Année universitaire

2024 - 2025

BUT Informatique

BERNARD Quentin

RAPPORT TECHNIQUE

Sujet : µManager

SAÉ – Développement avancé

Département Informatique

Site d'Arles

Tuteur

DESBENOIT Brett

Tables des matières

Introduction

Dans le cadre de la SAÉ du Semestre 5, il nous a été demandé de développer une application mettant en pratique les différentes compétences enseignées à l'IUT. Ce projet doit permettre de démontrer notre maîtrise technique tout en répondant à un besoin concret.

Dans ce cadre, j'ai choisi de développer **µManager**, une application web destinée à faciliter la gestion administrative et financière des micro-entrepreneurs. Ce choix me permet de mettre en œuvre les compétences acquises tout en répondant à une problématique réelle, rencontrée par de nombreux auto-entrepreneurs.

Présentation de µManager

µManager est une application web conçue pour aider les micro-entrepreneurs dans la gestion de leurs tâches administratives et financières.
Elle offre des fonctionnalités clés telles que :

- L'archivage de contrats.
- La génération et le suivi des factures.
- La gestion des paiements et des clients.
- Un historique du Chiffre d'affaires ainsi que des statistique sur ce dernier

L'objectif principal de µManager est de centraliser et simplifier ces processus pour permettre aux entrepreneurs de gagner du temps, d'éviter les erreurs, et de se concentrer sur leur cœur de métier. L'application propose une interface intuitive, accessible à tous, et intègre des outils modernes pour répondre efficacement aux besoins des utilisateurs.

Conception

1. Structure

L'application µManager repose sur une architecture modulaire composée des éléments suivants pour respecter l'approche API-first:

1.1. Frontend :

- Le frontend est une application web développée avec **React** (utilisant Vite pour l'efficacité).
- Il est chargé de fournir une interface utilisateur simple et intuitive pour la gestion des contrats, factures, paiements, et clients.

1.2. Backend :

- Le backend utilise **ExpressJS** avec **TypeScript** pour garantir une structure claire et robuste.
- Il implémente une API REST qui assure la communication entre le frontend et la base de données, tout en gérant la logique métier (CRUD pour les entités principales).
- L'authentification sécurisée est gérée via **JSON Web Tokens (JWT)**, protégeant ainsi les données sensibles.

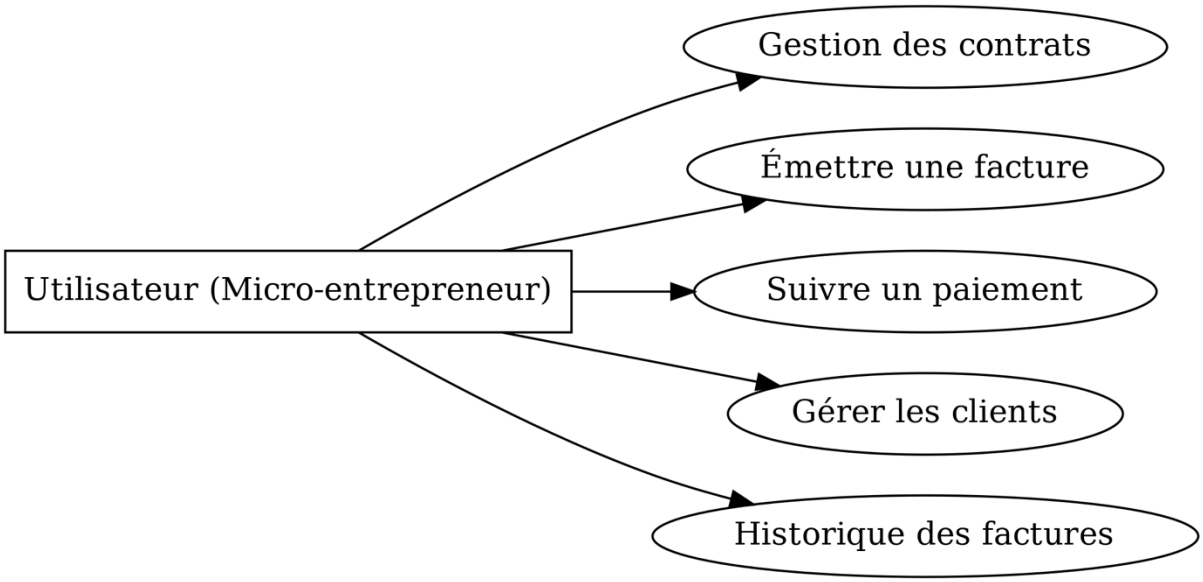
1.3. Base de données :

