Contexte:

L'application est une application web qui permet à ses utilisateurs d'enregistrer et consulter leurs dépenses. Elle permet, de manière simple et intuitive, d'ajouter, modifier ou supprimer des dépenses et de les consulter efficacement grâce à ses fonctionnalités de tris, de filtres et de recherche.

Architecture:

L'application se distingue en deux parties:
- Une application Angular (front-end),
- Un serveur node (back-end), réalisé avec le framework NestJS, qui communique avec une base de données PostgreSQL

Les deux communiquent via des requêtes HTTP. Elles sont envoyées depuis le client web grâce au module HTTP d'Angular, arrivent dans les *controllers* où sont définies les routes, et déclenchent les *useCases* associés. La communication avec la base de données se fait grâce aux *repositories* présents dans les *services.*

Objectif:

Le but est de développer une nouvelle partie *admin* qui permettrait aux utilisateurs manager de consulter et d'afficher des statistiques sur les dépenses des employés. La différentiation des rôles est déjà présente mais il faut cloisonner les modules en fonction des autorisations et des rôles, que ce soit au niveau de l'application front-end ou au niveau du serveur.

Description des useCases à implémenter:
- L'utilisateur manager se connecte et s'authentifie. L'application lui donne accès au module admin où il peut accéder à toutes les dépenses de tous les employés.
- L'utilisateur manager accède à la page des statistiques où il retrouve tous les widgets. Chaque widget présente des courbes ou des graphes représentant les statistiques calculées par l'application.
- L'utilisateur peut choisir les données d'entrées de chaque graphe pour préciser les informations qu'il recherche

Exigences fonctionnelles:

Lien vers la partie admin	Ajouter un lien vers la partie admin dans le menu de la navbar	Front
Consultation des dépenses des employés	Retrouver toutes les dépenses de tous les employés et les ranger dans un tableau intégrant des fonctionnalités de tris, filtres et recherche (voir maquette #01 en annexe) dans le composant consult	Front
Statistiques (calculs)	Calculer: - Moyenne des dépenses par mois - Montants totaux par mois - Fréquence par catégorie par mois	Front
Statistiques (visualisation)	Retrouver toutes les statistiques calculées et les afficher, via chart.js, dans le composant charts. Lier la donnée aux graphes pour permettre l'affichage d'informations supplémentaires au survol.	Front
Module d'authentification	Mettre en place une stratégie d'authentification (passport)	Back
Module d'autorisation	Déterminer le rôle de l'utilisateur authentifié. Donner l'accès ou non aux routes en fonction de ses rôles.	Back

Exigences non fonctionnelles:

L'enjeu va être la vitesse de calcul des statistiques et de leur affichage: il faut que les calculs soient optimisés et que la donnée soit bien liée aux graphes pour permettre la meilleure expérience utilisateur possible.

Nouveautés architecturales:

Créer un nouveau module pour la partie admin	Créer un module *admin* dans lequel on y trouvera les composants liés aux nouvelles fonctionnalités. Ce module aura pour dépendance le module *shared* où on y trouve les composants de base pour l'interface utilisateur notamment.	Front
Intégrer chart.js	Créer un nouveau service abstrait de graphes. Ce service doit implémenter une interface, et sera implémentée avec chart.js	Front
Middlewares d'authentification et d'autorisation	Implémenter les middlewares d'authentification (passport) ainsi que les guards pour chaque route. Un resolver pour déterminer les rôles métier de l'utilisateur authentifié est nécessaire	Back
Implémenter les guards sur les routes existantes	Restreindre l'accès aux routes en fonction des rôles. Il faut aussi vérifier et adapter les useCases en fonction des rôles si une route est autorisée à plusieurs rôles différents	Back

Tests et validation:

- Prévoir des tests unitaires pour chaque calcul pour chaque statistique.
- Vérifier que l'interface ne déclenche pas de mauvais calculs (exemple: division par 0)
- Prévoir des sets de tests avec énormément de données et des données manquantes pour identifier la robustesse et la vitesse des calculs
- Tester l'autorisation et le cloisonnement des modules en fonction des rôles