PROJET BACKEND NODE.JS & MONGODB

Habit Tracker API

Membre du groupe:

- ➢ Florient Gael KALUMUNA
- ➢ KHEFFACHE Ines
- ➢ HADJ SADOK Aya
- ➢ TOURATIER Felix
- ➢ IZARGUI Jad
- ➢ GOBRON Antoine

Sommaire:

1. Introduction:

Le projet **Habit Tracker API** s'inscrit dans le développement d'une application moderne de suivi d'habitudes, permettant aux utilisateurs de **créer**, **organiser** et **analyser** leurs routines quotidiennes. Dans un contexte où l'amélioration du bien-être et de la productivité devient un enjeu majeur, ce type d'outil offre une approche structurée et personnalisée pour accompagner l'utilisateur dans l'atteinte de ses objectifs.

L'API repose sur une stack technologique robuste et largement utilisée dans l'industrie : **Node.js** et **Express** pour le serveur, ainsi que **MongoDB** et **Mongoose** pour la gestion et la modélisation des données. Cette combinaison technologique a été choisie pour sa flexibilité, sa performance et sa capacité à évoluer facilement.

Chaque étudiant a contribué à un ensemble de routes spécifiques : **gestion des utilisateurs**, **gestion des habitudes**, **enregistrement des logs**, **statistiques**, **analyses avancées**. Ce découpage a permis une collaboration efficace tout en garantissant la cohérence générale du projet.

Ce rapport présente l'architecture du projet, les modèles de données mis en place, ainsi que les routes implémentées par chaque étudiant, offrant une vue d'ensemble précise et structurée du backend développé.

2.Objectifs généraux:

- **Concevoir une architecture backend claire, modulaire et évolutive**

Mettre en place une structure de fichiers professionnelle : routes, modèles, middlewares, configuration et interface de test.

- **Utiliser une base de données NoSQL moderne**

Exploiter la flexibilité des documents MongoDB et la puissance des agrégations pour produire des statistiques avancées.

- **Fournir une interface de test simple et efficace**

Permettre de tester toutes les routes via une page HTML, sans nécessiter d'application front-end avancée.

3.Besoins fonctionnels et non-fonctionnels:

3.1Besoins fonctionnels:

ID	Besoin fonctionnel	Description
1	Gestion des utilisateurs	Créer, rechercher, modifier et supprimer des utilisateurs.
2	Gestion des habitudes	Créer, consulter, filtrer, modifier et archiver des habitudes liées à un utilisateur.
3	Classification des habitudes	Associer chaque habitude à une catégorie.
4	Gestion de la fréquence des habitudes	Définir la fréquence (quotidienne, hebdomadaire, personnalisée…) pour chaque habitude.
5	Enregistrement des logs d'habitudes	Enregistrer, pour une date donnée, si l'habitude a été réalisée ou non.
6	Consultation de l'historique	Consulter l'historique des logs d'une habitude sur une période donnée.
8	Calcul des indicateurs de suivi	Calculer les séries de jours consécutifs réalisés (current streak, longest streak).
9	Statistiques globales par utilisateur	Fournir des statistiques globales (nombre d'habitudes, taux de complétion, progression…).
10	Statistiques par catégorie	Générer des stats agrégées par catégorie (nombre d'habitudes, complétions, popularité…).
11	Export / synthèse des données	Fournir des données synthétiques (JSON) pour un tableau de bord ou un export.
12	Gestion des notifications / rappels	Créer, consulter, activer / désactiver des rappels liés à une habitude.
13	Interface de test de l'API (HTML)	Proposer une page web simple permettant de tester les principales routes de l'API.

14	Recherche avancée avec filtres et pagination	Permettre de rechercher utilisateurs ou habitudes avec filtres + pagination + tri.
15	Gestion des erreurs côté API	Retourner des réponses JSON structurées en cas d'erreur (validation, route introuvable, serveur).