- La base de données choisie est **MariaDB**, accessible via **phpMyAdmin** pour faciliter la gestion et la visualisation.
- Les principales tables incluent Users, Companies, Clients, Contracts et Invoices, permettant une structuration claire des données administratives.

L'ensemble de ces composants est orchestré via **Docker**, offrant un environnement isolé et facilement déployable.

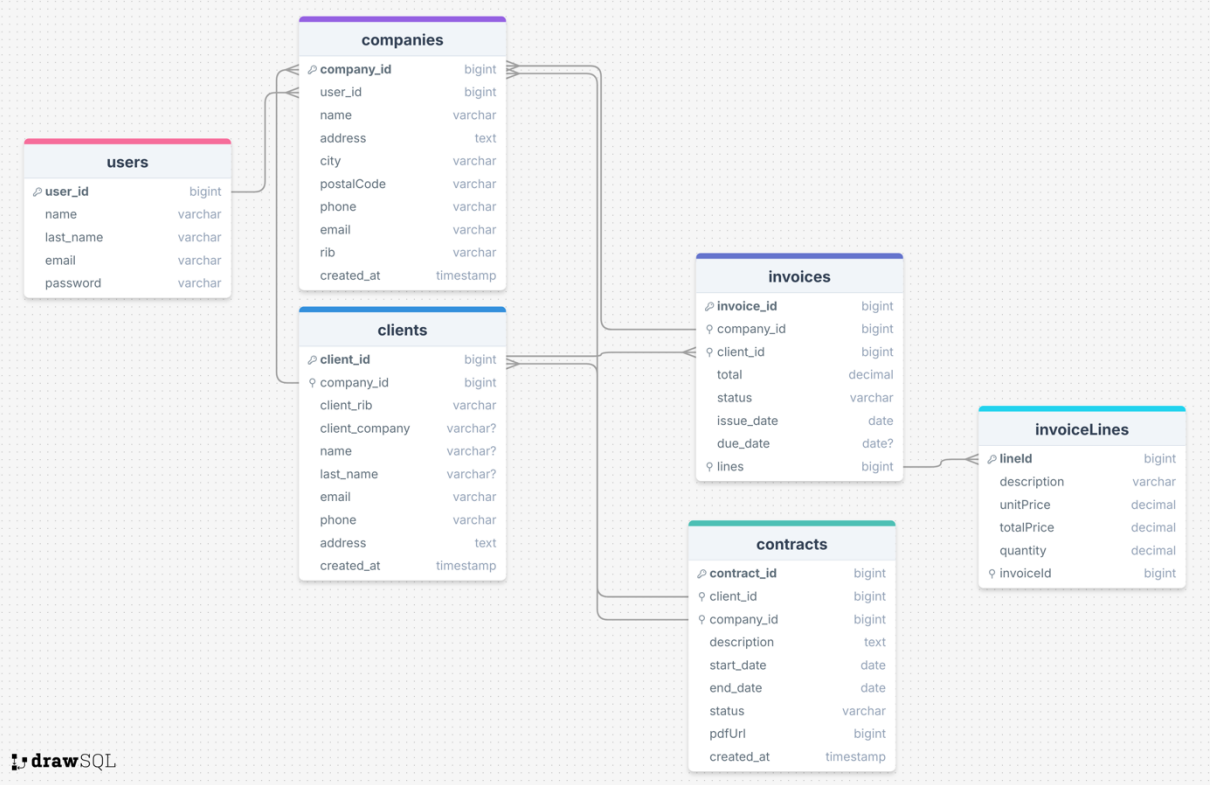
2 Diagrammes UML

2.1. Diagramme de Cas d'Utilisation :



• Illustration des principales interactions entre l'utilisateur (micro-entrepreneur) et l'application, comme la création d'un contrat, l'émission d'une facture, ou le suivi des paiements.

