Planit - Application Web de type Trello

Nom du projet (en cours) : Planit
Équipe : Fred TOSSOU et Mardochée ZOSSOUNGBO
Durée : 3 mois

1. Aperçu du projet

Ce document définit les **exigences fonctionnelles** et la **feuille de route** pour la création d'une application web professionnelle de type Trello dans un délai de 3 mois. Le projet vise à démontrer une architecture solide, des fonctionnalités principales, des pratiques de développement basées sur les tests et des pipelines de déploiement.

L'application fournira une hiérarchie principale : **Espaces de travail → Tableaux → Listes → Cartes**, en mettant l'accent sur une interface Material-UI propre et la fonctionnalité de glisser-déposer.

2. Objectifs

- Livrer un **MVP maintenable, stable et de haute qualité**.
- Mettre en avant un processus de développement professionnel : contrôle de version (Git), workflow basé sur les issues (Kanban), CI/CD, tests et documentation.
- Implémenter une base solide avec des points d'extension clairs pour les futures fonctionnalités.
- Assurer que l'application soit **performante, réactive et adaptée aux écrans desktop**.

3. Portée

MVP (Fonctionnalités indispensables)

- **Authentification et gestion des utilisateurs** : inscription, connexion/déconnexion (JWT), profil utilisateur (nom, avatar, email)
- **Espaces de travail** : CRUD, métadonnées (nom, description)

- **Tableaux** : CRUD, paramètres (titre, description, couleur/image de fond), archivage/suppression
- **Listes** : CRUD, réorganisation par glisser-déposer, archivage
- **Cartes** : CRUD, détails (titre, description), réorganisation intra et inter-listes par glisser-déposer, archivage
- **UX/UI** : composants Material-UI, snackbars pour feedback utilisateur, sidebar de navigation, dashboard listant les espaces de travail
- **Tests & CI** : tests unitaires pour les routes backend et logique frontend, pipeline GitHub Actions pour exécuter les tests sur PR
- **Déploiement** : frontend sur Vercel ou Netlify, backend sur Render/Railway ou similaire, MongoDB sur Atlas

Objectifs secondaires (si temps disponible)

- **Collaboration basique** : inviter des membres à un espace, rôles (Propriétaire, Membre)
- **Fonctionnalités avancées de cartes** : commentaires, labels, dates d'échéance, assignations, checklists, pièces jointes
- **Améliorations** : journal d'activité simple, mise à jour en temps réel via WebSockets (Socket.IO), notifications, mode sombre, recherche

4. Exigences non fonctionnelles

- **Performance** : l'application doit rester fluide pour 10-20 listes et 100-200 cartes par tableau; optimisation pour 1000+ cartes en objectif secondaire
- **Sécurité** : authentification JWT, CORS configuré, en-têtes de sécurité de base (helmet)
- **Scalabilité** : séparation claire des préoccupations (API stateless), modèle de données normalisé
- **Maintenabilité** : bonnes pratiques de code (ESLint, Prettier), README documenté, architecture modulaire
- **Disponibilité** : monitoring de base via plateformes de déploiement (Render, Vercel)

5. Stack technique

Frontend

- React.js (dernière version stable)
- Material UI (MUI)

- Redux Toolkit ou React Query
- react-router-dom
- dnd-kit ou Pragmatic Drag and Drop
- Axios ou Fetch

Backend

- Node.js + Express
- MongoDB (Atlas) avec Mongoose
- JWT pour authentification
- Jest + Supertest pour tests API

DevOps / Outils

- GitHub (repo, issues, Projects/Kanban)
- GitHub Actions pour CI
- Docker (développement et production)
- Vercel (frontend) / Render/Railway (backend)
- MongoDB Atlas
- Postman / Insomnia

6. Benchmark & Justification Technique

Frontend

Technologie choisie	Alternatives	Justification
React.js	Angular, Vue.js	Réactivité, large écosystème, bonne intégration avec Material UI. Angular plus lourd pour MVP, Vue moins répandu.
Material UI (MUI)	Bootstrap, Ant Design	Composants modernes et responsive, cohérence UI, documentation claire. Bootstrap plus basique, Ant Design moins flexible.
dnd-kit	React DnD	Léger et performant pour drag & drop intra/inter-list. React DnD plus complexe.

Backend

Technologie choisie	Alternatives	Justification
Node.js + Express	Django, Spring Boot	Léger et rapide pour APIs REST, JavaScript fullstack. Django/Spring Boot plus lourds.
MongoDB Atlas + Mongoose	PostgreSQL, MySQL	Flexible et facile à scaler, adapté aux structures dynamiques. SQL plus rigide pour ce type de hiérarchie.
JWT (jsonwebtoken)	OAuth, sessions côté serveur	Auth stateless, simple à intégrer et scalable. OAuth plus complexe pour MVP.

