كلية العلوم و التقنيات بطنجة
Faculté des Sciences et Techniques de Tanger
جامعة عبد المالك السعدي
Université Abdelmalek Essaâdi

LST : Ingénierie de Développement d'Applications Informatiques

Module : Développement Web

Département : Génie Informatique

RAPPORT DE MINI-PROJET :

Application Web de Réservation des Ressources d'un Data Center

Sous l'encadrement de :

Pr.AIT KBIR Mhamed

Pr. ELYUSUFI Yasyn

Réalisé par :

***EL HAJIOUI Houssam**

***El Bourmaki Salim**

***Dany Homam**

***Farssi Fatima Zahra**

Table de matière :

1. INTRODUCTION:

1.1 Contexte du projet:

Dans le monde informatique actuel, un Data Center est le cœur névralgique de toute organisation. Il regroupe des serveurs physiques, des routeurs, des commutateurs et des machines virtuelles. Cependant, nous avons remarqué que la gestion de ces équipements coûteux se fait encore souvent de manière artisanale : fichiers Excel partagés, échanges d'emails ou post-its. Cette méthode montre vite ses limites dès que le nombre d'utilisateurs augmente, entraînant erreurs et pertes de temps.

1.2 Objectifs:

Notre projet, **"DCR Management"**, a pour but de remplacer ces méthodes obsolètes par une application Web robuste. L'objectif n'est pas seulement de lister le matériel, mais de rendre le Data Center "intelligent" et accessible. Concrètement, l'application doit permettre à un utilisateur de réserver une ressource en quelques clics, tout en garantissant à l'administrateur une vision claire de l'occupation du parc en temps réel.

1.3 Problématique:

Le défi principal de ce projet réside dans la gestion temporelle : *Comment garantir une allocation optimale des ressources sans créer de conflits ?* Il fallait concevoir un système capable d'empêcher techniquement qu'une même ressource soit réservée par deux personnes sur la même plage horaire (problème d'overlapping), tout en gérant des niveaux de droits très stricts entre les simples utilisateurs et les responsables techniques.

2. ANALYSE ET CONCEPTION :

2.1 Identification des acteurs :

Pour structurer notre application, nous avons identifié quatre profils distincts :

***L'Invité :** Il n'a pas encore de compte. Il peut visiter le catalogue public pour voir le matériel, mais ne peut rien réserver.