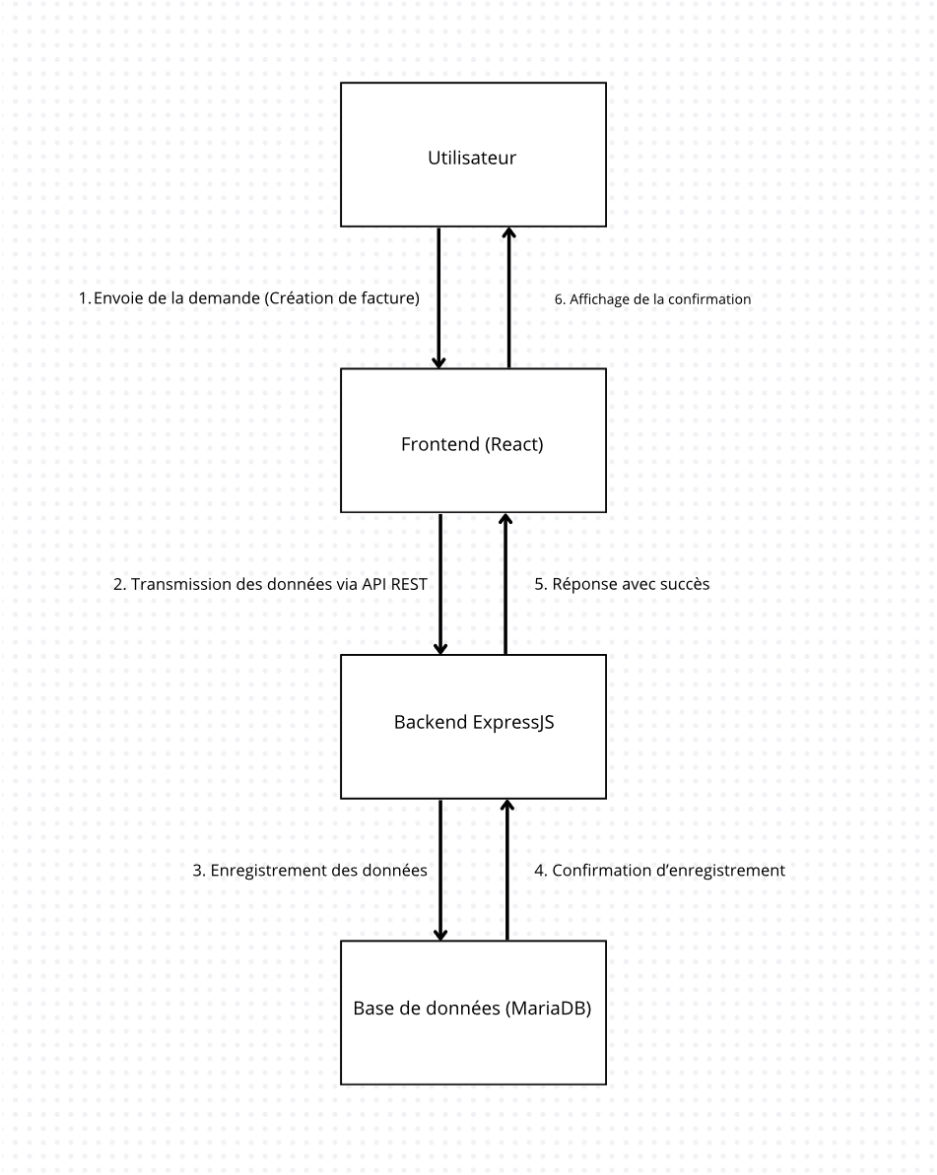
2.2. Diagramme de Classes / Structure de la base de donnée :



- Présente les relations entre les entités principales telles que User, Company, Contract, Invoice.

- Chaque entité inclut des attributs et leurs relations (par exemple, une entreprise peut être lié à plusieurs factures).

