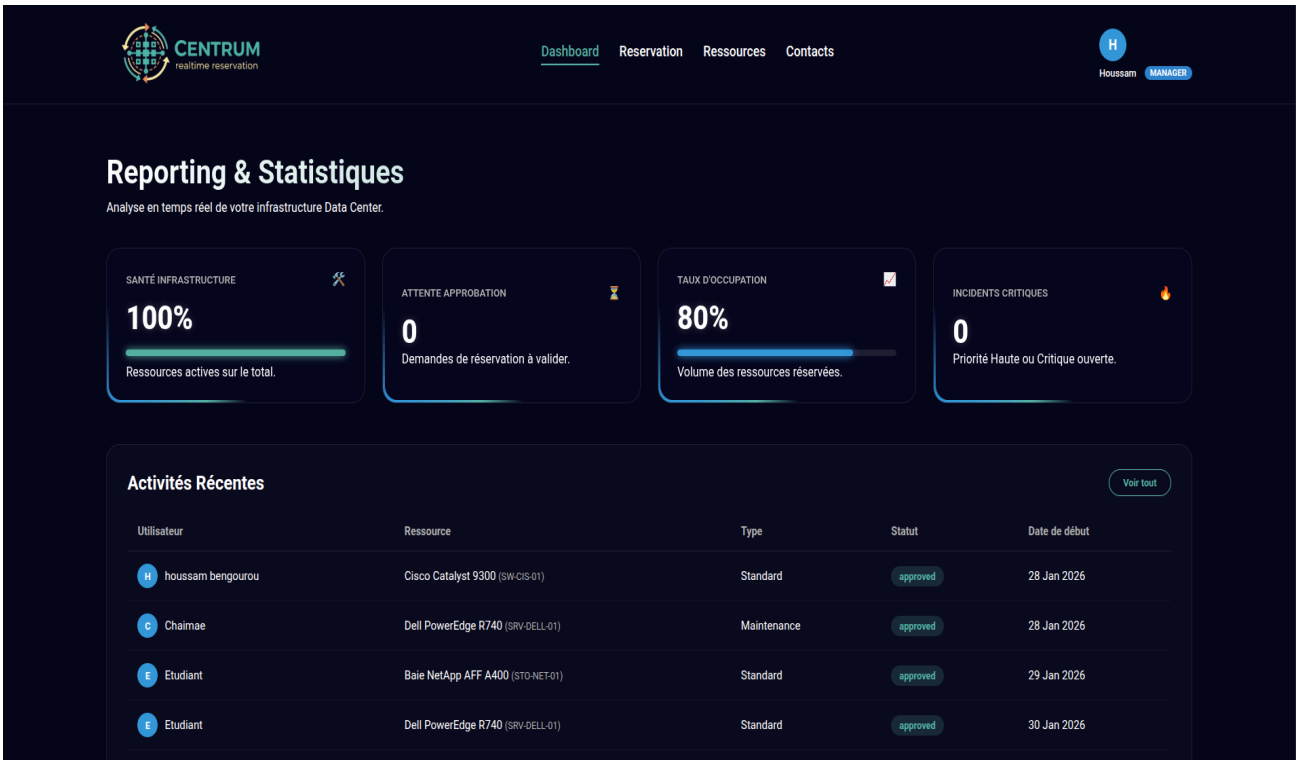
جامعة عبد المالك السعدي
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵄⴰⴱⴷⵍⵎⴰⵍⵉⴽ ⵙⵙⴰⴷⵉ
Université Abdelmalek Essaadi

FACULTE DES SCIENCES ET TECHNIQUES
TANGER
كلية العلوم والتقنيات - طنجة

Rapport de Projet

PROJET : APPLICATION GESTION DATA CENTER(CENTRUM)



Réalisé par :

Alae Jaaouani
chaimae zaki
houssam bengourou
Yassine chenayti

Encadré par :

M.M'hamed AIT KBIR
M.Yasyn EL YUSUFI

Partie 1 : Introduction & Architecture

Responsable : Chaimae

1. Contexte et Problématique

La gestion d'un Data Center universitaire est un défi logistique et technique complexe. Au-delà de la simple maintenance des serveurs, il existe un flux constant de demandes et de mouvements de matériel qui, s'ils sont mal gérés, entraînent une perte d'efficacité majeure.

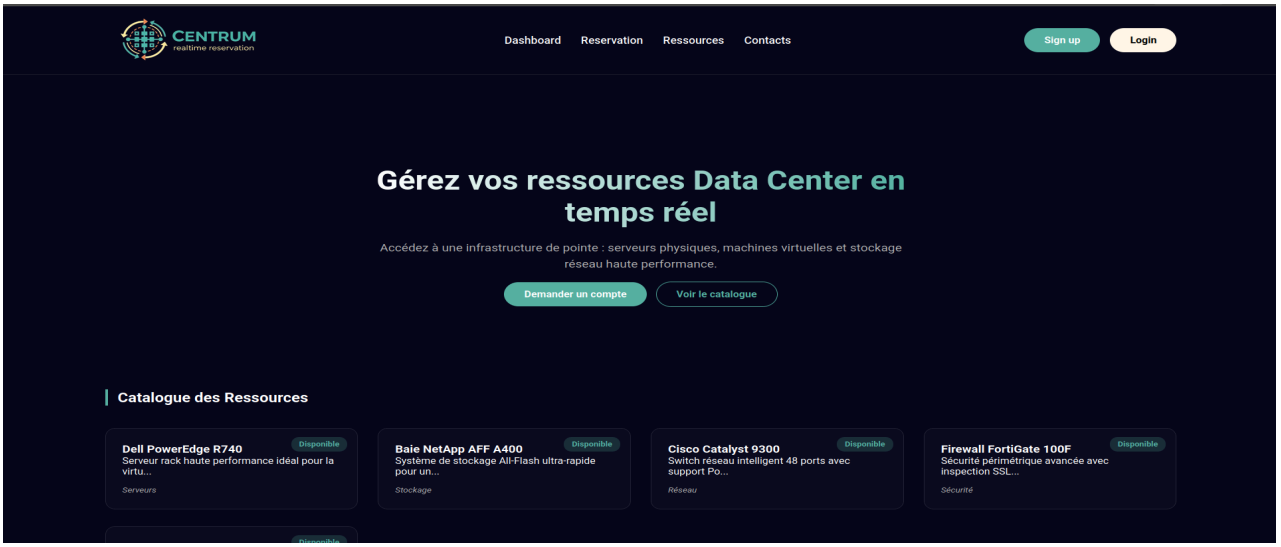
Nous avons identifié trois points de friction critiques dans la gestion actuelle :

1. **La gestion archaïque des incidents :** Le signalement des pannes se fait souvent de manière informelle (oral, emails dispersés), rendant le suivi et la priorisation impossibles pour les techniciens.
2. **L'allocation conflictuelle des ressources :** La réservation de salles ou d'équipements (switchs, câblage, serveurs de test) manque de traçabilité. Cela crée des conflits d'usage (double réservation) et une méconnaissance de la disponibilité réelle du matériel.
3. **L'absence de pilotage par la donnée (Statistiques) :** Les responsables naviguent à l'aveugle". Sans statistiques claires (ex: *Quels équipements tombent le plus souvent en panne ? Quel est le temps moyen de résolution ?*), il est impossible d'anticiper les problèmes ou d'optimiser les budgets de maintenance.

Notre Solution : "Centrum"

Centrum est bien plus qu'un simple outil de ticketing : c'est une véritable plateforme ERP de pilotage pour Data Center. Notre solution centralisée repose sur trois piliers fondamentaux :

1. **Sécurité et Rôles :** Une gestion stricte des accès (Admin, Tech, User) garantissant l'intégrité des données.
2. **Gestion Intelligente :** Un workflow de réservation automatisé qui prévient les conflits en temps réel.
3. **Aide à la Décision :** Un tableau de bord statistique puissant permettant de passer d'une maintenance réactive à une gestion proactive de l'infrastructure.



2. Choix Techniques et Architecture

Pour garantir la robustesse et la pérennité de l'application, nous avons opéré des choix technologiques stratégiques :

2.1. Back-end : Framework Laravel (PHP 8)

Nous avons choisi Laravel pour son architecture **MVC** (Modèle-Vue-Contrôleur) qui permet de séparer clairement la logique métier, la gestion des données et l'affichage.

- **Sécurité native :** Protection contre les failles CSRF, XSS et injections SQL.
- **Eloquent ORM :** Facilite les interactions complexes avec la base de données sans écrire de requêtes SQL brutes.
- **Système d'Authentification :** Gestion robuste des sessions utilisateurs.