3.2Besoins non fonctionnels:

ID	Besoin non fonctionnel	Description
1	Performance	Les principales routes doivent répondre en un temps raisonnable pour des volumes de données usuels.
2	Scalabilité	L'architecture doit permettre d'ajouter facilement de nouvelles routes, modèles ou modules.
3	Modularité du code	Le code doit être organisé par modules (routes, modèles, middlewares, config) pour faciliter la maintenance.
4	Lisibilité et maintenabilité	Respect d'une structure claire, nommage cohérent, séparation des responsabilités.
5	Portabilité	Le projet doit pouvoir être exécuté sur différentes machines avec un simple npm install + .env adapté.
6	Utilisation de technologies standard	Stack Node.js / Express / MongoDB / Mongoose pour garantir compatibilité et support communautaire.
7	Robustesse face aux entrées invalides	L'API doit gérer proprement les IDs invalides, les champs manquants, les formats incorrects.

4. Répartition du travail par étudiant

Étudiant 1: User Management:

Objectif: Gérer les utilisateurs : création, recherche, modification et statistiques.

1️⃣POST /api/users/register

Création d'un nouvel utilisateur (validation email + password).

2️⃣GET /api/users/search

Recherche avancée + pagination + tri.

3**PUT** /api/users/:id

Mettre à jour un utilisateur (username)

4GET /api/users/:id/stats

Agrégation complexe : Statistiques utilisateur avec $lookup et $project

<u>Étudiant 6:</u>User Management:

1 GET /api/users/stats/global

Statistiques utilisateurs (total, croissance, utilisateurs actifs).

2 GET /api/users/import

Importer des utilisateurs depuis initial-users.json

3GET /api/users/stats/export

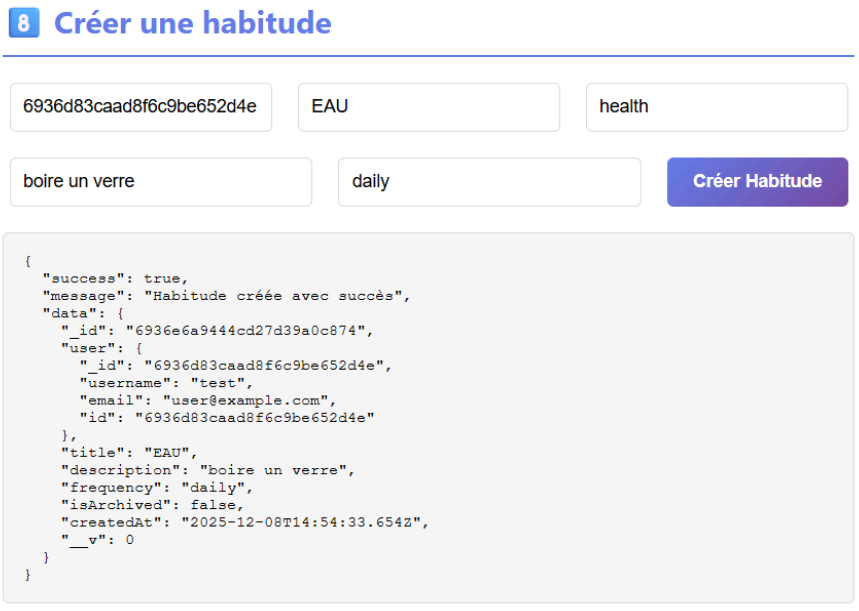
Exporter les statistiques dans un fichier

Étudiant 2 : Habit Management

Objectif: Création, recherche et analyse des habitudes

1POST /api/habits

Créer une habitude.



2️⃣ GET /api/habits/filters

Recherche filtrée par user, catégorie, fréquence, recherche textuelle.

3️⃣ GET /api/habits/analytics/categories

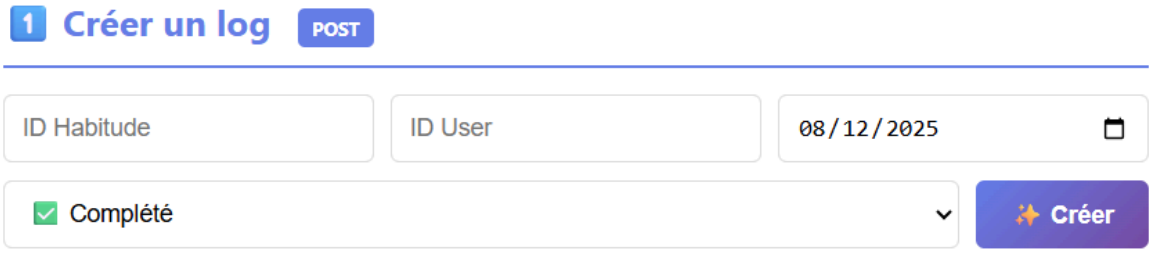
Agrégations MongoDB : total des habitudes par catégorie

<u>Étudiant 3</u>: Habit Logs & Tracking

Objectif: Enregistrer la réalisation d'une habitude + récupérer l'historique.