2.3. Diagramme de Séquence :



- Montre le déroulement d'une action clé, comme la création d'une facture :
    - L'utilisateur envoie une demande via le frontend.
    - Le backend traite la requête, applique la logique métier, et interagit avec la base de données pour enregistrer la facture.
    - Une confirmation est renvoyée au frontend pour l'afficher à l'utilisateur.

3. Choix Technologiques

3.1. Frontend : React avec Vite

Le frontend de µManager est développé avec **React**, une bibliothèque JavaScript largement adoptée pour sa performance, sa flexibilité, et sa capacité à créer des interfaces utilisateur dynamiques et interactives.

- **Avantages de React** :

  • **Composants réutilisables** : La logique et le rendu sont encapsulés dans des composants indépendants, ce qui facilite la maintenance et la modularité. Par exemple, un composant spécifique peut être créé pour afficher les factures, et ce composant peut être réutilisé ailleurs dans l'application.

  • **Performance avec le Virtual DOM** : Les mises à jour de l'interface sont optimisées en limitant les manipulations directes du DOM, ce qui améliore la réactivité de l'application.

  • **Écosystème riche** : La large communauté React fournit un accès à de nombreuses bibliothèques complémentaires pour gérer des cas spécifiques, comme React Router pour la navigation ou Axios pour les requêtes HTTP.

Vite, un outil de développement moderne, est utilisé pour compléter React.

- **Compilation rapide** : Contrairement à des outils comme Webpack, Vite fonctionne sur une architecture basée sur les modules ES, ce qui réduit considérablement les temps de build.
- **Hot Module Replacement (HMR)** : Cette fonctionnalité permet de visualiser immédiatement les modifications du code dans le navigateur sans recharger toute la page, accélérant ainsi le développement.
- **Optimisation pour la production** : Vite génère un build optimisé et performant, prêt pour un déploiement efficace.

Ces choix garantissent une expérience de développement fluide et un rendu utilisateur réactif et agréable.

3.2. Backend : ExpressJS avec TypeScript

Le backend de µManager repose sur une architecture **API-first**, où toutes les fonctionnalités sont exposées via une API REST, j'ai donc utiliser **ExpressJS**, un framework minimaliste et extensible.

- **ExpressJS** :
  - **Flexibilité** : Permet de configurer rapidement une API tout en laissant une grande liberté dans l'organisation du code.
  - **Simplicité** : Sa syntaxe simple facilite l'apprentissage et la productivité, rendant le framework idéal pour des projets nécessitant un développement rapide.
  - **Extensibilité** : Express peut être facilement combiné avec des middlewares pour gérer des fonctionnalités spécifiques, comme l'authentification avec JSON Web Tokens (JWT).

TypeScript renforce ExpressJS grâce à l'ajout d'un typage statique.

- **Sécurité accrue** : Les erreurs courantes, comme l'utilisation incorrecte de types, sont détectées dès la compilation.
- **Lisibilité** : Les types rendent le code plus clair, facilitant la compréhension et la maintenance, surtout dans un contexte où plusieurs développeurs pourraient intervenir sur le projet.
- **Évolutivité** : TypeScript favorise la structuration du projet, notamment avec la séparation des responsabilités entre les contrôleurs, services et modèles.

Ensemble, ces outils assurent un backend robuste, structuré, et facile à maintenir.

3.3. Base de données : MariaDB

Pour la gestion des données, µManager s'appuie sur **MariaDB**, un système de gestion de base de données relationnel (SGBD) open-source et performant.

- **Avantages de MariaDB** :

  • **Transactions structurées** : Idéal pour les opérations complexes comme l'association d'un client à plusieurs contrats ou factures.

  • **Fiabilité et performance** : MariaDB offre des optimisations par rapport à MySQL, ce qui le rend adapté aux applications nécessitant des opérations fréquentes sur de grands volumes de données.

  • **Compatibilité avec phpMyAdmin** : Cet outil permet de visualiser, gérer et administrer facilement les tables, ce qui simplifie les interactions avec la base de données.

3.4. Docker

Docker a été utilisé pour simplifier la gestion des environnements de développement et de production.

