en cours d'exécution. Elle offre une vue centralisée et structurée des journaux, facilitant le suivi du comportement des applications déployées dans le cluster. Grâce à cette interface, les développeurs et les administrateurs peuvent identifier rapidement les erreurs, surveiller les performances et diagnostiquer les incidents sans avoir à accéder directement aux pods. Elle permet aussi souvent de filtrer les logs par namespace, pod ou conteneur rendant l'analyse plus efficace et ciblée.

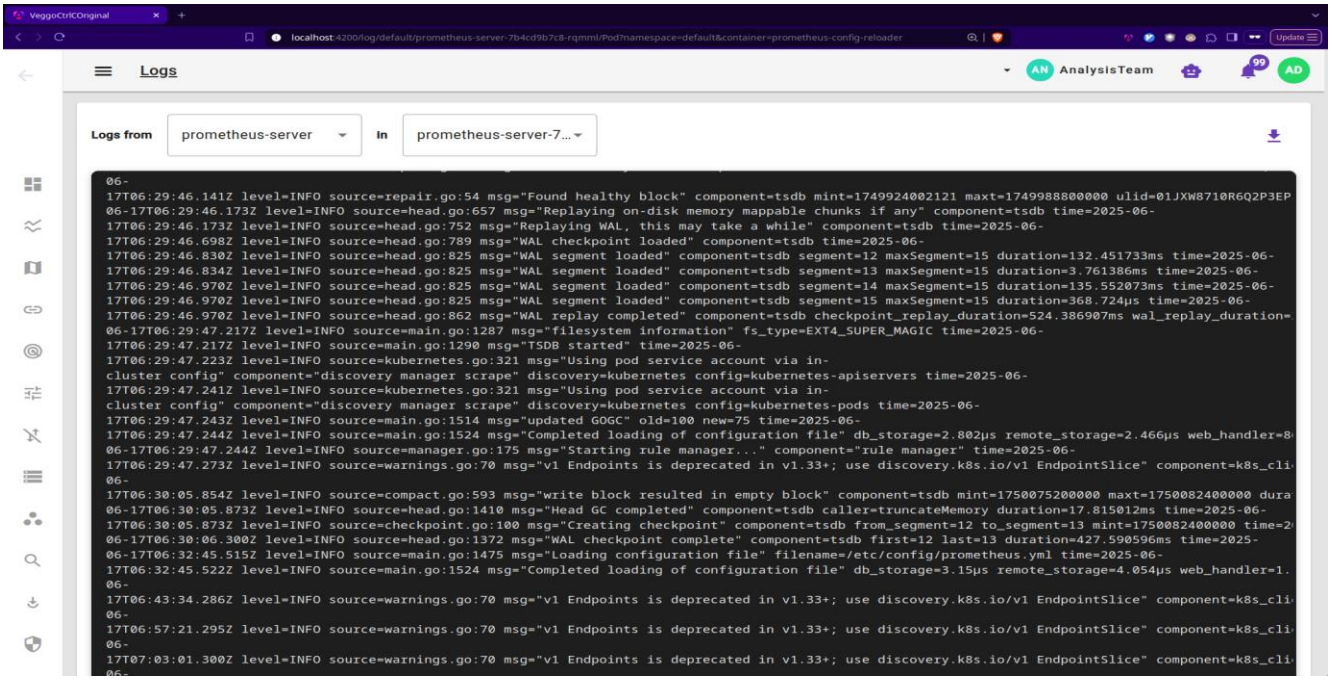

Figure 4.15 : Interface « Journaux d'application »

4.9. Interface « Package Installation »

L'interface **Package Installation** permet à l'utilisateur de rechercher et d'installer une application spécifique au sein d'un *namespace* sélectionné.