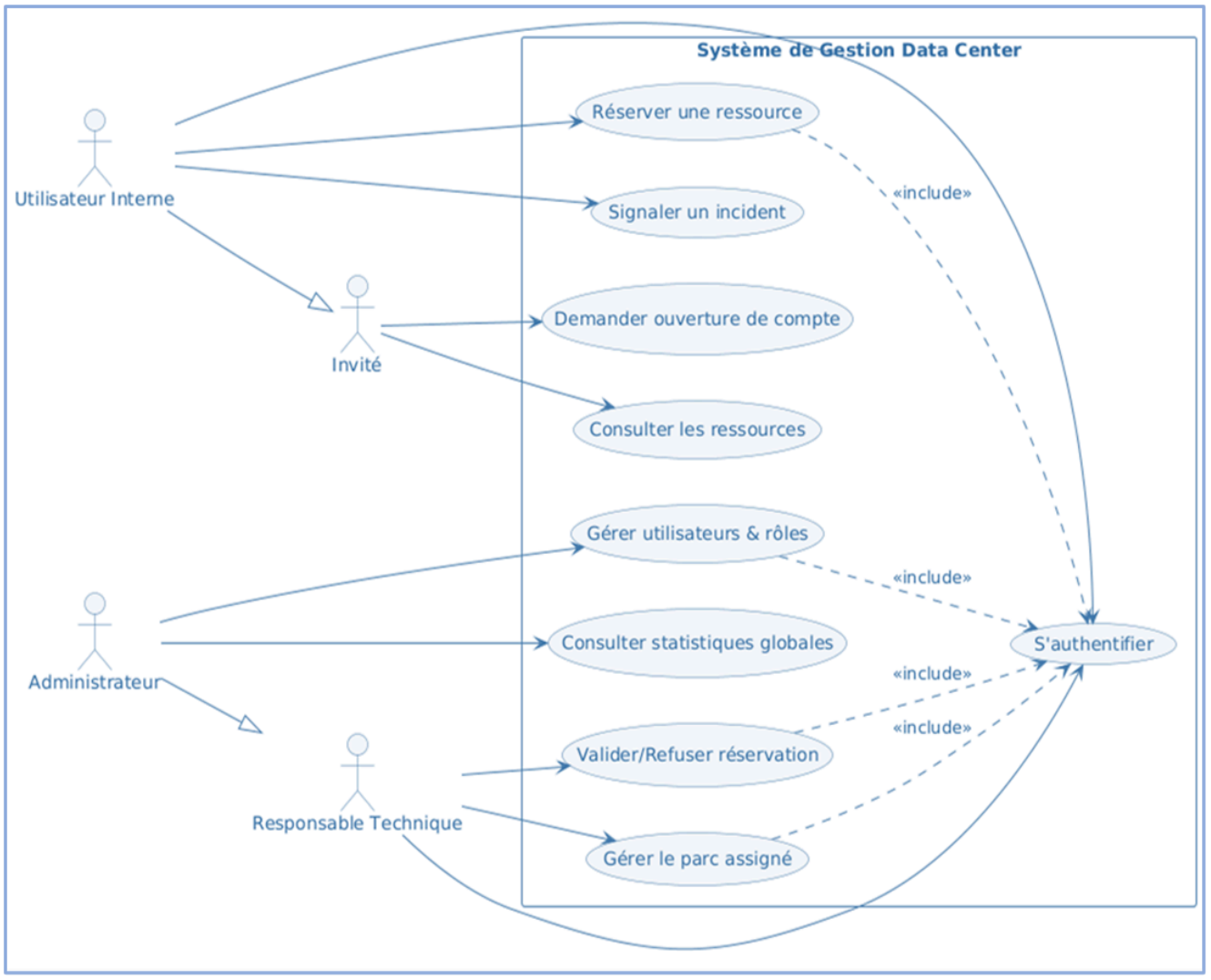
***L'Utilisateur Interne (Enseignant/Ingénieur) :** C'est le cœur de la cible. Il consulte la disponibilité, effectue des demandes et suit leur validation.

***Le Responsable Technique :** Il gère un sous-ensemble du parc (ex: uniquement les serveurs Linux). Il a le pouvoir de valider ou refuser les demandes.

***L'Administrateur :** Il possède tous les droits. Il gère les utilisateurs, crée les catégories de matériel et supervise les statistiques

2.2 Diagramme de Cas d'Utilisation (Use Case)

Ce diagramme synthétise les actions possibles pour chaque acteur au sein du système.



2.3 Modélisation de la Base de Données

La conception de la base de données est l'étape critique qui garantit la cohérence des informations. Nous avons d'abord établi un **Dictionnaire de Données** pour lister les informations à stocker, puis nous avons déduit le **Schéma Relationnel**.

-Dictionnaire des Données (Extrait) Pour répondre aux besoins du Data Center, nous avons identifié les entités suivantes :

- **Utilisateur :** Nom, Email, Mot de passe, Rôle.
- **Ressource :** Nom, Description, Caractéristiques techniques (JSON ou texte), État.
- **Catégorie :** Libellé (ex: Serveur, Réseau).
- **Réservation :** Date de début, Date de fin, Motif, Statut.

-Schéma Relationnel (MLD) Notre base de données MySQL est composée de 4 tables principales reliées entre elles :

1. **Table users** (Gestion des accès)
    - id (Clé Primaire)
    - name (Varchar)
    - email (Varchar, Unique)
    - password (Varchar, haché)
    - role_id (Entier) : Définit si l'utilisateur est Admin (1), Tech (2) ou User (3).
2. **Table categories** (Classification)
    - id (Clé Primaire)
    - name (Varchar) : Ex: "Serveurs", "Machines Virtuelles".
3. **Table resources** (Le Parc Informatique)
    - id (Clé Primaire)
    - name (Varchar) : Nom de la machine (ex: "Srv-Linux-01").
    - category_id (Clé Étrangère) : Lien vers la table catégories.
    - specs (Text/JSON) : CPU, RAM, Disque.