2.2. Front-end : CSS Natif (Sans Framework)

Contrairement à la tendance d'utiliser Bootstrap ou Tailwind, nous avons fait le choix audacieux de ne pas utiliser de framework CSS.

- **Objectif Pédagogique :** Démontrer une maîtrise complète du CSS3 (Flexbox, Grid, Responsive Design).
- **Performance :** Le code est plus léger car nous ne chargeons que les styles nécessaires.
- **Identité visuelle unique :** Un design sur-mesure (Dark Mode, effets de verre "Glassmorphism") qui ne ressemble pas aux sites standards.

3. Analyse Fonctionnelle : Diagramme de Cas d'Utilisation

Pour structurer les droits d'accès, nous avons modélisé les interactions via un diagramme de cas d'utilisation UML. Le système repose sur un principe d'héritage des droits : un administrateur est avant tout un utilisateur.

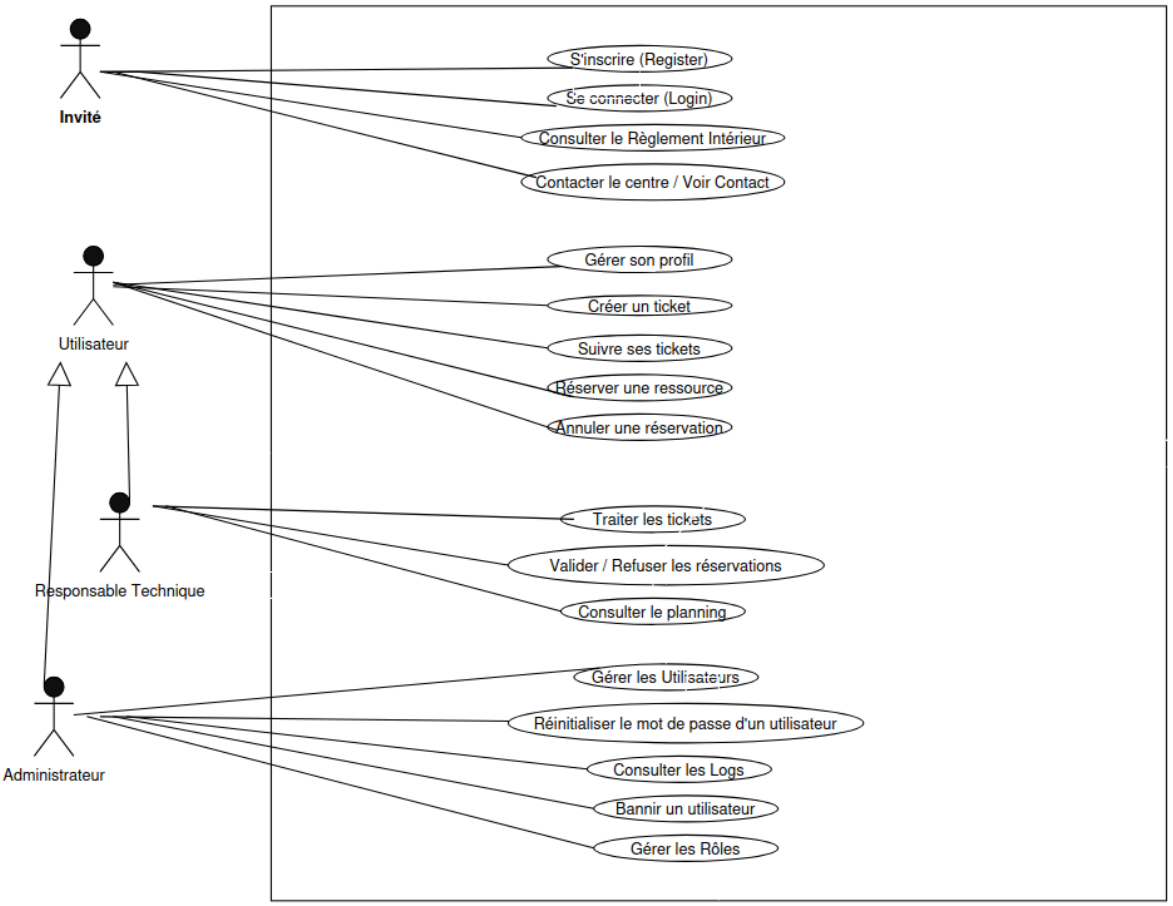


Figure 1 : Diagramme de Cas d'Utilisation du Module d'Administration

Le système identifie 4 acteurs :

1. **Invité :** Accès limité (Inscription, Connexion, Contact, Règlement).
2. **Utilisateur :** Acteur connecté standard (Création de tickets, Profil).
3. **Responsable Technique :** Gestion opérationnelle (Résolution des tickets, Validation des réservations).
4. **Administrateur :** Gestion stratégique (Utilisateurs, Logs, Bannissement, Sécurité).

4. Sécurité et Gestion des Accès (Authentification & Autorisation)

La sécurité de Centrum ne repose pas uniquement sur une page de connexion, mais sur une architecture de Middlewares en cascade définie dans le routeur de Laravel (`web.php`). Nous avons segmenté l'application en trois zones de sécurité distinctes.

4.1. Protection Globale (Middleware `auth`)

La majorité de l'application (Dashboard, Réservations, Support) est protégée par le middleware natif `auth`. Comme visible dans le code, nous regroupons toutes les routes sensibles dans un groupe sécurisé. Si un utilisateur non connecté tente d'accéder à `/dashboard`, le middleware intercepte la requête et le redirige automatiquement vers la page de login.

```
70  Route::middleware('auth')->group(function () {
71
72      // Déconnexion
73      Route::post('/logout', [AuthController::class, 'logout'])->name('logout');
74
```

"Le groupe de routes `auth` protège l'intégralité du tableau de bord et des fonctionnalités internes."

4.2. Gestion des Rôles (RBAC - Role-Based Access Control)

Au-delà de la simple connexion, nous avons implémenté une sécurité granulaire pour protéger l'intégrité des données. Tout le monde ne peut pas modifier le matériel du Data Center.

- ***Lecture (Read) :** Accessible à tous les connectés.*
- ***Écriture (Create/Update) :** Nous utilisons un middleware personnalisé `role:admin,manager`. Cela permet aux Techniciens et Admins d'ajouter du matériel, mais bloque les simples utilisateurs.*
- ***Suppression (Delete) :** Action critique réservée exclusivement à l'Administrateur via `middleware('role:admin')`*

```
110     Route::middleware(['role:admin,manager'])->group(function () {
111         Route::get('/ressources/create', [RessourceController::class, 'create'])->name('ressources.create');
112         Route::post('/ressources', [RessourceController::class, 'store'])->name('ressources.store');
113     });
```

"Utilisation de middlewares personnalisés pour restreindre la création et la suppression de ressources aux seuls personnels autorisés (Admin et Manager)."

4.3. Gestion de la zone "Invité" (`guest`)

Pour éviter des incohérences de session, les routes de connexion et d'inscription sont protégées par le middleware `guest`. Si un utilisateur est déjà connecté, il ne peut pas accéder à ces pages et est redirigé vers son tableau de bord.

5. Module d'Administration : Gestion des Utilisateurs

En tant qu'administrateur, j'ai développé une interface complète (CRUD) pour gérer la vie des comptes utilisateurs.

5.1. Liste et Visualisation

L'interface offre une vue d'ensemble instantanée de tous les utilisateurs inscrits.

Pour faciliter la lecture et la gestion, nous avons intégré **un système de badges visuels** dans la colonne "Rôle". Ce mécanisme permet à l'administrateur de distinguer rapidement les différents niveaux d'accréditation (Administrateurs, Techniciens, Utilisateurs) sans avoir à lire chaque ligne en détail.

