Université de Paris
UFR Mathématiques et Informatique

Programmation Distribuée

Master 1 Intelligence Artificielle Distribuée

Daniel LATORRE
Encadré par Benoit CHARROUX

Année universitaire 2023 - 2024

Table des matières

Chapitre 1

Introduction

Ce rapport explore une application de gestion des tâches développée dans le cadre de la matière "Programmation distribuée", en mettant l'accent sur les technologies et l'architecture utilisées. L'application est conçue avec un ensemble de technologies modernes : React pour la construction d'interfaces utilisateurs interactives, Material-UI pour le design des composants de l'interface, Node.js pour le traitement côté serveur, et MongoDB pour la gestion des données. Ces technologies ont été choisies pour leur performance, leur flexibilité et leur popularité dans le développement d'applications web modernes.

L'architecture de l'application repose sur un modèle à trois niveaux (three-tier), avec une présentation frontend dynamique, une logique backend exécutée sur Node.js, et une persistance des données via MongoDB. Cette structure est déployée dans un environnement Kubernetes, qui supporte la scalabilité et la haute disponibilité nécessaires pour une application d'entreprise.

Ce rapport détaille chacune des technologies choisies, leur rôle dans l'application, et la manière dont elles interagissent pour fournir une solution robuste et scalable. L'analyse se concentre également sur la manière dont cette architecture supporte les besoins fonctionnels de gestion des tâches.

Chapitre 2

Aperçu du Projet

L'application est spécialement conçue pour optimiser la gestion quotidienne des tâches via une interface interactive permettant aux utilisateurs d'ajouter, de supprimer, de modifier et de visualiser leurs tâches. Elle intègre des fonctionnalités de mise à jour et de synchronisation des données en temps réel, assurant que les utilisateurs aient accès en permanence aux informations les plus récentes. Cette synchronisation est cruciale pour améliorer à la fois la satisfaction et la productivité des utilisateurs, en minimisant les erreurs et les duplications dans le traitement des tâches.

Chapitre 3

Architecture Détaillée

3.1 Architecture Frontend

Le frontend, développé avec React, facilite des interactions utilisateur fluides et réactives grâce à son modèle de composants dynamiques. Material-UI est utilisé pour fournir une suite de composants UI prêts à l'emploi qui ne sont pas seulement fonctionnels mais aussi esthétiquement agréables, renforçant ainsi l'engagement utilisateur.
Les fichiers clés du frontend incluent :

- **App.js** : C'est le composant principal qui gère l'état de l'application et orchestre le rendu de l'interface utilisateur, incluant les sous-composants pour la gestion des tâches. Il gère les transitions d'état basées sur les actions des utilisateurs, telles que l'ajout d'une nouvelle tâche ou la marque d'une tâche comme complétée.
- **Tasks.js** : Responsable de la communication avec le backend, ce fichier contient des fonctions qui réalisent les opérations CRUD en utilisant Axios pour envoyer et recevoir des données du backend, assurant que la liste des tâches est toujours à jour.

3.2 Architecture Backend

Le backend utilise Node.js et le framework Express pour créer une API robuste capable de gérer les requêtes efficacement. L'architecture backend comprend :

- **Model_task.js** : Définit le schéma MongoDB pour les tâches en utilisant Mongoose, qui aide à valider les données avant leur stockage dans la base de données. Il comprend des définitions pour les descriptions de tâches et le statut d'achèvement.
- **Routes-task.js** : Gère les routes API qui facilitent les fonctionnalités CRUD. Cette configuration lie efficacement les requêtes utilisateur du frontend aux opérations de base de données correspondantes.
- **Db.js** : Gère les paramètres de connexion à la base de données, utilisant des variables d'environnement pour renforcer la sécurité et assurer des connexions fiables à la base de données MongoDB.

Chapitre 4

Stratégie de Déploiement Kubernetes

Le déploiement de l'application dans un environnement Kubernetes garantit sa scalabilité et sa haute disponibilité :

- **Configuration MongoDB** : MongoDB est déployé comme un ensemble stable dans Kubernetes, utilisant une revendication de volume persistant pour garantir que les données ne sont pas perdues lorsque le pod est redémarré. Il est configuré pour permettre les connexions uniquement à l'intérieur du cluster pour des raisons de sécurité.
- **Configuration API Backend** : L'API backend est déployée avec plusieurs répliques pour assurer une haute disponibilité. Elle utilise une stratégie de mise à jour continue pour permettre des mises à jour sans temps d'arrêt.
- **Configuration Frontend et Proxy** : Le frontend est servi par un proxy inverse Nginx, configuré via un ConfigMap Kubernetes, qui route efficacement le trafic entre le frontend et le backend. Cette configuration assure des performances optimales et une scalabilité.