DevOps & CI/CD

Technologie choisie	Alternatives	Justification
GitHub Actions	Jenkins, GitLab CI	Intégration continue facile et native GitHub. Jenkins trop complexe pour MVP.
Docker	Vagrant, VM	Uniformise dev/prod, hot reload possible. Vagrant/VM plus lourds et moins flexibles.
Vercel / Render / Railway	Heroku, Netlify	Déploiement rapide, monitoring intégré, free tier. Heroku limité sur certaines options.

7. Architecture générale

- **Client (React)** : architecture par composants (Sidebar, WorkspacePage, BoardPage, ListComponent, CardModal), gestion local + globale/serveur
- **Serveur (Express)** : API RESTful, structure modulaire (auth, users, workspaces, boards, lists, cards), middlewares (JWT, gestion erreurs)
- **Base de données (MongoDB)** : collections normalisées

8. Modèle de données (simplifié)

Utilisateur (users)

```
{
  "_id": "ObjectId",
  "name": "String",
  "email": "String",
  "passwordHash": "String",
  "avatarUrl": "String",
  "createdAt": "Date"
}
```

Espace de travail (workspaces)

```
{
  "_id": "ObjectId",
  "name": "String",
  "description": "String",
  "owner": "ObjectId (ref: User)",
  "createdAt": "Date"
}
```

Tableau (boards)

```
{
  "_id": "ObjectId",
  "title": "String",
  "description": "String",
  "workspaceId": "ObjectId (ref: Workspace)",
  "settings": {"background": "String"},
```

```
  "archived": "Boolean",
  "createdAt": "Date"
}
```

Liste (lists)

```
{
  "_id": "ObjectId",
  "title": "String",
  "boardId": "ObjectId (ref: Board)",
  "position": "Number",
  "archived": "Boolean"
}
```

Carte (cards)

```
{
  "_id": "ObjectId",
  "title": "String",
  "description": "String",
  "listId": "ObjectId (ref: List)",
  "boardId": "ObjectId (ref: Board)",
  "position": "Number",
  "archived": "Boolean",
  "createdAt": "Date"
}
```

9. User Stories (MVP)

- Créer un compte et se connecter
- Créer un espace pour regrouper les tableaux
- Créer un tableau dans un espace
- Ajouter, renommer et archiver des listes dans un tableau
- Réorganiser les listes via glisser-déposer
- Ajouter des cartes (titre/description) à une liste
- Déplacer une carte d’une liste à une autre
- Réorganiser les cartes dans une même liste
- Archiver une carte

10. Jalons & Planning (12 semaines)

Sprint	Semaines	Focus
0	**0**	**Setup repo, CI, layout React**
1	**1-2**	**Auth, profil, dashboard**
2	**3-4**	**Workspaces + Boards CRUD**
3	**5-6**	**Lists + Cards CRUD, affichage Board**
4	**7-8**	**Glisser-déposer cartes**
5	**9-10**	**Glisser-déposer listes + archivage**
6	**11-12**	**Tests d'intégration, CI/CD, documentation, demo**

11. Stratégie de tests

- **Backend** : Jest + Supertest pour les routes API (auth, workspaces, boards, lists, cards)
- **Frontend** : Jest + React Testing Library pour composants, hooks et reducers
- **CI** : exécution des tests sur PR, passage obligatoire avant merge

12. CI/CD & Déploiement

- **CI** : GitHub Actions pour lint, tests et build
- **CD** : frontend sur Vercel, backend sur Render/Railway
- **Secrets** : clés API et URI DB dans variables d'environnement
- **Rollback** : redeployer le build précédent si nécessaire

13. Critères d'acceptation (MVP)

- Authentification fonctionnelle et profil de base géré

- Création d’espaces et de tableaux
- Création de listes et de cartes
- Glisser-déposer de cartes entre listes et réorganisation de listes
- Tests existants et exécutés en CI
- Application déployée et accessible via URL
- README complet avec instructions de setup, lancement et déploiement

14. Livrables

- Répertoire GitHub (monorepo ou deux repos) avec README documenté
- Déploiement fonctionnel (frontend + backend) avec DB MongoDB Atlas
- Suite de tests intégrée à CI
- Vidéo démonstration (3-5 min) des fonctionnalités MVP
- Documentation technique (architecture, modèle de données, API)

15. Améliorations futures (Post-MVP)

- Collaboration : invitations, rôles, commentaires
- Cartes : labels, dates d’échéance, assignations, checklists, pièces jointes
- Journal d’activité
- Mise à jour en temps réel (Socket.IO)
- Optimisation pour performances (1000+ cartes)
- Recherche, mode sombre, notifications