Documentation de Développement de l'Application de Blog

Ce document décrit en détail la structure du projet, la répartition des rôles, les processus de développement et les meilleures pratiques pour la création de notre application de blog. L'application permettra aux utilisateurs de créer des articles, de voter (upvote/downvote) et de commenter. Nous utiliserons Express.js pour le backend, Vue.js pour le frontend, et PostgreSQL avec Sequelize comme ORM pour la gestion de la base de données. Bien que l'intégration du déploiement ne soit pas une priorité pour le moment, l'accent sera mis sur la qualité du code, les tests automatisés et la sécurité.

1. Présentation du Projet

Technologies et Stack Utilisés :

- **Backend :** Express.js
- **Frontend :** Vue.js
- **Base de Données :** PostgreSQL avec Sequelize (ORM)
- **Tests :**
  - Jest pour les tests unitaires et d'intégration
  - Postman pour les tests d'API
- **Sécurité :**
  - Respect des directives OWASP
  - Protection contre les injections SQL et autres vulnérabilités
- **CI/CD (sans déploiement) :**
  - Utilisation de GitHub Actions pour automatiser les tests et assurer la qualité du code

Fonctionnalités Principales :

- **Articles de Blog :** Création, modification et suppression des articles.
- **Système de Votes :** Possibilité de voter positivement (upvote) ou négativement (downvote) sur les articles.

- **Commentaires :** Possibilité pour les utilisateurs de commenter les articles.
- **Gestion des Utilisateurs :** Inscription, authentification et gestion des profils utilisateurs.

2. Structure du Répertoire et des Fichiers

Organisation Globale du Répertoire

Nous adoptons une approche « monorepo » où l'ensemble du code est regroupé dans un seul dépôt. Cela permet une séparation claire entre le backend, le frontend et la documentation, tout en facilitant la gestion des fichiers de configuration globaux.

```
/projet-root
  ├── .gitignore
  ├── .env
  ├── README.md
  ├── /docs               # Documentation, spécifications d'API,
normes de codage, schéma de la BDD, etc.
  ├── /backend
  │     ├── /controllers  # Logique métier pour le traitement des
requêtes
  │     ├── /routes       # Définition des routes (posts.js, users.js,
votes.js, etc.)
  │     ├── /models       # Modèles Sequelize et associations
  │     ├── /migrations   # Fichiers de migration Sequelize pour la
mise à jour dynamique de la BDD
  │     ├── /seeders      # Fichiers seeders pour la population
initiale de la BDD
  │     ├── /middleware   # Middleware pour l'authentification, la
gestion des erreurs, etc.
  │     └── /tests        # Tests backend (tests unitaires et
d'intégration avec Jest)
  ├── /frontend
  │     ├── /components   # Composants Vue réutilisables (ex. :
PostComponent.vue, UserComponent.vue)
  │     ├── /views        # Composants de pages (ex. : Home.vue,
Login.vue, PostDetail.vue)
```

```
  │     ├── /store        # Configuration de Vuex (ou autre gestion
d'état)
  │     └── /tests        # Tests frontend (tests unitaires et
d’intégration)
  └── /.github
        └── /workflows    # Workflows GitHub Actions (fichiers YAML
pour les tests et les contrôles de qualité)
```

Fichiers Globaux vs. Fichiers Spécifiques aux Dossiers

- **Fichiers Globaux :**
  - `.gitignore` : Fichier placé à la racine pour indiquer les fichiers et dossiers à ignorer par Git.
  - `.env` : Fichier de configuration d’environnement global. Il est possible d’ajouter des fichiers d’environnement spécifiques (ex. : `.env.backend`, `.env.frontend`) si nécessaire, mais la configuration principale reste globale.
- **Fichiers Spécifiques aux Dossiers :**
  - Les configurations et scripts liés au backend (migrations, seeders, etc.) se trouvent dans `/backend`.
  - Les configurations et composants propres au frontend se trouvent dans `/frontend`.
  - Les workflows CI/CD sont stockés dans `/.github/workflows` et sont automatiquement détectés par GitHub Actions.

3. Gestion Dynamique de la Base de Données avec Sequelize

Pour permettre des mises à jour dynamiques de la base de données sans avoir à la reconstruire manuellement, nous utiliserons les fonctionnalités de migrations et seeders de Sequelize.

- **Modèles :**

Tous les modèles Sequelize seront placés dans le dossier `/backend/models`. Ces fichiers définissent la structure des données et leurs relations.

- **Migrations :**

Les fichiers de migration qui mettent à jour le schéma de la base de données se trouveront dans `/backend/migrations`. Cela permet une gestion versionnée des modifications apportées à la BDD.

