Back-end EMS - Kali

1. **Gestion des erreurs :**
   - Remplacer les `console.error` par `console.log` dans les blocs `catch`. (Si je vois ça à l'exam, je me dis tout de suite que c'est tonton qui a fait ça car les console.error provoquent des erreurs dans certains environnement Node).
2. **Validation des inputs:**
   - Appliquer `trim()` et des conditions pour la récupération des entrées.
   - Exemple : Dans `createMission`, vérifier que `title` et `description` ne sont pas vides après avoir appliqué `trim()`.
   - Importance : Ces vérifications sont essentielles, surtout pour les requêtes POST et PUT/PATCH pour assurer la sécurité et l'intégrité des données.

```
export const createMission = async (req, res) => {
  const { title, description, staffId, startDate, endDate } =
req.body;

  // Application de trim() et vérification des champs vides
  if (title.trim() === "" || description.trim() === "") {
    return res.status(400).json({ message: "Veuillez remplir
tous les champs!" });
  }

};
```

1. **Organisation du code:**
   - Séparer la connexion à la base de données dans un fichier de configuration distinct, plutôt que de la laisser dans `server.js`.

- Avantage : Cela rend le code plus propre.

2. **Modules :**
   - Remplacer `bodyParser.json()` par `express.json()` et `express.urlencoded`.
   - Raison : Les versions récentes d'Express n'ont plus besoin de `bodyParser`, ce qui réduit la dépendance à des modules externes (ton node_modules est moins lourd donc + performant).
3. **Commentaires et Documentation :**
   - Le code est bien très bien commenté.
4. Fragmentation**:**
   - Extraire la logique des e-mails dans un dossier `utils` et appeler ces fonctions chaque fois qu'un e-mail doit être envoyé.
   - Remarque : Bien que `adminController` soit volumineux, il semble adapté au contexte de ton projet.
5. **Sécurité :**
   - Aucune faille de sécurité majeure ou critique j'ai pu voir.
   - La gestion de l'authentification est bien réalisée.
6. **Modèles:**
   - Les modèles sont bien conçus avec des références (`ref`) appropriées.
   - Tu utilises aussi les `$gte` et autres opérateurs mongoose donc c'est top!

Conclusion :

- **Note** : **14,66/20 selon la grille du jury (car il n'y a pas de vérif d'inputs avec ça donne 15,66).**

Très bon travail honnêtement, c'est très bien 👏🏿

- Les axes d'améliorations c'est la fragmentation du code (mail, controller qui fait +500 lignes et DBConexion et **sécurité des inputs**)

Si tu veux aller encore plus loin dans le back, je te conseille de créer un dossier "data" dans ton dossier "config" où tu mettras toutes les données JSON de tes collections MongoDB. Tu pourras les initialiser au démarrage de ton serveur au cas où il ne récupère pas les données. Cela peut également servir de sauvegarde et de back-up.