- **Isolation des environnements** : Chaque composant (backend, base de données) fonctionne dans un conteneur distinct, éliminant les conflits liés aux dépendances entre les environnements.
- **Reproductibilité** : Grâce au fichier docker-compose, il est possible de recréer exactement le même environnement sur n'importe quelle machine, garantissant des déploiements sans surprises.
- **Facilité de collaboration** : Même si le projet a été réalisé individuellement, l'utilisation de Docker prépare l'application à un éventuel travail en équipe, car tous les développeurs pourraient bénéficier du même environnement.

- **Standardisation** : Docker assure que les différences entre les environnements de développement, de test et de production sont minimisées, évitant ainsi des bugs liés à des configurations spécifiques.

Réalisation

1. Fonctionnalités Principales

Le projet **µManager** s'articule autour de plusieurs entités principales, chacune dotée de fonctionnalités spécifiques :

A. **Users**

- **Inscription et connexion** : Les utilisateurs peuvent créer un compte et se connecter grâce à un système d'authentification sécurisé basé sur **JWT**.
- **Gestion du compte** : Fonctionnalités de mise à jour des informations personnelles et de suppression du compte.

B. **Companies**

- **Création et gestion** : Chaque utilisateur peut créer et gérer l'entreprises associé à son compte.
- **Attribution aux factures et contrats** : Les entreprises servent de point central pour relier les clients, factures et contrats.

C. **Clients**

- **Enregistrement et gestion** : Les utilisateurs peuvent ajouter et gérer des fiches clients, incluant nom, adresse et contact.
- **Liaison avec les entreprises** : Les clients sont associés à des entreprises spécifiques pour une gestion centralisée.

D. **Invoices**

- **Génération automatique** : Les factures sont créées à partir des données clients et entreprises, avec calcul automatique des montants et de la TVA.
- **Statuts des factures** : Chaque facture dispose d'un statut (payée, impayée, en attente) pour le suivi des paiements.
- **Export PDF** : Les factures peuvent être téléchargées au format PDF pour une distribution facile.

E. **InvoiceLine**

- **Détails des factures** : Cette entité représente les lignes de produits ou services inclus dans une facture.
- **Calcul des montants** : Gestion des quantités, prix unitaires et calculs pour chaque ligne.

F. **Contracts**

- **Archivage** : L'utilisateur peut importer des contrats afin de les centralisé.

Ces fonctionnalités offrent une gestion complète et centralisée des tâches administratives, adaptées aux besoins des micro-entrepreneurs.

2. Défis Techniques

A. Problème avec Flask

- **Problème détaillé** :

  • Flask a été initialement choisi pour sa simplicité et sa légèreté. Cependant, des limitations sont apparues lors de l'utilisation avec Docker. Par exemple :

    - Les logs ne s'affichaient pas correctement, ce qui rendait le débogage difficile.
    - La gestion des extensions pour des fonctionnalités comme l'authentification ou la gestion de la base de données nécessitait des configurations supplémentaires qui n'étaient pas intuitives.

  • Ces obstacles ont ralenti la progression, rendant nécessaire une réflexion sur une alternative plus adaptée.

- **Conséquences** :

  • Perte de temps à tenter de corriger les problèmes de compatibilité et à rechercher des solutions adaptées à Docker.

- **Solution** :

  • Transition vers **NestJS**, un framework plus structuré, qui intègre nativement un système de logging performant et une meilleure gestion des dépendances grâce à son architecture modulaire.

B. Difficulté avec NestJS

- **Problème détaillé** :

  • Bien que NestJS soit puissant et adapté aux projets complexes, il nécessite une montée en compétence importante pour en tirer pleinement parti.

  • Les points de difficulté incluaient :

- La compréhension de son architecture en modules, qui nécessite une structuration stricte du code.
- L'apprentissage des décorateurs et de concepts comme les Providers ou les Guards, propres à NestJS.
- La configuration des middlewares et des outils tiers, comme la gestion des bases de données avec TypeORM ou Sequelize.

• Ces obstacles ont entraîné une baisse de la productivité et des retards dans la mise en œuvre des fonctionnalités.