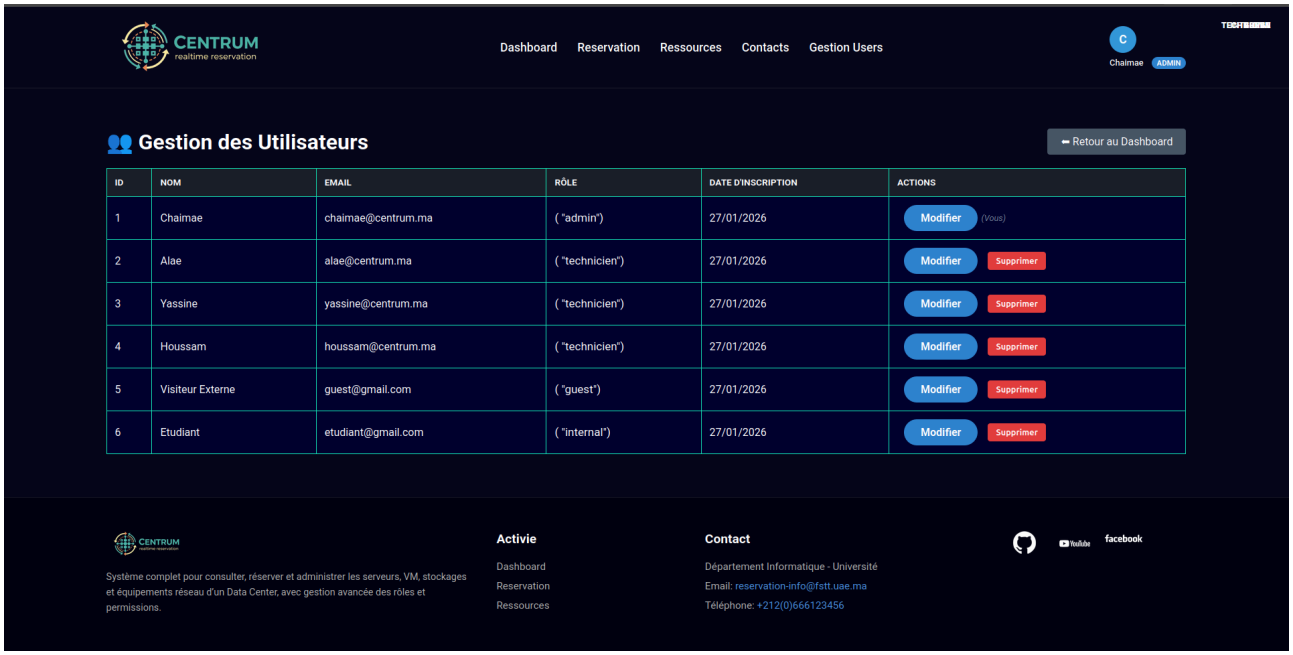


Figure : Tableau de bord d'administration listant l'ensemble des utilisateurs inscrits.

5.2. Modification des Rôles et Sécurité

L'administrateur a le pouvoir de promouvoir un utilisateur (ex: passer un étudiant en "Technicien") ou de le bannir. Cette action est critique : elle met à jour la base de données via une méthode sécurisée dans le contrôleur `UserController`, empêchant un utilisateur de s'attribuer lui-même des droits qu'il ne devrait pas avoir.

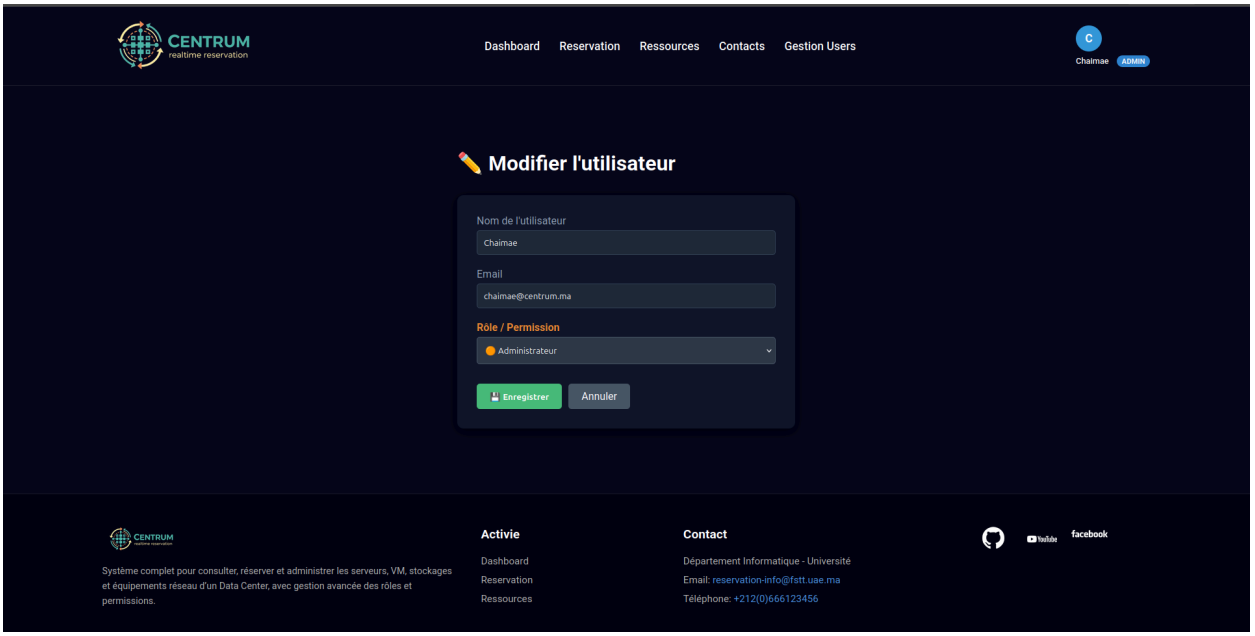


Figure : Interface d'édition de profil permettant l'attribution des rôles et permissions.

6. Module d'Authentification et Sécurité

La sécurité est la pierre angulaire de notre application. Avant d'accéder aux ressources sensibles du Data Center, chaque utilisateur doit être identifié de manière formelle.

6.1. Page de Connexion (Login)

Nous avons développé une interface de connexion moderne et sécurisée. Contrairement aux pages de login classiques, nous avons opté pour un design "Split Screen" (écran scindé) offrant une expérience utilisateur fluide.

- **Sécurité technique :** Le formulaire utilise le middleware d'authentification de Laravel. Il intègre une protection **CSRF** (Cross-Site Request Forgery) pour empêcher les attaques externes.
- **Gestion des sessions :** Une fois connecté, le système identifie le rôle de l'utilisateur (Admin, Tech, User) et le redirige automatiquement vers son tableau de bord approprié.

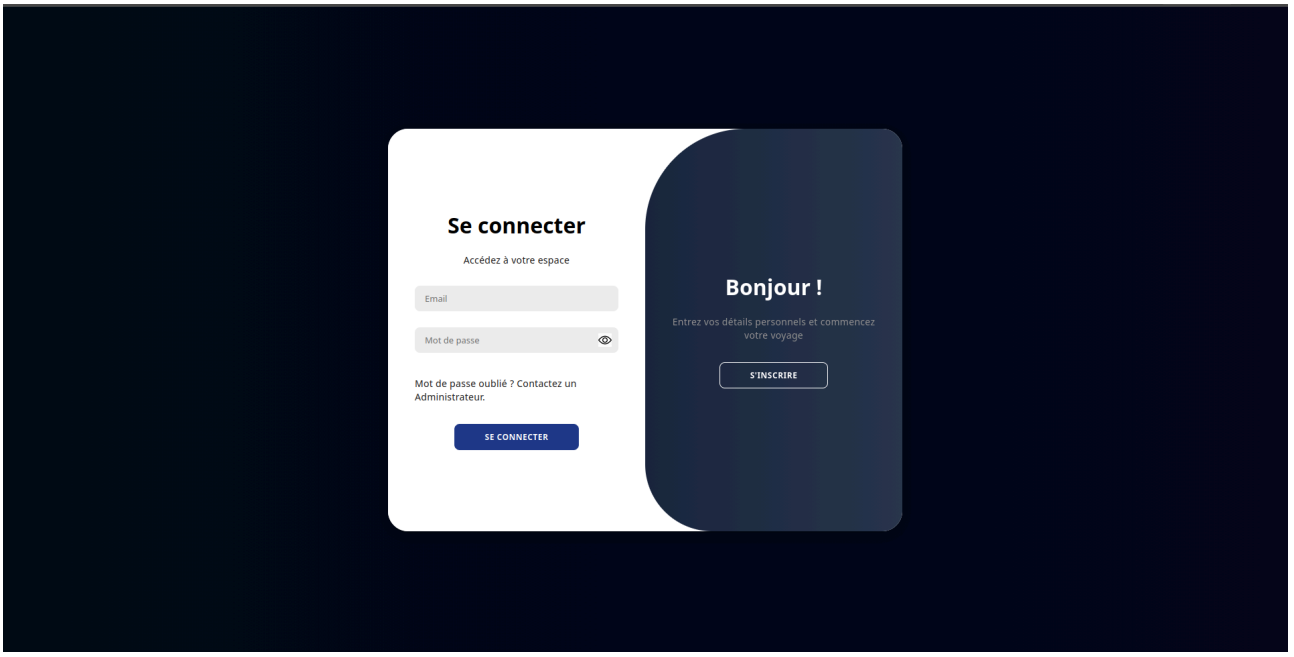
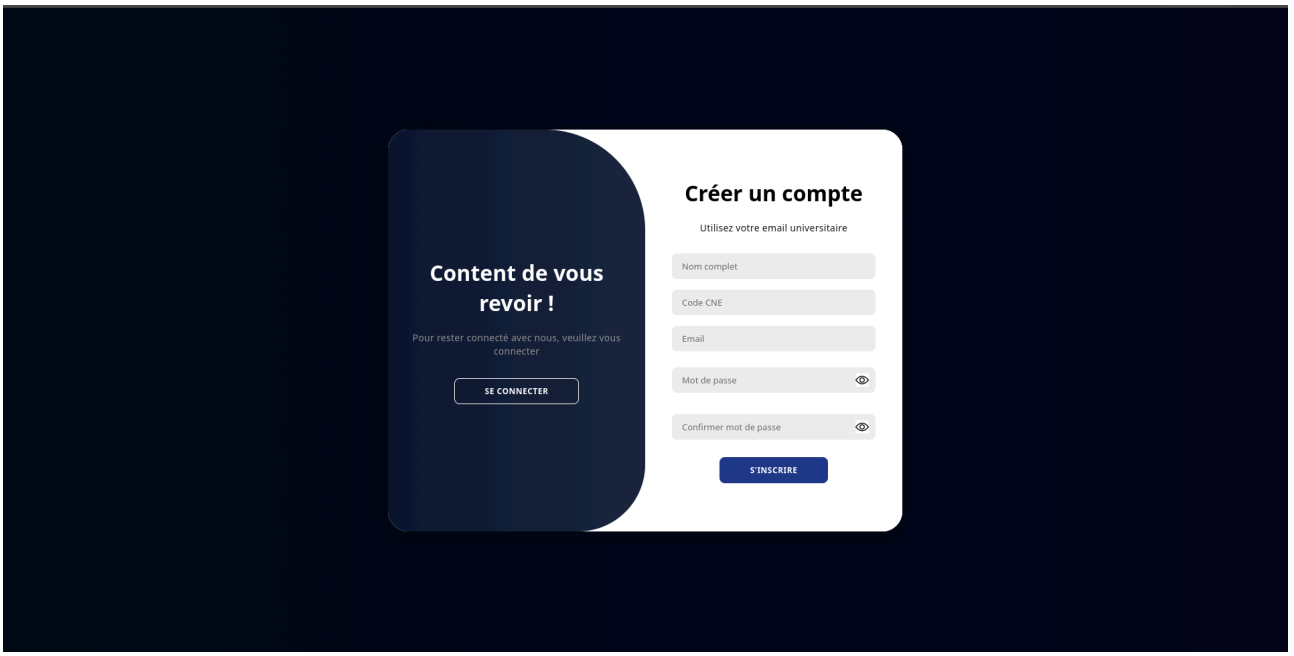