- is_active (Booléen) : Pour mettre une machine en maintenance.

4. **Table reservations** (Le cœur du système)
   - id (Clé Primaire)
   - user_id (Clé Étrangère) : Qui a réservé ?
   - resource_id (Clé Étrangère) : Quelle machine ?
   - start_time (DateTime) : Début du créneau.
   - end_time (DateTime) : Fin du créneau.
   - status (Enum) : 'pending', 'approved', 'rejected'.

Règles de Gestion et Relations

- **Relation 1-N (Un à Plusieurs) :** Une catégorie contient plusieurs ressources, mais une ressource n'appartient qu'à une seule catégorie.
- **Relation 1-N :** Un utilisateur peut avoir plusieurs réservations.
- **Contrainte d'intégrité :** La suppression d'une catégorie est bloquée tant qu'elle contient des ressources (pour éviter les orphelins).

3. CHOIX TECHNOLOGIQUES ET ENVIRONNEMENT

3.1 Backend : Framework Laravel (MVC)

Laravel et sa structure MVC (Modèle-Vue-Contrôleur). nous a permis de séparer proprement le code :

- Les **Contrôleurs** gèrent la logique (vérifier les dates).
- Les **Vues** gèrent l'affichage HTML.
- Les **Routes** dirigent les URL. C'est une architecture professionnelle qui rend le code maintenable et sécurisé par défaut (protection CSRF/XSS).

3.2 Base de données : MySQL & Eloquent ORM

Plutôt que d'écrire du SQL brut, nous avons utilisé **Eloquent**, l'ORM de Laravel. Cela nous permet de manipuler la base de données comme des objets PHP. *Exemple :* Pour récupérer les réservations validées d'un utilisateur, il suffit d'écrire `$user->reservations()->where('status', 'approved')->get()`. C'est plus lisible et moins sujet aux erreurs.

3.3 Frontend : Approche "Responsive"

Conformément aux contraintes du projet, nous avons soigné l'interface pour qu'elle soit moderne et adaptative ("Responsive Design"). L'objectif était que l'application soit utilisable aussi bien sur l'écran large d'un administrateur que sur la tablette d'un technicien en salle serveur. Nous avons structuré notre CSS pour avoir une interface claire, épurée et ergonomique.

4. ARCHITECTURE ET FONCTIONNALITÉS IMPLÉMENTÉES

4.1 Gestion de l'authentification et des accès (Middleware)

La sécurité de l'application repose sur un principe de **filtrage en amont**. Plutôt que de vérifier les droits de l'utilisateur dans chaque fonction de chaque contrôleur (ce qui serait répétitif et source d'oublis), nous avons implémenté des **Middlewares personnalisés**.

Concrètement, le middleware agit comme un "douanier" placé entre la requête HTTP de l'utilisateur et le cœur de l'application. Voici le fonctionnement technique que nous avons mis en place :

1. **Interception du flux :** Lorsqu'un utilisateur tape une URL sensible (ex: `/admin/dashboard`), la requête est interceptée par la méthode `handle($request, Closure $next)` de notre middleware `EnsureUserIsAdmin`.
2. **Vérification des droits :** Le code vérifie deux conditions :
    - L'utilisateur est-il connecté ? (`Auth::check()`)
    - Son `role_id` correspond-il à celui d'un administrateur (ex: `1`) ?
3. **Décision de routage :**
    - **Si les conditions sont remplies :** La requête est transmise au contrôleur via `$next($request)`.
    - **Si l'accès est refusé :** Le script s'arrête immédiatement et renvoie une exception HTTP 403 (Forbidden) ou redirige vers la page d'accueil avec un message d'alerte.