- **Conséquences** :

• Un temps non négligeable a été consacré à la lecture de la documentation et à l'exploration de tutoriels, au détriment du développement concret des fonctionnalités du projet.

- **Solution** :

• Transition vers **ExpressJS**, un framework déjà maîtrisé, pour garantir un développement rapide tout en bénéficiant de la flexibilité nécessaire pour les besoins du projet.

3. Gestion du projet

Une bonne gestion du projet **µManager** a été essentielle pour garantir une progression fluide et efficace. Cette section présente les fonctionnalités prévues, la priorisation des tâches, et l'approche méthodologique adoptée pour structurer le travail.

Liste des fonctionnalités souhaitées

Pour répondre aux besoins des micro-entrepreneurs, les fonctionnalités suivantes ont été identifiées comme prioritaires :

1. **Authentification et gestion des utilisateurs** :
   - Inscription et connexion sécurisées.
   - Gestion des rôles et permissions.
2. **Gestion des entreprises** :
   - Création et gestion d'entreprises par l'utilisateur.
3. **Gestion des clients** :
   - Ajout, mise à jour, suppression, et consultation des fiches clients.
4. **Facturation** :
   - Génération et gestion des factures avec suivi des statuts (payée, impayée, en attente).
   - Export des factures en PDF.
5. **Gestion des contrats** :
   - Archivage des contrats.
6. **Optimisations diverses** :
   - Ajout de fonctionnalités futures comme un tableau de bord financier des notifications.

Priorisation des tâches

Pour garantir un développement progressif et structuré, les tâches ont été priorisées en fonction de leur importance et de leur dépendance à d'autres composants :

1. **Configuration de l'environnement** :
   - Mise en place des outils nécessaires (Docker, React, ExpressJS).
   - Création du fichier docker-compose pour orchestrer les conteneurs.
   - Initialisation du dépôt Git pour le suivi du code.

2. **Mise en place de la base de données** :

   • Création du schéma relationnel pour les entités principales (Users, Companies, Clients, Invoices, InvoiceLine, Contracts).

   • Initialisation de la base avec **MariaDB** et configuration de phpMyAdmin pour une gestion simplifiée.

3. **Création des entités et de leur logique métier** :

   • Développement des modèles correspondant aux entités principales.

   • Définition des relations entre les tables (ex. : une entreprise peut avoir plusieurs clients).

4. **Développement des routes API** :

   • Implémentation des routes CRUD pour chaque entité (ex. : /users, /clients, /invoices).

   • Gestion de l'authentification avec JWT pour sécuriser les routes sensibles.

5. **Construction du backend** :

   • Intégration des middlewares nécessaires (gestion des erreurs, logging, validation des données).

   • Tests unitaires des fonctionnalités backend pour garantir leur robustesse.

6. **Développement du frontend** :

   • Mise en place des composants React pour chaque fonctionnalité (authentification, tableau de bord, gestion des clients, etc.).

   • Intégration avec l'API backend pour récupérer et afficher les données dynamiquement.

Compétences mises en œuvre

Le projet **µManager** a permis de mobiliser et de développer les compétences suivantes, en lien avec les objectifs pédagogiques de la formation :

C1 : Adapter des applications sur un ensemble de supports (embarqué, web, mobile, IoT…)

- **AC1 : Choisir et implémenter les architectures adaptées**

  • Le choix d'une architecture **API-first** a permis de séparer clairement le frontend (React avec Vite) et le backend (ExpressJS avec TypeScript).

  • Cette approche garantit une modularité et une flexibilité accrues, rendant l'application adaptable à d'autres supports comme les appareils mobiles.

- **AC3 : Intégrer des solutions dans un environnement de production**

  • L'utilisation de **Docker** a permis de créer un environnement de production reproductible et standardisé, minimisant les différences entre les phases de développement et de déploiement.

  • Le fichier docker-compose orchestre efficacement les différents services (backend, base de données, phpMyAdmin), assurant une transition fluide vers un environnement opérationnel.