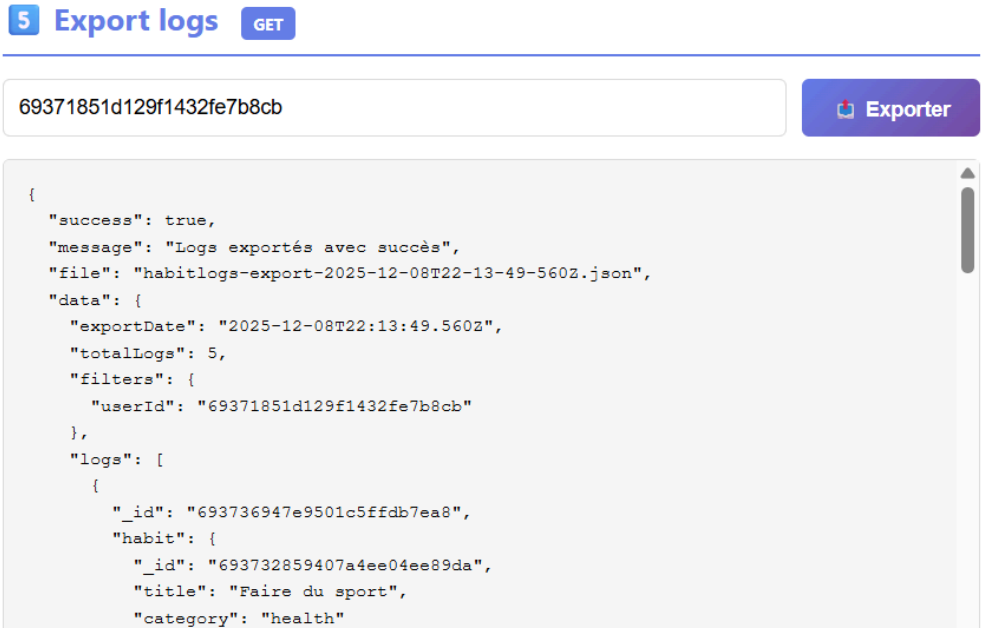
1️⃣POST /api/habits/:habitId/logs

Enregistrer un log (journée réalisée).



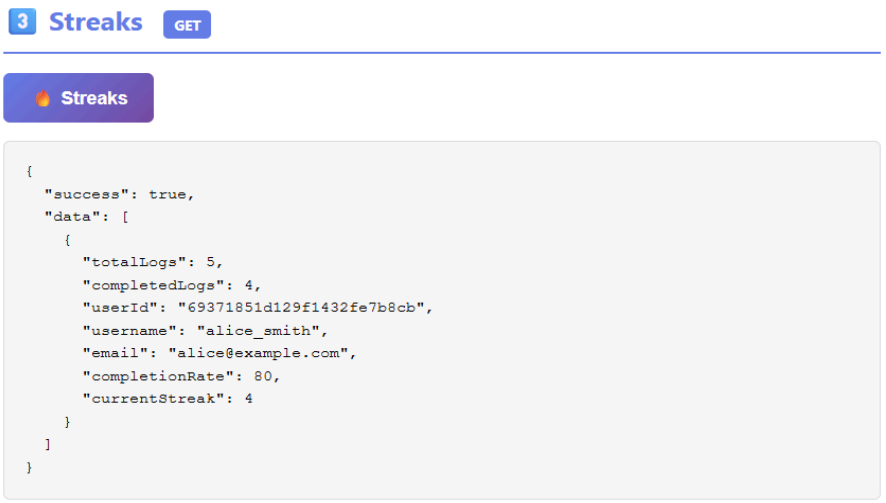
2️⃣GET /api/habits/:habitId/logs

Filtres disponibles : date min/max, tri, limite



3️⃣GET /api/habits/:habitId/logs/streaks

Agrégation MongoDB :