Figure : Interface d'authentification sécurisée.



7. Communication et Support Technique

Pour garantir la transparence et la réactivité au sein du Data Center, nous avons mis en place un espace dédié à la communication entre les utilisateurs et l'équipe technique.

7.1. Présentation de l'Équipe et Règlements

Les utilisateurs ont accès à une page "Contacts" qui présente l'organigramme technique du Data Center. Chaque membre (Administrateur, Technicien) est affiché avec son rôle et ses coordonnées.

De plus, cette section met en avant le **Règlement Intérieur** (via le bouton "Voir le Règlement"). Cela permet de s'assurer que tous les utilisateurs connaissent les normes de sécurité et de bonne conduite avant de réserver une ressource.

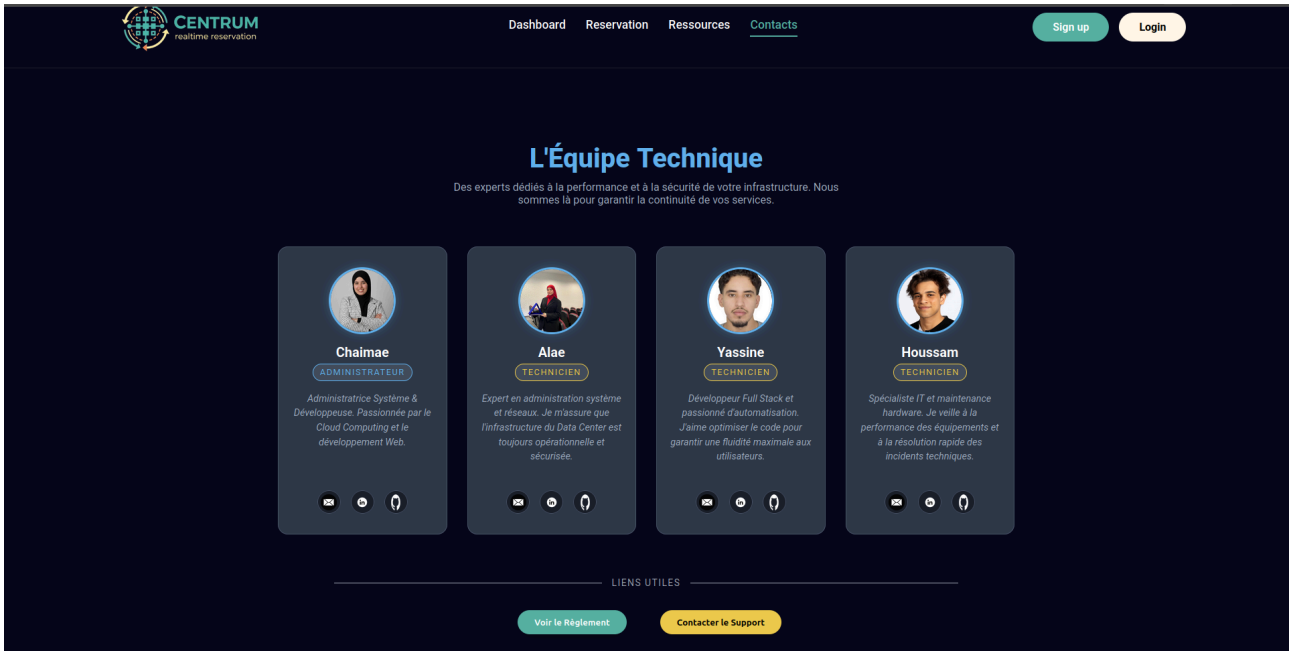


Figure : Page de contact présentant l'équipe technique et l'accès au règlement

Figure : Page du Règlement Intérieur détaillant les normes de sécurité et de confidentialité.

7.2. Gestion et Signalement des Incidents

Dans un environnement critique comme un Data Center, la rapidité d'information est vitale. Nous avons intégré un module de **Signalement d'Incidents**.

- **Fonctionnement :** Si un utilisateur constate une panne (serveur hors ligne, problème de climatisation, etc.), il peut remplir un formulaire dédié.
- **Traitement :** L'alerte est immédiatement enregistrée en base de données et visible par les administrateurs et techniciens, qui peuvent alors intervenir rapidement pour résoudre le problème.

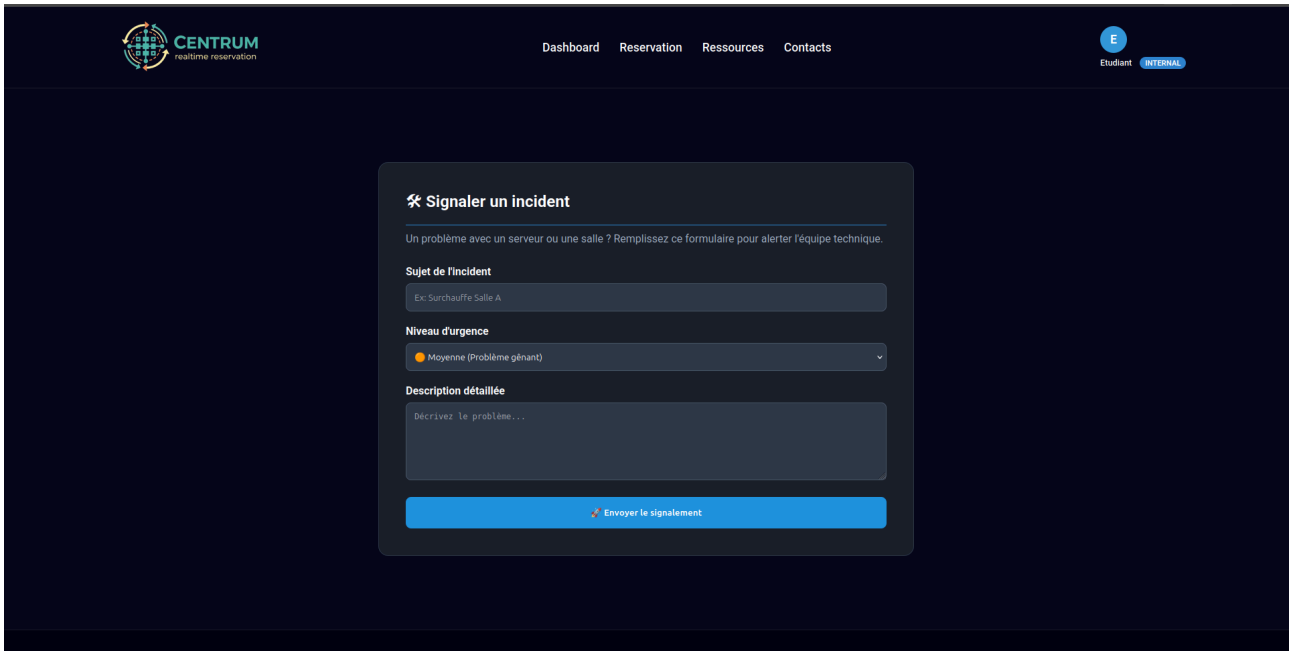


Figure : Formulaire de signalement d'un incident technique avec gestion des priorités.

