Guide de Structure du Projet et Bonnes Pratiques

Ce guide a pour objectif de présenter la structure de notre projet et de détailler le rôle de chacun des dossiers. Il servira de référence pour l'ensemble de l'équipe afin d'assurer une cohérence et une compréhension partagée de l'organisation du code.

1. Structure Générale du Répertoire

Le projet suit une architecture « monorepo » avec la répartition suivante :

```
bash
CopyEdit
/projet-root
  ├── .gitignore
  ├── .env
  ├── README.md
  ├── /docs                # Documentation (ce guide, normes,
spécifications, etc.)
  ├── /backend             # Code serveur (Express.js)
  │     ├── /controllers   # Logique métier et gestion des requêtes
  │     ├── /routes        # Définition des routes API (posts, users,
votes, etc.)
  │     ├── /models        # Modèles Sequelize et associations
  │     ├── /migrations    # Scripts de migration de la base de données
  │     ├── /seeders       # Données initiales pour la BDD
  │     ├── /middleware    # Gestion des erreurs, authentification,
etc.
  │     └── /tests         # Tests unitaires et d'intégration du
backend (Jest)
  ├── /frontend            # Code client (Vue.js)
  │     ├── /components    # Composants réutilisables
  │     ├── /views         # Pages et vues principales
  │     ├── /store         # Gestion d'état (Vuex ou autre)
  │     └── /tests         # Tests unitaires et d'intégration du
```

```
frontend
  └── /.github
        └── /workflows    # Workflows GitHub Actions (CI/CD et
vérifications de qualité)
```

Chaque dossier a une fonction précise pour garantir une séparation claire des responsabilités et faciliter la maintenance et la collaboration.

2. Détails par Dossier

A. Dossier backend

Ce dossier contient l'ensemble du code côté serveur qui utilise Express.js et Sequelize pour gérer l'API et la base de données.

- **/controllers :**
  - Contient la logique métier qui reçoit et traite les requêtes HTTP.
  - Chaque contrôleur est responsable d'un ensemble de fonctionnalités (ex. : gestion des articles, des utilisateurs, etc.).
- **/routes :**
  - Définit les endpoints de l'API.
  - Chaque fichier de route (ex. : `posts.js`, `users.js`) organise les différentes actions (GET, POST, PUT, DELETE) et relie ces actions aux contrôleurs.
- **/models :**
  - Regroupe les modèles Sequelize qui représentent les tables de la base de données.
  - Chaque modèle définit la structure des données et les relations entre elles (ex. : `User`, `Post`, `Vote`, `Comment`).
- **/migrations :**
  - Contient les scripts de migration pour mettre à jour le schéma de la base de données de façon dynamique.
  - Utilisez ces fichiers pour versionner et appliquer les modifications de la BDD sans la reconstruire manuellement.
- **/seeders :**
  - Fournit des scripts pour initialiser la base de données avec des données de base.

    - Idéal pour les environnements de développement ou de test afin d'avoir une BDD pré-remplie.
- **/middleware :**
    - Inclut des fonctions intermédiaires pour gérer des tâches transversales comme l'authentification, la gestion des erreurs ou la validation des données.
- **/tests :**
    - Contient les tests unitaires et d'intégration pour le backend.
    - Utilisez Jest pour vérifier que chaque module fonctionne comme attendu et que l'API répond correctement.

B. Dossier frontend

Ce dossier regroupe tout le code côté client, développé en Vue.js.

- **/components :**
    - Contient des composants réutilisables (ex. : boutons, formulaires, cartes d'articles).
    - Ces composants facilitent la maintenance et la cohérence de l'interface utilisateur.
- **/views :**
    - Regroupe les pages principales de l'application (ex. : page d'accueil, page de connexion, détails d'un article).
    - Ces vues combinent plusieurs composants pour former une interface complète.
- **/store :**
    - Utilisé pour la gestion d'état global (par exemple, via Vuex).
    - Assure la synchronisation des données entre les différents composants de l'application.
- **/tests :**
    - Contient les tests unitaires et d'intégration pour le frontend.
    - Permet de s'assurer que l'interface utilisateur se comporte comme prévu et que les composants interagissent correctement.

C. Dossier .github/workflows

Ce dossier contient tous les fichiers de configuration pour GitHub Actions, qui automatisent les tests et les vérifications de qualité.

- **Workflow Files :**

    - Chaque fichier YAML ici est dédié à une tâche précise (ex. : `backend-tests.yml`, `frontend-tests.yml`, `security-scans.yml`).
    - GitHub Actions détecte automatiquement ces fichiers et les exécute lors de la création de pull requests ou de push.
- **Bonne Pratique :**
    - Documentez chaque workflow avec des commentaires pour que l'équipe comprenne son rôle et sa configuration.

D. Dossier docs

Ce dossier est destiné à la documentation du projet.

- **Contenu Attendu :**
    - Guides d'utilisation et d'installation.
    - Spécifications techniques et schéma de la base de données.
    - Normes de codage et meilleures pratiques.
    - Ce guide de structure est également placé ici pour référence.
- **Mise à Jour :**
    - La documentation doit être régulièrement mise à jour afin de refléter les changements dans la structure ou les pratiques du projet.

E. Fichiers Globaux

- **.gitignore :**
    - Liste des fichiers et dossiers à ignorer par Git.
    - Placé à la racine, il garantit que les fichiers sensibles ou inutiles ne soient pas versionnés.
- **.env :**
    - Fichier de configuration d'environnement global (variables d'environnement).
    - Peut être complété par des fichiers spécifiques pour le backend ou le frontend si nécessaire.
- **README.md :**
    - Fournit une vue d'ensemble du projet, des instructions d'installation et d'utilisation.
    - Sert de première documentation pour toute personne découvrant le projet.

3. Bonnes Pratiques de Collaboration et de Développement

- **Respect de la Structure :**
  - Chaque développeur doit s'en tenir à la structure définie pour éviter les conflits et assurer une maintenance aisée.
  - Toute modification majeure de la structure doit être discutée en équipe.
- **Documentation Continue :**
  - Mettez à jour le dossier **docs** au fur et à mesure que de nouvelles pratiques ou structures sont adoptées.
  - Utilisez ce dossier comme source de vérité pour toutes les décisions techniques.
- **Utilisation des Branches :**
  - Suivez le modèle de branching défini dans la documentation globale du projet (branche main, develop et branches de fonctionnalité).
  - Assurez-vous que chaque fonctionnalité soit développée dans sa propre branche avant de fusionner dans develop.
- **Revues de Code et Pull Requests :**
  - Toutes les modifications doivent être soumises via une pull request et faire l'objet d'une revue de code.
  - Ceci permet de garantir la qualité et la cohérence du code.

4. Conclusion

Ce guide doit servir de référence pour tous les membres de l'équipe. En respectant cette structure et en suivant les bonnes pratiques décrites, nous assurons une collaboration efficace et une maintenance facilitée du projet.

N'hésitez pas à consulter ce document régulièrement et à le mettre à jour si de nouvelles questions ou besoins émergent. La clarté de notre organisation est la clé d'un développement réussi.