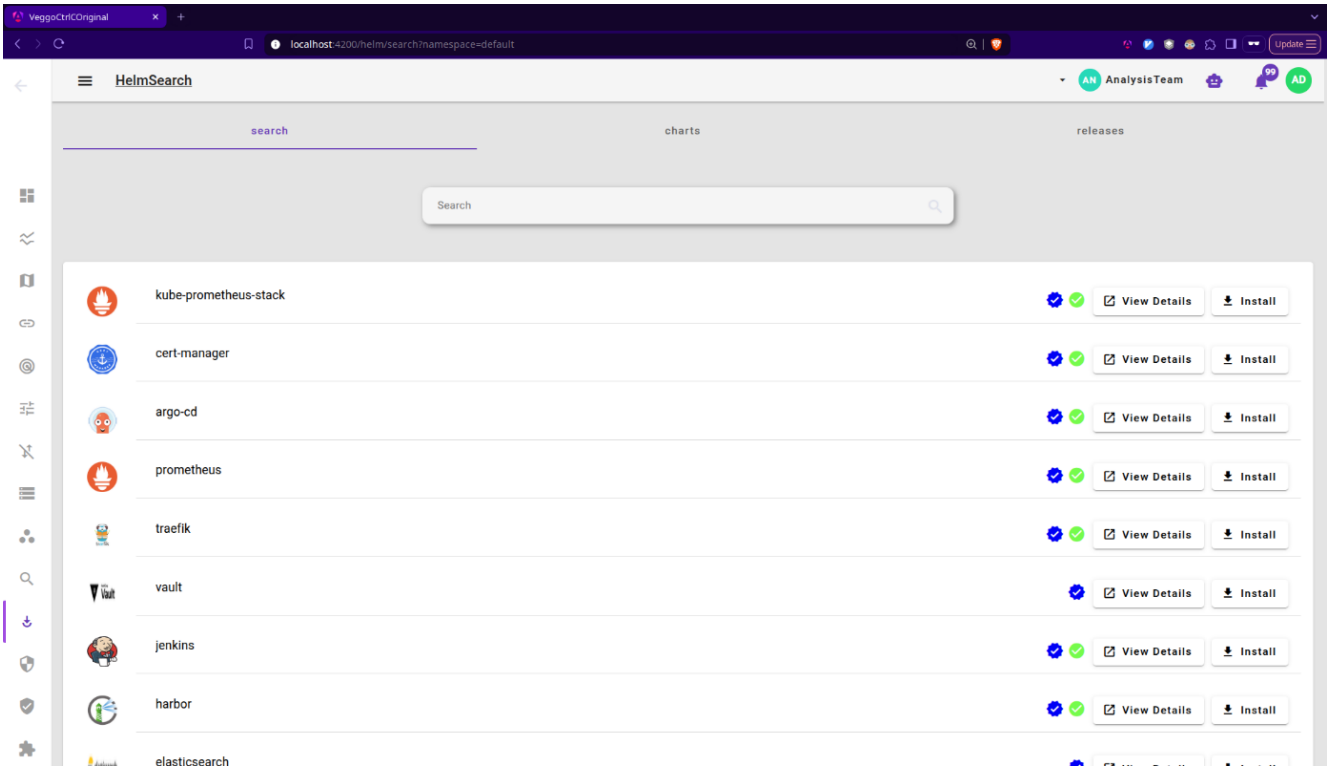

Figure 4.16: Interface « Pacage Installation »

Conclusion

Dans ce dernier chapitre, j'ai commencé par présenter les environnements logiciels et matériels que j'ai utilisés tout au long du projet. Ensuite, j'ai présenté les frameworks utilisés pour construire l'application, à savoir Angular côté front-end et Express côté back-end. En conclusion, j'ai présenté plusieurs interfaces illustrant les fonctionnalités principales de l'application.

Conclusion Générale

Mon projet, intitulé **« Veggo Control Center »**, synthétise le travail que j'ai réalisé pendant quatre mois au sein de la société **Vermeg**. Il s'agit d'une application web développée dans le but de faciliter la gestion de l'infrastructure kubernetes pour les équipes **DevOps**.

Dans le premier chapitre, j'ai commencé par présenter le **contexte général** du projet. J'ai défini le cadre dans lequel il s'inscrit, puis j'ai analysé les processus et les solutions informatiques existantes, afin de mieux orienter mon sujet. J'ai également comparé différentes méthodologies de gestion de projet, ce qui m'a conduit à choisir la méthode **2TUP**.

Le deuxième chapitre a porté sur l'analyse et la spécification des besoins, qu'ils soient fonctionnels ou non fonctionnels. Pour structurer cette phase, j'ai eu recours à des diagrammes de cas d'utilisation ainsi qu'a des diagrammes des séquences. Cette section comprend également la justification des choix techniques effectués, notamment l'adoption d'Angular pour le front-end et d'Express pour le back-end.