partie 2 :Conception et Gestion technique des Ressources

responsable: Alae Jaaouani

2.1.Introduction

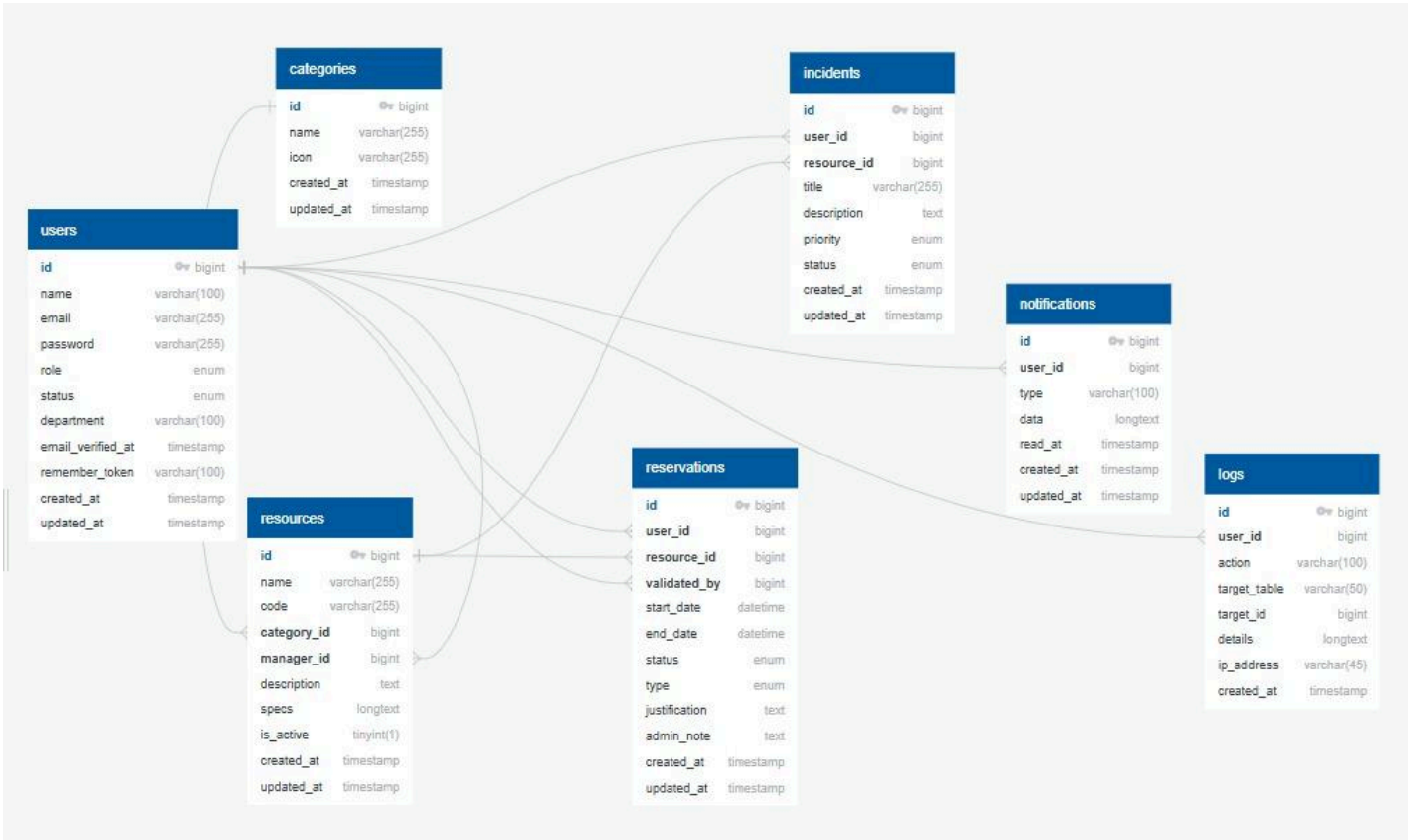
La gestion des ressources constitue le socle opérationnel de cette plateforme de Data Center. L'objectif de cette partie est de centraliser et d'automatiser le cycle de vie de chaque équipement informatique, depuis son intégration dans l'inventaire jusqu'à sa mise à disposition pour les utilisateurs.

Cette section détaille l'architecture technique choisie pour garantir une gestion fluide du parc. Nous aborderons la structuration des données par typologie, l'implémentation d'un système d'états dynamiques (Disponibilité/Maintenance), ainsi que les mécanismes de sécurité applicative permettant de protéger l'intégrité des ressources critiques.

2.1. Architecture des Données (Modèle Logique de Données - MLD)

Pour assurer la persistance et l'intégrité des informations de l'application, j'ai conçu une base de données relationnelle structurée autour de 7 entités principales.

Présentation du Schéma :



Architecture de la base de données et relations:

La persistance des données est assurée par une base de données relationnelle dont la structure a été pensée pour garantir l'intégrité et la traçabilité des ressources.

A. Association Ressources et Catégories Chaque matériel est rattaché à une catégorie unique via la clé étrangère **`category_id`**. Cette structure a été mise en place pour permettre un classement logique du parc (informatique, mobilier, etc.) et optimiser les requêtes de filtrage lors de la recherche de matériel disponible.

B. Attribution de la responsabilité (Manager) La relation entre **`ressources`** et **`users`** (via **`manager_id`**) permet d'assigner chaque objet à un agent spécifique. Ce choix technique est indispensable pour définir qui est responsable de la validation des réservations et du suivi de la maintenance pour chaque ressource.

C. Intégrité des flux de réservation et d'incidents

- Réservations : La table **`reservations`** lie un utilisateur et une ressource. J'ai configuré les relations pour qu'une ressource ne puisse pas être supprimée si des réservations actives y sont rattachées (intégrité référentielle).
- Signalement d'incidents : Lorsqu'un utilisateur crée un incident, le lien direct avec **`resource_id`** permet de remonter l'historique des pannes sur un matériel précis. Cela aide à décider si une ressource doit être définitivement réformée ou simplement réparée.

D. Journalisation (Logs) Le système utilise une table de logs qui pointe vers les autres entités via les champs **`target_table`** et **`target_id`**. Cette approche permet de garder une trace de chaque modification (UPDATE/DELETE) effectuée sur les ressources par les administrateurs.

2.2.Gestion Technique des Ressources

Ce module constitue le cœur opérationnel du projet, permettant une administration complète du parc informatique. Il assure l'inventaire précis du matériel (Serveurs, Stockage, Réseau) tout en gérant leur cycle de vie via des états dynamiques comme "Disponible" ou "Maintenance". L'architecture a été conçue pour différencier les accès : les administrateurs contrôlent l'inventaire tandis que les étudiants bénéficient d'une interface simplifiée pour leurs réservations.

A- Analyse de la Typologie du Matériel

le parc est segmenté en 5 catégories stratégiques. Cette classification permet une organisation granulaire des ressources du Data Center.

Serveurs : Puissance de calcul brute.

Stockage : Gestion de la donnée.

Réseau : Flux et connectivité.

Sécurité : Protection périmétrique (Firewalls).

Virtualisation : Ressources logiques (VMs).