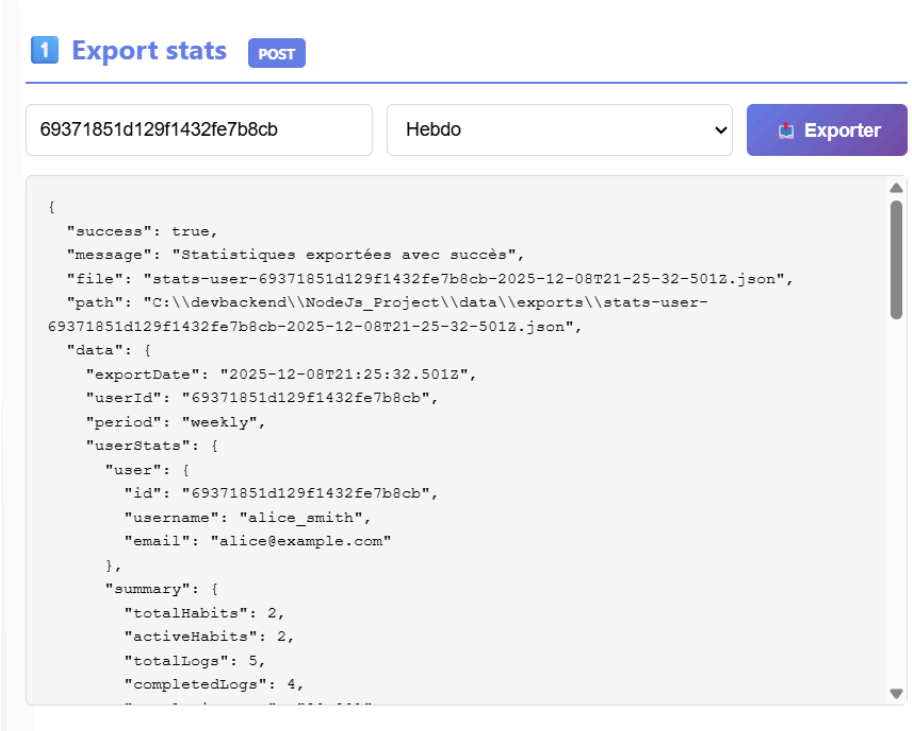


<u>Étudiant 4:</u> Statistics & Analytics

Objectif: Générer des statistiques globales exportables.