Chapitre 5

Sécurité et Conformité

Des mesures de sécurité rigoureuses sont intégrées, incluant l'utilisation de Secrets Kubernetes pour la gestion des données sensibles et les Politiques Réseau afin de contrôler l'accès au cluster, garantissant ainsi que seules les requêtes légitimes sont traitées.

— **Secrets Kubernetes** : C'est le composant principal qui gère l'état de l'application et orchestre le rendu de l'interface utilisateur, incluant les sous-composants pour la gestion des tâches. Il gère les transitions d'état basées sur les actions des utilisateurs, telles que l'ajout d'une nouvelle tâche ou la marque d'une tâche comme complétée.

— **Politiques Réseau** : Responsable de la communication avec le backend, ce fichier contient des fonctions qui réalisent les opérations CRUD en utilisant Axios pour envoyer et recevoir des données du backend, assurant que la liste des tâches est toujours à jour.

Chapitre 6

Performance et Scalabilité

L'équilibrage de charge est assuré par Nginx et les services Kubernetes, qui répartissent efficacement le trafic pour une meilleure gestion des ressources. L'Autoscaler de Pod Horizontal adapte le nombre de pods selon la charge détectée, permettant une réponse flexible aux variations de demande.

— **Équilibrage de Charge** : Effectué via Nginx et les services Kubernetes, cette approche assure une distribution uniforme du trafic sur les ressources disponibles, optimisant ainsi les performances et minimisant la latence.
— **Autoscaler de Pod Horizontal** : Ce dispositif modifie automatiquement le nombre de répliques de pods en fonction de la consommation de CPU et de mémoire observée, garantissant une adaptation dynamique de l'application à l'intensité du trafic.

Chapitre 7

Résultats

Voici quelques captures d'écran illustrant différentes fonctionnalités de l'application, aussi disponibles dans le répertoire Github :

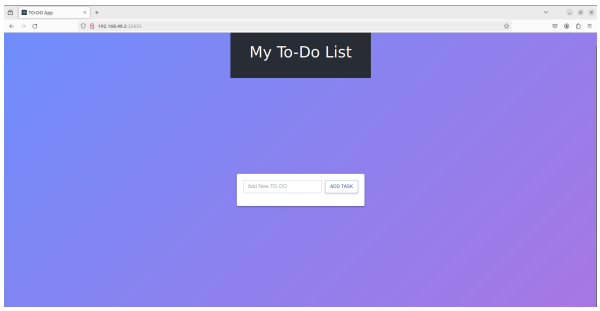

FIGURE 7.1 – Exécution de l'application dans le navigateur.

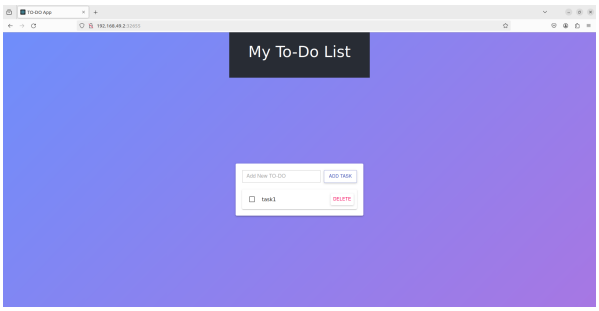

FIGURE 7.2 – Interface permettant l'ajout d'une nouvelle tâche.

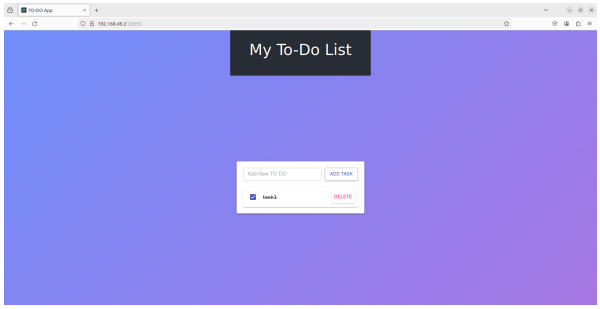

FIGURE 7.3 – Validation d'une tâche après accomplissement.