B-Présentation de l'Interface d'Inventaire

Cette page est le centre de contrôle du matériel. Elle permet de visualiser et de gérer l'ensemble du parc informatique en un coup d'œil mais pour l'admin.

car par exemple pour le client on a cette interface :

Les points clés du tableau :

- **Visibilité immédiate :** Le tableau liste chaque matériel avec son nom, sa catégorie (Serveur, Stockage, Réseau) et une brève description technique.
- **Gestion des statuts :** Un badge coloré (ex: Turquoise pour "Disponible") indique instantanément si la ressource est prête à être utilisée ou non.
- **Actions rapides :** Pour chaque ligne, trois boutons permettent de gérer le matériel :
    - Détails : Pour voir la fiche complète.
    - Modifier : Pour mettre à jour les informations (ex: changer la RAM).
    - Supprimer : Pour retirer le matériel du parc.
- **Ajout de matériel :** Le bouton brillant "+ NOUVELLE RESSOURCE" permet d'accéder directement au formulaire de création pour agrandir l'inventaire.

Pourquoi cette interface ?

L'objectif est de permettre à l'administrateur de trouver une information en moins de 3 secondes grâce au contraste des couleurs et à l'organisation épurée des colonnes.

c. Gestion des Droits et Vues Différenciées

“L'une des forces du système est de proposer une interface personnalisée selon le rôle de l'utilisateur. Cela garantit que les actions sensibles (ajout, modification, suppression) sont réservées au personnel autorisé”

D.Focus Technique : Pilotage des États (Disponibilité vs Maintenance)

Le système de gestion des ressources intègre une logique de contrôle de disponibilité en temps réel. Cette fonctionnalité est cruciale pour garantir qu'aucune ressource indisponible ne puisse être sollicitée par un utilisateur.

Architecture Logique (Backend)

L'état de chaque matériel est piloté par un attribut booléen `is_active` au sein de la table `resources`. Ce champ fait office de "verrou logique" :

- **Valeur 1 (True) :** Ressource opérationnelle, exposée aux utilisateurs pour réservation.
- **Valeur 0 (False) :** Ressource en maintenance, masquée ou verrouillée pour toute action utilisateur.

```
<td>
    @if($ressource->is_active)
        <span class="status-badge approved">Disponible</span>
    @else
        <span class="status-badge refused">Maintenance</span>
    @endif
</td>
```

Traduction Visuelle Dynamique (Frontend)

Pour assurer une lecture rapide par l'administrateur, j'ai implémenté une logique de rendu conditionnel dans la vue Blade. Le système génère automatiquement des composants visuels distincts en fonction de l'état .

Sécurité et Intégrité du Workflow

La gestion de l'état dépasse le simple affichage. Elle est intégrée à la logique métier pour empêcher les réservations invalides. Si une ressource est basculée en Maintenance.

```
@auth
    @if(auth()->user()->role !== 'guest')
    <td class="actions-cell">
        <div style="display: flex; gap: 8px; align-items: center;">
            {{-- Détails --}}
            <a href="{{ route(name: 'ressources.show', parameters: $ressource->id) }}" title="Voir" style="color: var(--teal-button);">
                <i class="fas fa-eye"></i> Détails
            </a>
```

Partie 3 : Gestion des Réservations et Algorithme (Backend)

Responsable : Yassine Chenayti

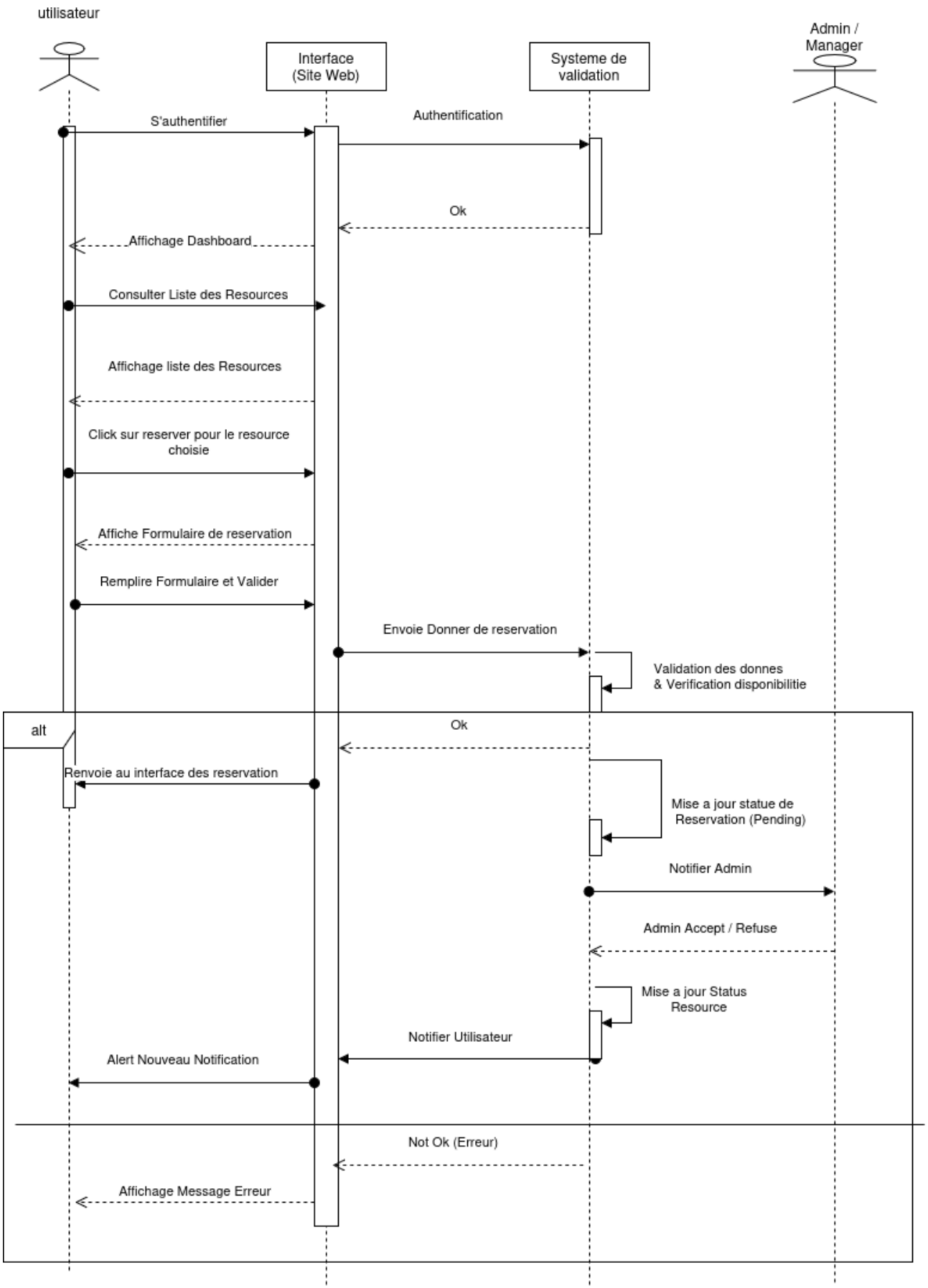
3.1. Introduction

Pour moi, cette partie était la plus technique du projet. Contrairement à une simple gestion de stock, gérer des réservations demande de manipuler le **temps**. Mon objectif principal était d'assurer l'intégrité des données : il fallait absolument empêcher que deux étudiants puissent réserver la même machine sur le même créneau, ce qui aurait rendu l'application inutilisable.

3.2. Le Parcours de Réservation

Avant de coder, j'ai défini les étapes logiques d'une réservation pour être sûr de ne rien oublier. Voici le fonctionnement que j'ai mis en place :

1. **La demande :** L'étudiant choisit une ressource. J'ai utilisé des champs `datetime-local` dans le formulaire pour récupérer des dates précises.
2. **La vérification automatique :** Dès l'envoi du formulaire, mon code vérifie si le créneau est libre (voir l'algorithme ci-dessous).
3. **L'attente :** Si c'est libre, la réservation est créée mais reste en statut `pending` (En attente).
4. **La décision :** Le Manager reçoit une alerte et peut soit valider, soit refuser la demande.


utilisateur
Interface (Site Web)
Systeme de validation
Admin / Manager
S'authentifier
Authentification
Ok
Affichage Dashboard
Consulter Liste des Resources
Affichage liste des Resources
Click sur reserver pour le resource choisie
Affiche Formulaire de reservation
Remplire Formulaire et Valider
Envoie Donner de reservation
Validation des donnes & Verification disponibilitie
alt
Ok
Renvoie au interface des reservation
Mise a jour statue de Reservation (Pending)
Notifier Admin
Admin Accept / Refuse
Mise a jour Status Resource
Notifier Utilisateur
Alert Nouveau Notification
Not Ok (Erreur)
Affichage Message Erreur

3.3. L'Algorithme de Disponibilité (Le défi technique)

C'était la partie la plus complexe. Au début, je pensais devoir gérer plein de cas différents avec des `if` (si la date est dedans, si elle dépasse, si elle englobe, etc.). C'était trop lourd et source d'erreurs.