Cette logique est appliquée via le fichier de routes (`web.php`) en utilisant des **groupes de routes**. Cela nous permet de protéger l'intégralité du panneau d'administration en une seule ligne de code, garantissant ainsi une étanchéité parfaite contre la "navigation forcée" (tentative d'accès direct via l'URL).

4.2 Système de réservation et gestion des conflits (Overlapping)

C'est la clé de voûte de l'application. Pour garantir la disponibilité réelle des ressources, nous avons développé une logique stricte pour empêcher les doublons :

Vérification préventive Avant même d'enregistrer une demande, le système interroge la base de données pour détecter tout chevauchement temporel. Nous ne nous contentons pas de vérifier si la ressource est "libre" à l'instant T, mais sur toute la durée de la plage demandée.

Algorithme de détection Nous avons implémenté une condition logique précise pour identifier les conflits. Le système considère qu'il y a chevauchement si : `Date_Début_Demandée < Date_Fin_Existante` **ET** `Date_Fin_Demandée > Date_Début_Existante`. Cette requête permet de couvrir tous les cas de figure (chevauchement partiel au début, à la fin, ou inclusion totale).

Intégrité des données Si cette condition est vérifiée, le contrôleur Laravel bloque instantanément la transaction et renvoie une erreur explicite à l'utilisateur. Cela rend techniquement impossible la double réservation d'un même équipement, assurant ainsi une fiabilité totale du planning.

4.3 Gestion du parc matériel (CRUD)

Pour l'administration, nous avons privilégié la sécurité et la simplicité d'utilisation :

Ajout et validation des données L'ajout de matériel est entièrement sécurisé par les **FormRequests** de Laravel. Le système vérifie automatiquement les données (ex : la RAM doit être un nombre) avant l'enregistrement. Nous avons aussi automatisé le traitement des images (renommage et stockage) pour éviter tout conflit de fichiers.

Gestion des pannes (Mode Maintenance) Plutôt que de supprimer une machine en panne, l'administrateur peut activer le **"Mode Maintenance"**. Cela masque instantanément la ressource pour les réservations utilisateurs, tout en la gardant dans l'inventaire technique.

Suppression sécurisée Pour éviter les erreurs, nous avons bloqué la suppression des équipements actifs. Si un administrateur tente de supprimer un serveur qui a des réservations futures, le système refuse l'action. Cela garantit qu'aucun planning ne soit cassé par inadvertance.

4.4 Système de notifications et suivi

Pour assurer une transparence totale entre les utilisateurs et l'équipe technique, nous avons mis en place un flux de communication visuel et instantané :

Workflow de validation Chaque demande de réservation suit un cycle de vie strict défini en base de données : pending (En attente), approved (Validée) ou rejected (Refusée). Ce statut est modifiable uniquement par les responsables techniques via leur tableau de bord.

Retours visuels (Feedback UI) Côté utilisateur, l'interface a été pensée pour être immédiate : grâce à des codes couleurs standardisés (Jaune pour l'attente, Vert pour l'approbation, Rouge pour le refus), l'utilisateur connait l'état de sa demande en un coup d'œil, sans avoir besoin d'ouvrir chaque dossier.

Traçabilité En cas de refus, le système oblige l'administrateur à fournir un motif. Cette justification est enregistrée et affichée à l'utilisateur, ce qui évite les incompréhensions et permet de garder une trace de chaque décision prise sur le parc.