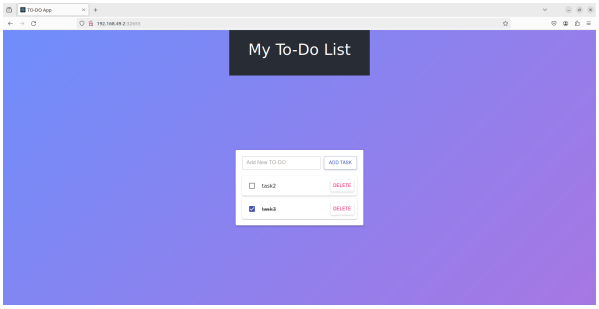

FIGURE 7.4 – Suppression d'une tâche de la liste.

Chapitre 8

Conslusion

Ce rapport a présenté une exploration détaillée de l'application de gestion des tâches développée pour la matière "Programmation distribuée". Les choix technologiques ont été guidés par les exigences de performance, flexibilité, et popularité dans le domaine du développement web moderne. L'utilisation de React et Material-UI pour le développement du frontend a permis de créer une interface utilisateur interactive et esthétiquement plaisante, tandis que Node.js a été choisi pour sa capacité à gérer efficacement les interactions côté serveur grâce à son modèle d'opérations non bloquantes.

La base de données MongoDB complète cette architecture en fournissant un système de gestion de données flexible et performant, idéal pour les opérations CRUD qui sont au cœur de l'application. L'architecture à trois niveaux, intégrant une présentation dynamique, une logique serveur robuste, et une gestion persistante des données, a été conçue pour être déployée dans un environnement Kubernetes, garantissant ainsi scalabilité et haute disponibilité.

En conclusion, cette combinaison de technologies avancées et d'une architecture structurée soutient non seulement les besoins fonctionnels de l'application mais pose également les bases d'une plateforme évolutivement robuste et maintenable. L'architecture détaillée permet une interaction fluide entre les différents composants, optimisant ainsi les performances globales et la réactivité de l'application.

Chapitre 9

Google Labs

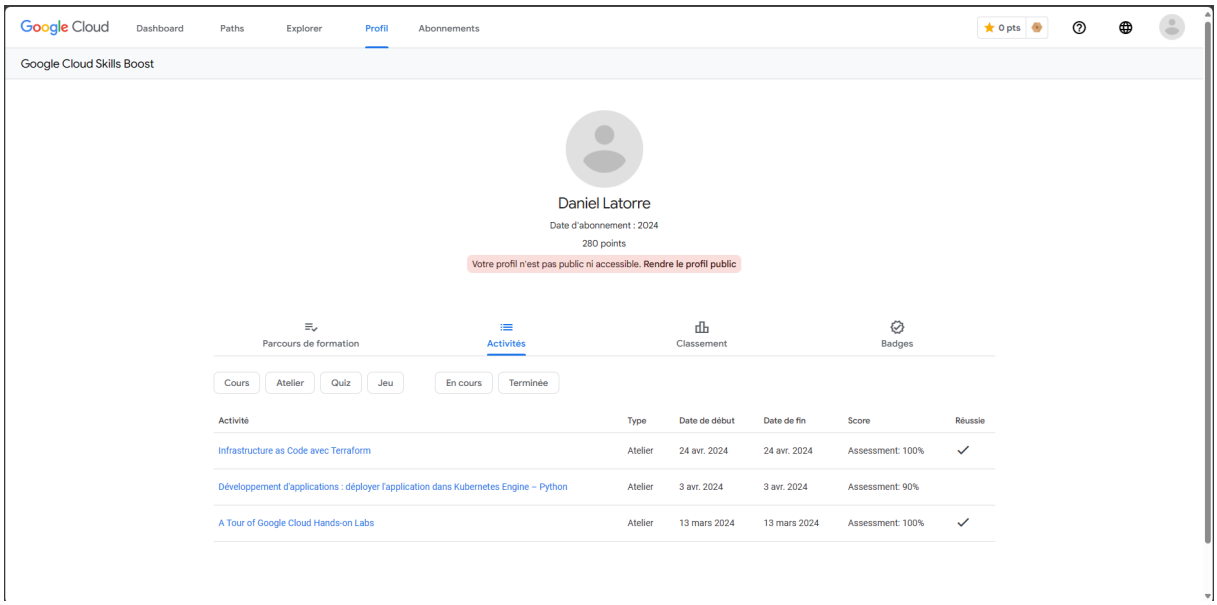

FIGURE 9.1 – Activités Profil