En faisant des recherches, j'ai compris qu'il était plus simple d'utiliser une logique inversée SQL. J'ai donc implémenté une fonction `isAvailable` directement dans le modèle `Reservation`.

Comment ça marche ? Au lieu de chercher quand c'est libre, je cherche s'il existe **une seule** réservation qui pose problème. Pour qu'il y ait un conflit, il suffit que deux conditions soient réunies :

1. Le *nouveau début* est avant l'ancienne *fin*.
2. La *nouvelle fin* est après l'ancien *début*.

Mon code dans le Modèle (`App\Models\Reservation.php`) :

```php
public static function isAvailable($resourceId, $start, $end, $ignoreId = null)
{
    return !self::where('resource_id', $resourceId)
        ->whereIn('status', ['pending', 'approved'])
        ->where(function ($query) use ($ignoreId) {
            if ($ignoreId) {
                $query->where('id', '!=', $ignoreId);
            }
        })
        ->where(function ($query) use ($start, $end) {
            $query->where('start_date', '<', $end)
                  ->where('end_date', '>', $start);
        })
        ->exists();
}
```

Grâce à cette requête Eloquent, le système renvoie `false` s'il y a le moindre chevauchement, ce qui me permet de bloquer la réservation et d'afficher un message d'erreur à l'utilisateur.

3.4. Validation et Système de Notifications

Une fois que l'algorithme a validé les dates, il reste la validation "humaine".

Gestion des statuts J'ai choisi d'utiliser des méthodes simples dans mon contrôleur pour changer le statut (`approved` ou `rejected`). Pour le refus, je ne voulais pas simplement annuler la demande. J'ai ajouté une étape où le manager doit écrire une justification. J'ai fait cela via une fenêtre modale (Popup) pour que ce soit plus ergonomique qu'une nouvelle page.

Les Notifications en JSON Pour les notifications, je ne voulais pas créer une table complexe avec plein de colonnes vides. J'ai opté pour une colonne `data` de type JSON. Cela me permet de stocker des infos variables comme : `{"message": "Votre réservation est validée", "reservation_id": 42}`

C'est très pratique car si demain je veux ajouter d'autres infos dans la notification, je n'ai pas besoin de modifier la structure de ma base de données.

3.5. Le Planning Visuel (Gantt Custom) (Front End)

Pour afficher les créneaux occupés, je n'ai pas voulu utiliser de librairie complexe (comme FullCalendar). J'ai préféré créer mon propre système pour qu'il soit léger et sur-mesure.

1. La Logique (Maths & CSS) Le défi était de transformer une heure (ex: 10h30) en position sur l'écran. J'ai utilisé une règle de trois simple : je convertis l'heure de début et la durée de la réservation en pourcentages par rapport à la journée totale.

- *Exemple :* Si la journée fait 10h et que je réserve à la 5ème heure, la barre commence à `left: 50%`.

2. Adaptation Mobile Afficher une journée entière sur un petit écran de téléphone rendait le planning illisible. J'ai résolu ce problème avec le scroll horizontal (`overflow-x: auto`). Sur mobile, le planning garde une taille fixe et l'utilisateur glisse simplement son doigt pour voir la suite de la journée.

Inspiration: Youtube: **Simple Responsive Gantt Chart With HTML CSS**

partie 4: Interface Utilisateur (UI/UX) & Statistiques

Responsable : HOUSSAM BENGOUROU

Le défi du "NO framework"

Contrairement aux développements classiques qui s'appuient sur Bootstrap ou Tailwind CSS, la contrainte majeure de ce projet était de réaliser une interface **100% CSS natif**. En tant que responsable UI/UX, ma mission était double : garantir une esthétique moderne ("Dark Theme") tout en assurant une structure de code maintenable pour que Chaimae, Alae et Yassine puissent intégrer leurs fonctionnalités sans casser le design.

Charte Graphique et Architecture CSS

Pour répondre aux exigences de performance et d'identité visuelle, j'ai opté pour une approche modulaire basée sur les variables CSS et les nouvelles normes de mise en page.

Identité Visuelle (Dark Mode)

J'ai défini une palette de couleurs centralisée dans le `:root` du fichier `style.css` pour assurer la cohérence sur toutes les pages. L'ambiance sombre a été choisie pour correspondre à l'univers "Tech / Data Center.

- **Background :** `#05051a` (Bleu nuit profond pour réduire la fatigue visuelle).
- **Action Color :** `#56b3a3` (Teal/Turquoise pour les boutons et indicateurs positifs).
- **Alerts :** Un système de couleurs sémantiques (Rouge pour les incidents, Jaune pour les attentes).

```
:root {
   --bg-color: #05051a;
   --teal-button: #56b3a3;
   --cream-button: #fff9e6;
   --text-gray: #b0b0b0;
   --border-line: rgba(255, 255, 255, 0.1);
}
```

Mise en page : Flexbox & Grid

L'absence de Bootstrap m'a obligé à reconstruire un système de grille réactif :

- **Flexbox :** Utilisé pour la barre de navigation (`.navbar`), l'alignement des boutons et les formulaires. Cela permet de gérer l'espacement dynamique entre les éléments.
- **CSS Grid :** Utilisé pour le Dashboard (`.stats-grid`) et le catalogue (`.resource-grid`). J'ai utilisé la syntaxe `grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));` qui permet aux cartes de se redimensionner automatiquement sans media queries excessives.

```
.navbar {
   display: flex;
   justify-content: space-between;
   align-items: center;
   padding: 20px 8%;
   border-bottom: 1px solid var(--border-line);
}
```

```
.stats-grid {
   display: grid;
   grid-template-columns: repeat(auto-fit, minmax(260px, 1fr));
   gap: 25px;
}
```

Architecture Blade (Layout Master)

Pour éviter la duplication de code HTML, j'ai conçu un template maître `layout.blade.php`.

- Il contient la structure fixe : `<head>`, `header` (Navbar), et `footer`.
- Il utilise la directive `@yield('content')` pour injecter le contenu spécifique des autres développeurs.
- **Avantage :** Si Chaimae ajoute un lien dans le menu, ou si je modifie le logo, le changement se répercute instantanément sur les 10+ pages du site.

```
<main>
      @yield('content')
   </main>
```

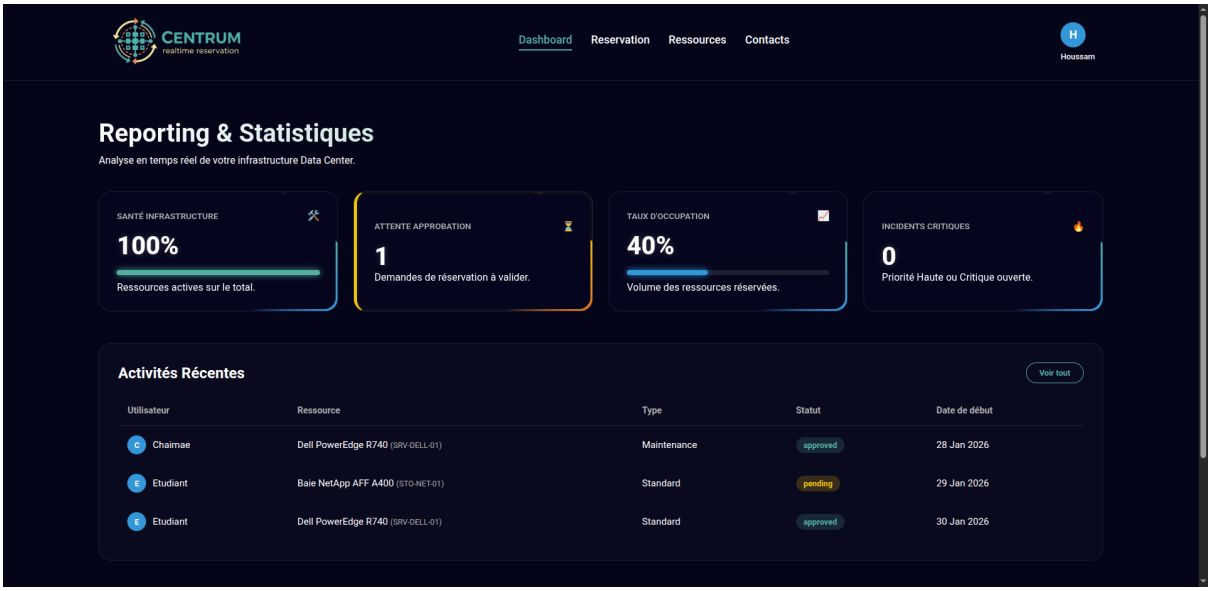
Le Dashboard Analytique

Le tableau de bord (`dashboard.blade.php`) est le cœur névralgique pour l'administrateur. Il ne se contente pas d'afficher des données, il les interprète visuellement.