Le troisième chapitre est consacré à la présentation de l'architecture du système, à la conception de la base de données ainsi qu'à la conception logicielle détaillée. J'y ai également intégré le maquettage de certaines interfaces de l'application, afin d'illustrer visuellement les choix de conception.

Enfin, le quatrième chapitre est consacré à la présentation des outils logiciels et matériels utilisés tout au long du développement. Il se termine par quelques captures d'écran illustrant les principales fonctionnalités mises en œuvre.

Ce projet m'a offert l'opportunité d'appliquer les connaissances théoriques acquises durant ma formation universitaire. Il a également contribué à renforcer mes compétences, en particulier en gestion de projet et en développement d'applications web.

En termes de perspectives, plusieurs axes d'amélioration sont possibles. Certaines fonctionnalités n'ont pas encore été développées, et je pense qu'il serait pertinent de les ajouter à l'avenir. Ce projet pourrait également être étendu pour couvrir des aspects plus complexes de Kubernetes, nécessitant un niveau d'expertise plus avancé.

Bibliographie

[1] Site officiel du Vermeg. Available: https://www.vermeg.com/ Consulté le : 10/02/2025.

[2] Site officiel du Veggo. Available: https://veggoplatform.com/ Consulté le : 15/02/2025.

[3] Site officiel du Kubernetes. Available: https://kubernetes.io/ Consulté le : 28/02/2025.

[4] Site officiel du Lens. Available: https://k8slens.dev/ Consulté le : 05/03/2025.

[5] Site officiel du Wikipedia. Available: https://fr.wikipedia.org/wiki Consulté le : 05/03/2025.

[6] Site officiel du Minikube. Available: https://minikube.sigs.k8s.io/docs/start/ Consulté le : 17/03/2025.

[7] Site officiel du Docker. Available: https://www.docker.com/ Consulté le : 17/03/2025.

[8] Site officiel du Neovim. Available: https://neovim.io/ Consulté le : 25/03/2025.

[9] Site officiel du Staruml. Available: https://staruml.io/ Consulté le : 03/04/2025.

[10] Site officiel du Balsamiq. Available: https://balsamiq.com/ Consulté le : 03/04/2025.

[11] Site officiel du Dex. Available: https://dexidp.io/docs/guides/kubernetes/ Consulté le : 10/05/2025.

[12] Site officiel du Oauth2 . Available: https://oauth.net/2/ Consulté le : 10/05/2025.

[13] Site officiel du Helm. Available: https://helm.sh/ Consulté le : 10/05/2025.

[14] Site officiel du Grafana. Available: https://grafana.com/ Consulté le : 24/03/2025.

[15] Site officiel du Prometheus. Available: https://prometheus.io/ Consulté le : 24/03/2025.

[16] Site officiel du Trivy. Available: https://trivy.dev/v0.62/docs/target/kubernetes/ Consulté le : 01/05/2025.

[17] Site officiel du kubernetes-client javascript. Available: https://kubernetes-client.github.io/javascript/ Consulté le : 17/03/2025.

[18] Site officiel du Express. Available: https://expressjs.com/ Consulté le : 17/03/2025.

[19] Site officiel du Angular. Available: https://angular.dev/ Consulté le : 10/02/2025.

[20] Site officiel du Swagger. Available: https://swagger.io/ Consulté le : 28/02/2025.

[21] Site officiel du Postman. Available: https://www.postman.com/ Consulté le : 03/04/2025.

[22] Site officiel du NodeJs. Available: https://nodejs.org Consulté le : 29/04/2025.

[23] Site officiel du PostgreSQL. Available: https://www.postgresql.org/ Consulté le : 29/04/2025.

Annexe

API Kubernetes

L'API Kubernetes est le cœur du système, servant d'interface centrale pour communiquer avec le cluster. Elle expose les fonctionnalités de kubernetes à travers une interface REST, permettant de gérer l'ensemble des composants du cluster de manière déclarative. Toutes les informations concernant l'état du cluster et les ressources telles que les pods, services ou encore les configurations sont représentées sous forme d'objets que l'on peut créer modifier ou supprimer via des requêtes HTTP. Les utilisateurs peuvent interagir avec cette API directement ou au moyen d'outils comme kubectl, qui facilitent l'envoi de commandes.