C2 : Analyser et optimiser des applications

- **AC1 : Anticiper les résultats de diverses métriques (temps d'exécution, occupation mémoire…)**
  - Le projet a été conçu en tenant compte de la performance, notamment en optimisant les requêtes entre le backend et la base de données MariaDB.
- **AC3 : Choisir et utiliser des bibliothèques et méthodes dédiées au domaine d'application**
  - Le choix de **TypeScript** pour renforcer la robustesse du code et réduire les erreurs.
  - Utilisation de bibliothèques spécifiques :
    - **JWT** pour l'authentification sécurisée.
    - **React Router** pour la navigation dynamique dans l'interface utilisateur.
  - Ces outils ont permis de répondre efficacement aux besoins du domaine d'application, tout en garantissant des performances optimales.

C6 : Manager une équipe informatique

- **AC1 : Organiser et partager une veille numérique**

  • Une veille active a été réalisée pour identifier les technologies les mieux adaptées, comme le passage de Flask à NestJS, puis à ExpressJS pour répondre aux contraintes du projet.

  • Bien que réalisé individuellement, le projet a été structuré pour faciliter une éventuelle collaboration en équipe (utilisation de Git pour le versionnement, standardisation via Docker, et séparation claire des responsabilités frontend/backend).

Conclusion et Perspectives

Bilan

Le projet **µManager** a permis de mettre en pratique de nombreuses compétences acquises durant la formation, notamment :

- **Conception d'applications web** : La mise en place d'un projet structuré avec un backend robuste (ExpressJS avec TypeScript) et un frontend dynamique (React).
- **Gestion d'une base de données relationnelle** : Conception et intégration de MariaDB, avec des tables et relations adaptées aux besoins des utilisateurs.
- **Gestion de projet** : Organisation méthodique des tâches grâce à une priorisation claire et une approche API-first.
- **Adaptabilité** : Transition entre différentes technologies backend (Flask → NestJS → ExpressJS) en fonction des besoins et des contraintes du projet.
- **Débogage et tests** : Identification et résolution de problèmes techniques tout en garantissant le bon fonctionnement des fonctionnalités principales.

Les objectifs principaux du projet ont été atteints :

- Une application permettant de gérer les utilisateurs, clients, entreprises, factures, et contrats.
- Une API REST bien structurée, évolutive, et sécurisée via JWT.
- Une interface claire et intuitive pour les utilisateurs.

Améliorations Futures

Bien que le projet réponde déjà aux besoins principaux des micro-entrepreneurs, plusieurs améliorations et extensions pourraient être envisagées :

1. **Création de contrats personnalisés**

• Ajouter une fonctionnalité avancée permettant aux utilisateurs de rédiger des contrats directement dans l'application, avec des modèles pré-remplis adaptés aux différents types de services.

2. **Gestion des micro-entrepreneurs commerçants et libéraux**

• Adapter les fonctionnalités pour répondre aux spécificités des commerçants (gestion des stocks, ventes de produits) et des professions libérales (gestion des honoraires).

• Intégrer des options spécifiques comme le suivi des marges pour les commerçants ou des calculs adaptés aux charges libérales.

3. **Notifications et rappels**

• Ajouter un système de notifications pour :

  - Les rappels de paiement pour les factures en attente.
  - Les échéances contractuelles à venir.
  - Les tâches administratives importantes.

4. **Analyse et suivi des données**

• Enrichir les statistiques disponibles (chiffre d'affaires, clients les plus rentables, suivi des dépenses).

• Proposer un tableau de bord détaillé pour aider les utilisateurs à mieux visualiser la performance de leur activité.

5. **Gestion d'entreprises multiples pour un utilisateur**

• Permettre à un utilisateur de gérer plusieurs entreprises simultanément, avec une séparation claire des données entre les entités.

6. **Amélioration de l'interface utilisateur**

   • Proposer une personnalisation de l'interface selon les préférences de l'utilisateur (choix de thèmes, agencement des tableaux de bord).

   • Optimiser l'interface pour une meilleure compatibilité avec les appareils mobiles.

Ces perspectives visent à rendre l'application encore plus complète et polyvalente, répondant ainsi à un éventail plus large de besoins pour les micro-entrepreneurs.