Indicateurs Clés de performance

Via le contrôleur, nous récupérons des agrégations de données (Count, Where) pour afficher :

1. **Santé Infrastructure :** Le ratio `(Ressources Actives / Total)`.
2. **Taux d'Occupation :** Pourcentage de matériel actuellement réservé.
3. **Alertes :** Nombre d'incidents critiques ouverts.



Design Dynamique et Animations

J'ai implémenté une logique conditionnelle dans les vues Blade pour attirer l'attention de l'administrateur uniquement quand c'est nécessaire :

- Si des réservations sont en attente, la carte devient jaune (`warning-card`).
- Si des incidents critiques sont détectés, la carte clignote en rouge (`danger-card`) grâce à une animation CSS personnalisée `@keyframes pulse-red`.

```
.stat-card.warning-card::before {

   background: conic-gradient(transparent, #f1c40f, #e67e22, transparent
30%);

}

.stat-card.danger-card::before {
```

```
    background: conic-gradient(transparent, #e74c3c, #c0392b, transparent 30%);

}
```

```
@keyframes pulse-red {

    0% { box-shadow: 0 0 0 0 rgba(231, 76, 60, 0.4); }

    70% { box-shadow: 0 0 0 10px rgba(231, 76, 60, 0); }

    100% { box-shadow: 0 0 0 0 rgba(231, 76, 60, 0); }

}
```

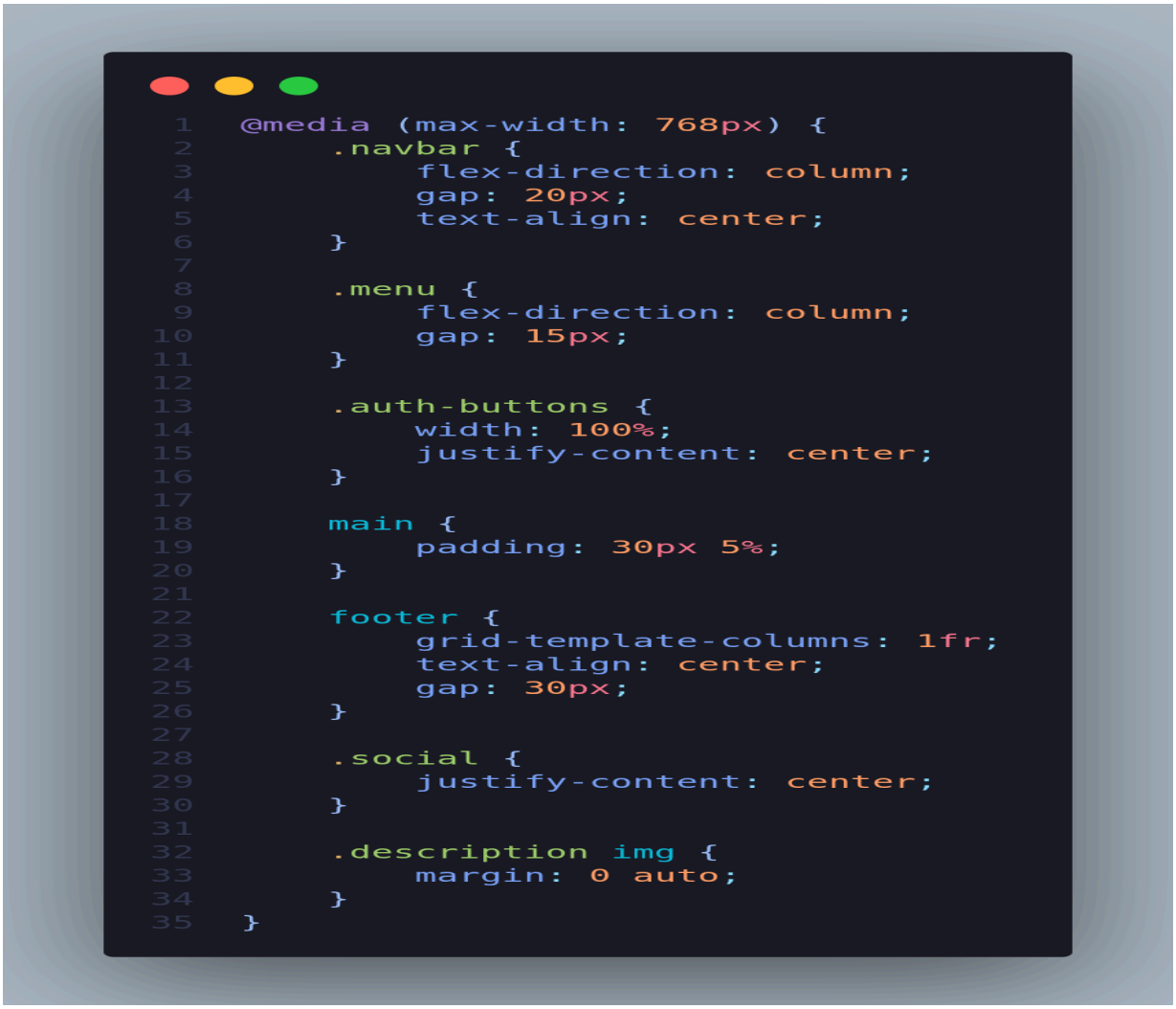
J'ai également créé un effet de "Bordure Tournante" (`rotateGradient`) en CSS pur (via `conic-gradient`) sur les cartes de statistiques pour donner un aspect futuriste et vivant à l'interface, sans utiliser de JavaScript lourd.

```
@keyframes rotateGradient {

    from { transform: rotate(0deg); }

    to { transform: rotate(360deg); }

}
```

Expérience Utilisateur (UX) et responsivité

Le site a été conçu selon le principe du *Mobile Friendly*.

- **Navigation :** Sur mobile (`max-width: 768px`), la Flexbox de la navbar change de direction (`flex-direction: column`) pour empiler le logo et le menu verticalement, rendant les liens tactiles plus accessibles.



```css
@media (max-width: 768px) {
    .navbar {
        flex-direction: column;
        gap: 20px;
        text-align: center;
    }

    .menu {
        flex-direction: column;
        gap: 15px;
    }

    .auth-buttons {
        width: 100%;
        justify-content: center;
    }

    main {
        padding: 30px 5%;
    }

    footer {
        grid-template-columns: 1fr;
        text-align: center;
        gap: 30px;
    }

    .social {
        justify-content: center;
    }

    .description img {
        margin: 0 auto;
    }
}
```

- **Feedback Visuel :** Chaque interaction utilisateur est accompagnée d'une transition CSS (0.3s). Par exemple, sur la page d'accueil (`welcome.blade.php`), les badges de disponibilité ("Disponible" en vert, "Occupé" en rouge) permettent à l'étudiant de scanner l'état du parc en une fraction de seconde.

```
.status-available { background: rgba(86, 179, 163, 0.2); color: #56b3a3; }
.status-busy { background: rgba(231, 76, 60, 0.2); color: #e74c3c; }
```

Conclusion Générale

La réalisation de ce projet de gestion de ressources pour Data Center a permis de répondre à une problématique centrale : la centralisation et la sécurisation du suivi des équipements informatiques. En développant cette solution, l'objectif a été de transformer une gestion potentiellement complexe en un système automatisé, fiable et intuitif.

Sur le plan technique, ce travail a nécessité la mise en place d'une architecture robuste basée sur le framework Laravel, garantissant ainsi la flexibilité du modèle de données et la sécurité des accès. L'implémentation du système de rôles et la gestion dynamique des états du matériel (Disponible/Maintenance) assurent une intégrité totale du parc, prévenant ainsi toute erreur opérationnelle de la part des utilisateurs.

Ce projet a permis de consolider des compétences avancées en développement Full-Stack, notamment sur :

- **La conception de bases de données** relationnelles structurées et évolutives.
- **La logique métier** et la sécurisation des flux de réservation en temps réel.
- **L'expérience utilisateur (UX)** via des interfaces adaptées aux besoins spécifiques de chaque profil.

En conclusion, cette plateforme dépasse le simple stade de l'inventaire pour devenir un véritable outil de pilotage. Les bases posées permettent d'envisager des évolutions futures, telles que l'intégration d'un système de monitoring en temps réel ou l'automatisation de l'allocation des ressources directement depuis l'interface.

“Nous adressons nos remerciements les plus sincères à M. M'hamed AIT KBIR et M. Yasyn EL YUSUFI, nos encadrants de projet, pour leur disponibilité, leurs conseils précieux et la confiance qu'ils nous ont témoignée tout au long de ce travail”