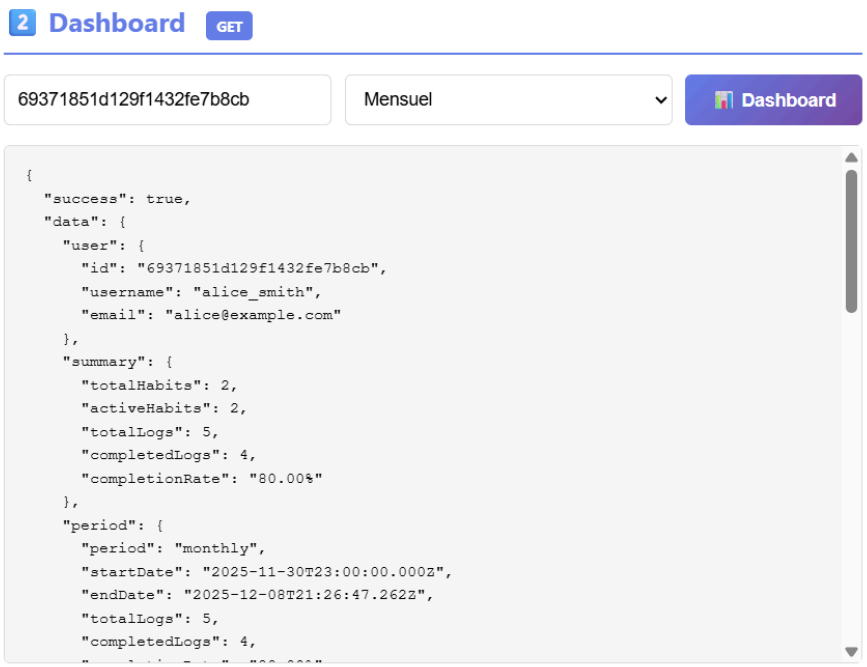
1️⃣POST /api/stats/export

Génère un fichier JSON



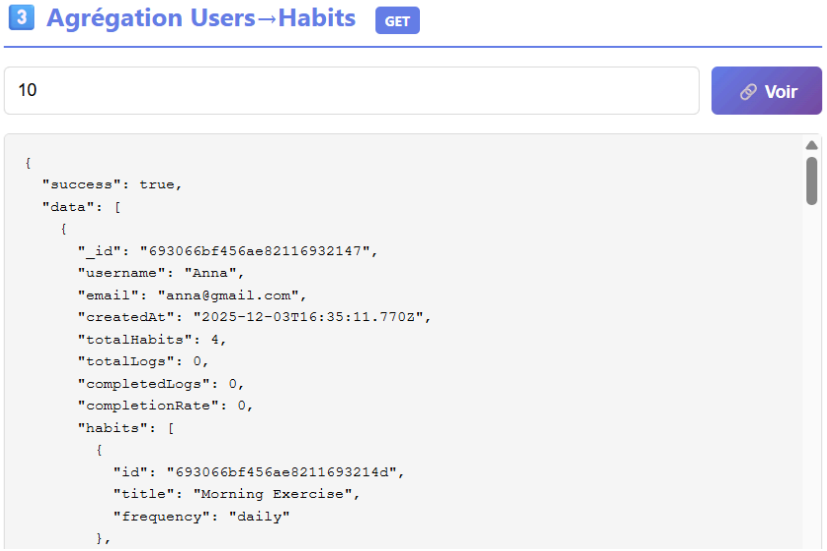
2️⃣GET /api/stats/dashboard

Statistiques par période (7j, 30j, 90j).

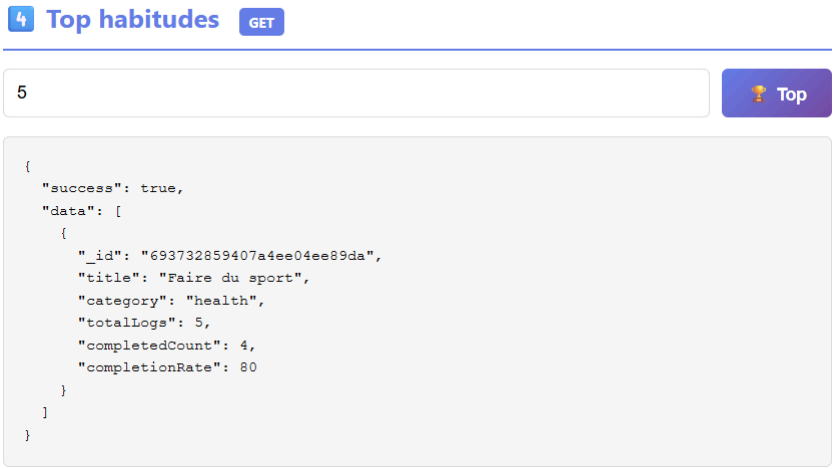


3 Agrégation avancée :

- **$lookup Users → Habits**



- **Top 5 habitudes**



- **Vue de l'ensemble**

5. Tests & Validation

HTML utilisé pour tester :

- Vérification des status codes
- Tests de validation Mongoose
- Tests de performance via agrégations

Interface Web locale (public/index.html) :

- Création d'utilisateurs
- Habitudes
- logs
- Statistiques

6. Résultats & Analyse:

- API 100% fonctionnelle pour CRUD complet.
- Agrégations avancées performantes.
- Structure modulaire facile à étendre.
- Cohérence respectée entre les étudiants.

Les fonctionnalités forment un backend mature, extensible, comparable à une vraie application .

7. Conclusion:

Ce projet **Habit Tracker** nous a permis de concevoir une API complète en Node.js, Express et MongoDB, structurée autour de modèles, contrôleurs et routes clairement définis.

Nous avons mis en place la gestion des utilisateurs, des habitudes, des logs de suivi ainsi que des statistiques avancées.

Ce travail nous a permis d'appliquer de vraies bonnes pratiques backend : validation, gestion d'erreurs, modularité, documentation et collaboration en équipe.

Au final, l'application constitue une base solide et évolutive pour un futur système complet de suivi d'habitudes.