App Container

Dans kubernetes, un conteneur d'application est un conteneur principal du pod, chargé de faire fonctionner la logique métier ou l'application elle-même. Ces conteneurs ne démarrent qu'une fois que tous les conteneurs d'initialisation (init container) du pod ont terminé avec succès. Les conteneurs d'initialisation servent à exécuter des tâches préparatoires (chargement de fichiers, vérifications de dépendances, etc.) qui ne sont plus nécessaires une fois que le ou les conteneurs d'application sont lancés. Dans le cas où aucun init container n'est défini, tous les conteneurs d'un pod sont alors considérés comme des conteneurs d'application.

Chart Helm

Un Chart Helm est un paquet qui regroupe un ensemble de ressources kubernetes prêtes à être déployées, organisées selon une structure standardisée. Il permet de décrire, versionner, et déployer des applications kubernetes de manière cohérente et automatisée à l'aide de l'outil Helm. Que ce soit pour déployer un composant simple (comme un pod avec un cache en mémoire) ou une architecture plus complexe (comme une application web multi-services avec bases de données, backend et frontend), les charts offrent un moyen réutilisable et partageable d'opérer sur kubernetes.

Cluster

Un cluster kubernetes est une unité de déploiement composée d'un ensemble de machines (physique ou virtuelles), appelées nœuds, qui exécutent des applications conteneurisées. Il est généralement constitué de :

- Un ou plusieurs nœuds de contrôle (ou « master nodes ») qui pilotent le cluster (API, planification, contrôleurs…)
- Plusieurs nœuds de travail (ou « worker nodes ») qui hébergent les conteneurs des applications.

Le cluster représente l'environnement dans lequel kubernetes orchestre et assure le cycle de vie des applications.

ConfigMap

Une ConfigMap est un objet kubernetes conçu pour stocker des données de configuration non sensibles, sous forme de paires clé-valeur. Elle permet d'externaliser la configuration d'une application hors du code source ou de l'image du conteneur. Ces données peuvent être injectées dans les pods sous forme d'environnement, d'arguments de ligne de commande ou de fichiers montés dans un volume. Pour des données confidentielles, il est recommandé d'utiliser des objets Secret.

Container Runtime

Le runtime de conteneur est le composant logiciel responsable de l'exécution des conteneurs sur un nœud. Kubernetes s'appuie sur ce runtime pour gérer le cycle de vie des conteneurs. Parmi les runtimes supportés figurent docker, containerd, CRI-O, entre autres. Chaque nœud kubernetes doit disposer d'un runtime compatible avec l'interface CRI.

Contrôleur

Un Contrôleur est une boucle de contrôle dans kubernetes qui surveille l'état actuel d'un cluster via l'API et prend des mesures pour rapprocher cet état de l'état souhaité. Exemples : DeploymentController, ReplicaSetController, EndpointController, etc. Chaque contrôleur agit de manière autonome mais coordonnées pour garantir la cohérence du système.

CronJob

Un CronJob permet de programmer l'exécution automatique de taches périodiques dans kubernetes, selon une syntaxe similaire à celle des fichiers crontab. Chaque exécution planifiée génère un Job, qui exécute la tâche à la fréquence spécifiée.

CustomResourceDefinition

Une CRD est une extension de l'API kubernetes permettant d'ajouter de nouveaux types de ressources personnalisées sans modifier le code de kubernetes. Cela permet aux développeurs

d’étendre les fonctionnalités de kubernetes pour répondre à des besoins spécifiques à leur application ou leur domaine.

DaemonSet

Un DaemonSet garantit qu’un pod spécifique est exécuté sur chaque nœud (ou sur une sélection de nœuds) du cluster. Il est souvent utilisé pour déployer des services de bas niveau comme des agents de monitoring, de log ou de sécurité.

Déploiement

Un Deployment est un objet kubernetes permettant de gérer le cycle de vie d’un ensemble de pods répliques. Il automatise la création, la mise à jour et la montée ou descente en charge d’une application conteneurisée tout en garantissant une haute disponibilité.

Endpoints

Les Endpoints sont des objets kubernetes qui listent les adressIP (et ports) des pods cibles par un service. Ils permettent d’associer dynamiquement un service a ses backends réels. Pour les cas à grande échelle, kubernetes propose aussi EndpointSlices, plus performants et extensibles.

Etcd

ETCD est une base de données distribuée clé-valeur utilisée comme source de vérité pour le cluster kubernetes. Elle stocke l’ensemble des données critiques, notamment la configuration du cluster, les objets API, les états des ressources, etc. Il est essentiel de prévoir des mécanismes de sauvegarde et restauration pour etcd.