5. RÉALISATION ET DÉMONSTRATION

5.1 Interfaces Utilisateurs

Figure 1 : Page de Connexion

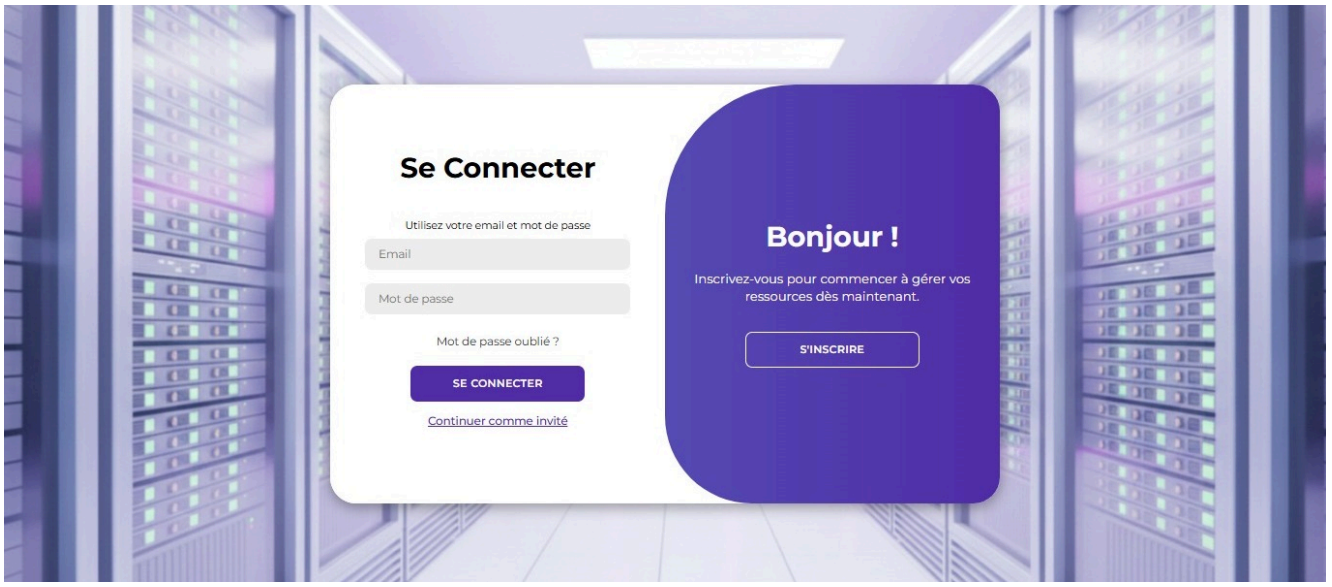
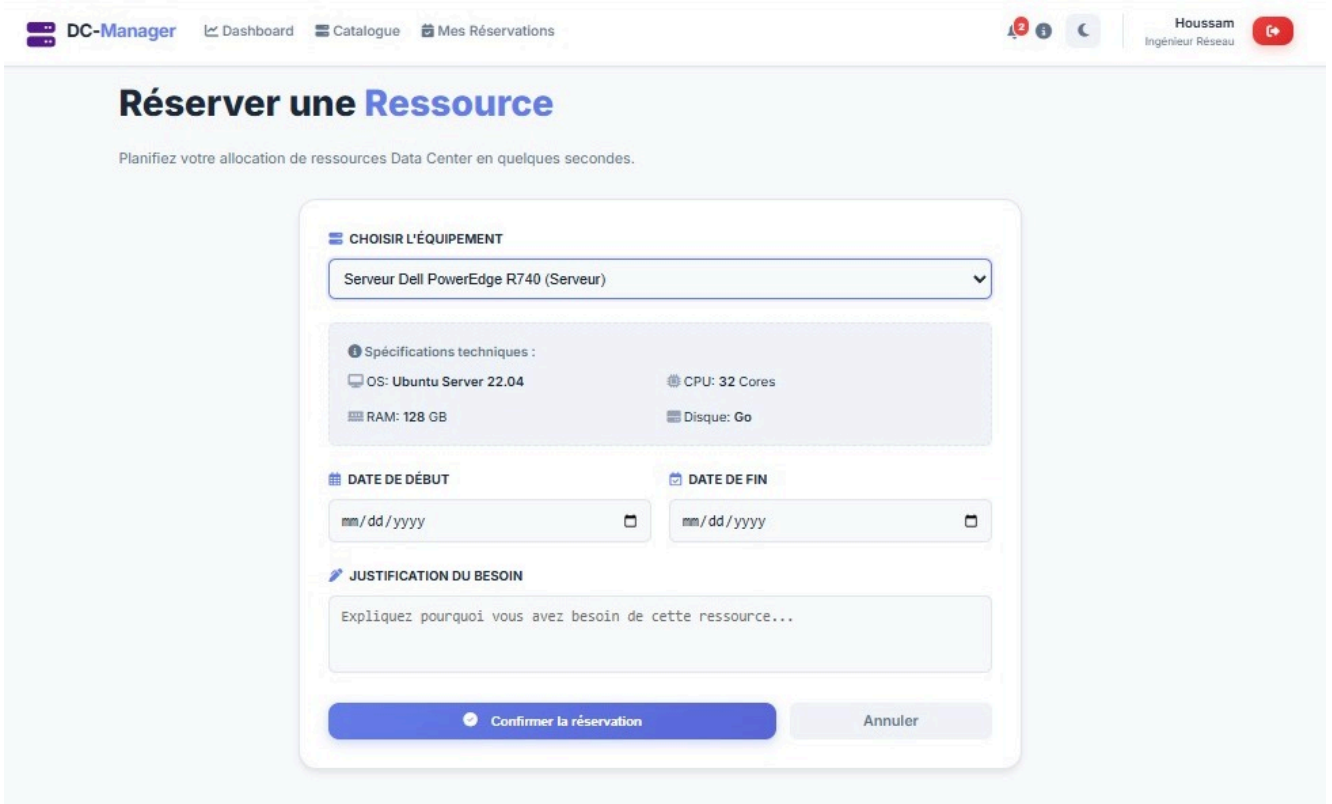