Groupe d'API

Un groupe d’API regroupe des chemins et des versions liées à une famille de ressources dans l’API Kubernetes. Cela permet de structurer l’API et de versionner des fonctionnalités indépendamment (exp : app/v1, batch/v1, networking.k8s.io/v1…).

Horizontal Pod Autoscaler

L’HPA est un objet kubernetes qui ajuste automatiquement le nombre de pods répliqués en fonction de la charge, comme l’utilisation CPU ou d’autres métriques personnalisées. Il fonctionne avec des objets élastiques comme Deployment ou ReplicaSet, mais pas avec des objets fixes comme DaemonSet.

Init Container

Un init container est un conteneur temporaire exécuté avant les conteneurs d'application dans un pod. Il est utilisé pour préparer l'environnement, par exemple en copiant des fichiers ou en s'assurant que des dépendances externes soient prêtes. Il doit se terminer avec succès pour que le pod continue son démarrage.

Job

Un Job est une ressource kubernetes utilisée pour exécuter des tâches ponctuelles ou de type batch. Il crée un ou plusieurs pds et veille à ce qu'un nombre donné d'exécutions se terminent avec succès. Une fois les objectifs atteints, le job est considéré comme terminé. Idéal pour les tâches de traitement de données, scripts ponctuels ou traitements automatisés.

JSON Web Token (JWT)

Un moyen de représenter des revendications (ou "claims") à transférer entre deux parties. Les JWT peuvent être signés numériquement et chiffrés. Kubernetes utilise les JWT comme jetons d'authentification pour vérifier l'identité des entités souhaitant effectuer des actions dans un cluster.

Kubernetes

Kubernetes est une plateforme d'orchestration open source conçue pour automatiser le déploiement, la gestion et la mise à l'échelle des applications conteneurisées. Développé à l'origine par Google, il s'inspire de leur système interne Borg. Kubernetes gère l'état désire de l'infrastructure applicative et répartit dynamiquement les ressources sur un cluster de machines.

Label

Un label est une étiquette clé-valeur attachée à des objets kubernetes pour faciliter leur identification, tri ou sélection. Les labels sont utilisés pour organiser les ressources, mais aussi pour les cibler avec des services, des policies ou des sélecteurs.

Manifest

Un manifeste est un fichier de configuration, au format YAML ou JSON, décrivant un ou plusieurs objets kubernetes. Il définit l'état désire d'une ressource : nom, type, sec, metadata, etc.

Name

Le nom est l'identifiant unique d'un objet kubernetes dans un namespace donné. Il est utilisé dans les URL de l'API pour faire référence à des ressources, par exp /api/v1/pods/nom-pod. Ce nom peut être réutilisé une fois l'objet supprimé.

Namespace

Un Namespace est une division logique du cluster permettant d'isoler des groupes de ressources. Il facilite la gestion multi-utilisateurs ou multi-environnements (exp : dev, staging, production), chacun pouvant contenir ses propres services, pods, secrets, etc.

Password Salt

L'entité représente une valeur aléatoire ajoutée devant le password original pour éliminer que deux personnes peuvent avoir un password identique.

Pod

Le Pod est l'unité de base dans kubernetes. Il encapsule un ou plusieurs conteneurs partageant le même espace réseau et stockage. Les pods sont éphémères et généralement orchestrés via des deployments pour assurer leur disponibilité et leur remplacement automatique.

StatefulSet

Le StatefulSet est un contrôleur kubernetes utilisé pour déployer des applications nécessitant une identité stable et une persistance. Contrairement aux Deployments, les pods crées par un statefulSet ont chacun un nom unique, un ordre de démarrage/arrêt contrôle, et conservent leur volume de stockage même après un redémarrage. Cela en fait le choix idéal pour les bases de données distribuées, les caches persistants, ou les services nécessitant un état durable.

Workload

Une workload désigne une application ou un processus tournant sur kubernetes. Elle peut être déployée à l'aide de plusieurs objets, comme Deployment, StatefulSet, DaemonSet, Job ou ReplicaSet. Chaque type d'objet répond à des besoins spécifiques, par exp : réplicabilité, persistance, tâches ponctuelles ou processus distribués.

WSEndpoint

L'entité Web Service Endpoint représente une url ou api Endpoint pour gérer l'authentification requête.