Figure 2 : Le Formulaire de Réservation



5.2 Interfaces Administrateur

Gestion du Catalogue (CRUD)/admin

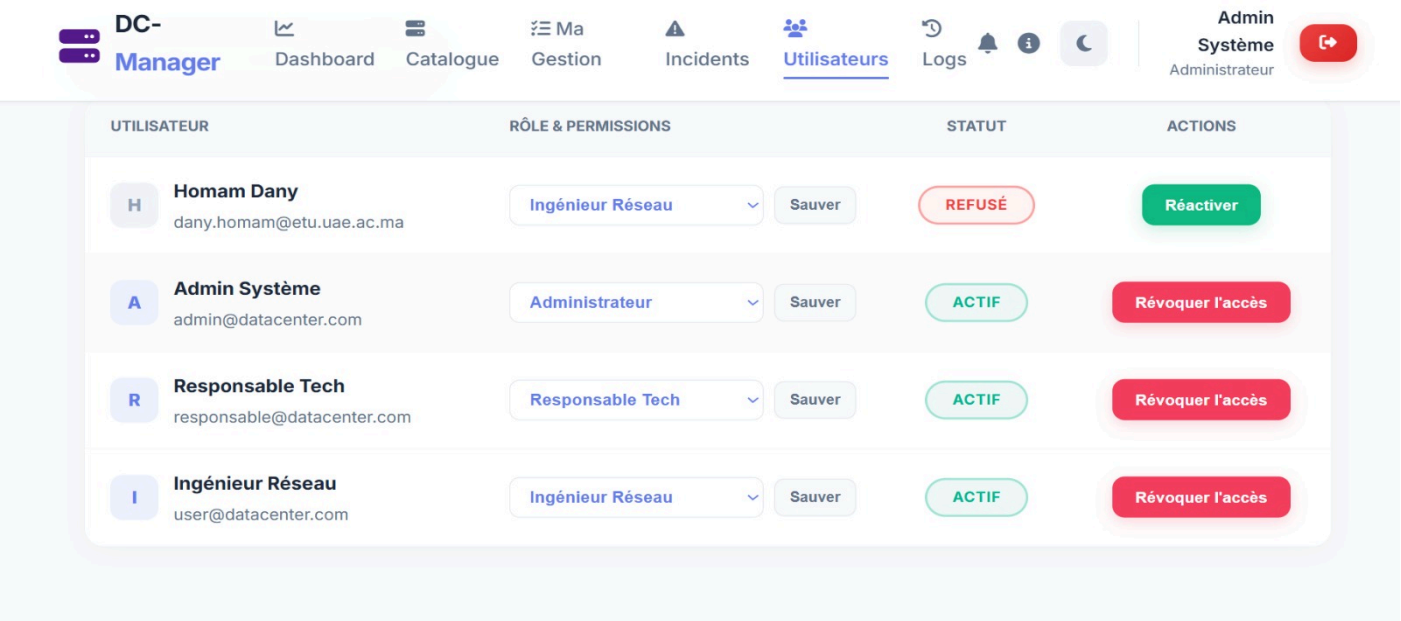
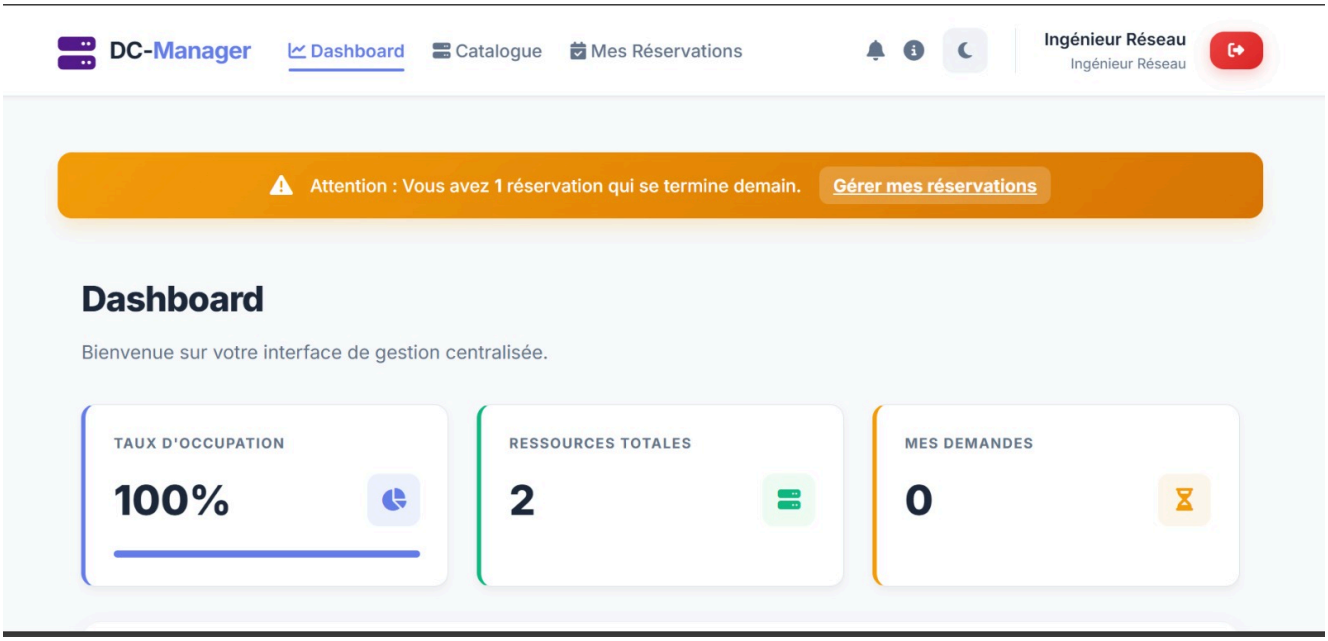


Tableau de Bord (Dashboard)/ingenieur



CONCLUSION

Ce projet "DCR Management" a été une expérience très enrichissante. Il nous a permis de mettre en pratique les concepts théoriques du développement Web (MVC, ORM, Relationnel) dans un contexte réaliste. Nous avons réussi à livrer une application fonctionnelle qui répond à la problématique initiale : simplifier et sécuriser l'accès aux ressources du Data Center. La gestion des conflits de dates (overlapping) a été le défi technique majeur, résolu grâce à la puissance des requêtes Eloquent.

Pour l'avenir, l'application pourrait être améliorée en ajoutant un module d'exportation PDF pour les rapports d'activité, ou une API pour connecter des outils de monitoring externes.