- **Seeders :**

Les fichiers seeders pour initialiser la base de données avec des données de départ seront placés dans `/backend/seeders`.

Note : Veillez à ce que tous les développeurs sachent comment exécuter les migrations (par exemple via la CLI de Sequelize) afin de maintenir une base de données dynamique et à jour.

4. Configuration CI/CD (Sans Intégration du Déploiement)

Même si le déploiement n'est pas une priorité actuellement, il est essentiel d'automatiser les tests et de garantir la qualité du code.

Workflows GitHub Actions

- **Emplacement :**

Tous les fichiers de workflow doivent être placés dans le dossier `/.github/workflows`. GitHub Actions détecte automatiquement tous les fichiers YAML présents dans ce dossier.

- **Séparation des Workflows :**

Il est recommandé de diviser les workflows en fichiers distincts pour chaque responsabilité :

    - **backend-tests.yml :** Pour exécuter les tests backend avec Jest.
    - **frontend-tests.yml :** Pour exécuter les tests frontend.
    - **security-scans.yml :** Pour lancer les analyses de sécurité, incluant la vérification contre les vulnérabilités OWASP et les risques d'injection SQL.
- **Bonnes Pratiques de Configuration :**

- Chaque fichier YAML doit être dédié à une tâche spécifique (tests, linting, sécurité).
- Ajoutez des commentaires dans chaque fichier pour expliquer son rôle et faciliter la compréhension par tous les membres de l'équipe.

5. Répartition des Tâches et Attribution des Rôles

L'équipe compte 10 membres, répartis entre les tâches principales et les responsabilités complémentaires. Chaque domaine (backend et frontend) possède 2 membres principaux, les autres ayant des rôles additionnels (CI/CD, tests, sécurité) qui seront pris en charge une fois les fonctionnalités de base stabilisées.

Équipe Backend (5 Membres au Total) :

- **Développeurs Principaux Backend (2 Membres) :**
  - **Développeur A :**
    - **Tâche Principale :** Gestion de la route des articles (CRUD sur les posts).
  - **Développeur B :**
    - **Tâche Principale :** Gestion de la route des utilisateurs (inscription, authentification, gestion du profil).
- **Membres Additionnels Backend (3 Membres) :**
  - **Développeur C :**
    - **Tâche :** Implémentation du système de votes et réactions (upvote/downvote et logique métier associée).
  - **Développeur D :**
    - **Tâche :** Rôle mixte Backend/CI-CD – contribution initiale aux fonctionnalités core (middleware, utilitaires) puis transition vers la gestion des workflows CI/CD pour le backend.
  - **Développeur E :**
    - **Tâche :** Rôle mixte Backend/Tests & Sécurité – contribution initiale aux fonctionnalités core puis focalisation sur les tests et la sécurité (y compris la prévention des injections SQL).

Équipe Frontend (5 Membres au Total) :

- **Développeurs Principaux Frontend (2 Membres) :**
  - **Développeur F :**

        - **Tâche Principale :** Conception de la mise en page principale et des composants de la page d'accueil.
    - **Développeur G :**
        - **Tâche Principale :** Développement des composants liés aux utilisateurs (connexion, inscription, vues de profil).
- **Membres Additionnels Frontend (3 Membres) :**
    - **Développeur H :**
        - **Tâche :** Conception des composants d'affichage et de gestion des articles (création, édition, affichage détaillé).
    - **Développeur I :**
        - **Tâche :** Rôle mixte Frontend/CI-CD – contribution initiale aux composants UI puis transition vers la gestion des workflows CI/CD pour le frontend.
    - **Développeur J :**
        - **Tâche :** Rôle mixte Frontend/Tests & Sécurité – initialement sur le développement des interfaces, puis focalisation sur la rédaction de tests et l'assurance de la sécurité (validation et assainissement des entrées).

Transition des Tâches Additionnelles :

Une fois que les fonctionnalités de base sont stables, les membres affectés aux tâches additionnelles passeront progressivement à leurs rôles secondaires (CI/CD, tests et sécurité) tandis que les développeurs principaux continueront à développer les fonctionnalités de base.

6. Modèle de Branching et Permissions

Workflow de Branching Recommandé

Nous adopterons un modèle de branches qui sépare le code prêt pour la production du code en développement, ce qui permet de réduire les conflits et d'organiser la collaboration de manière structurée.

- **Branche Main :**
    - Contient le code de production (stable et prêt à être déployé).

    - **Permissions :** Branche protégée – seuls les responsables désignés (ex. : Chef, Adjoint) sont autorisés à fusionner des modifications dans cette branche.
- **Branche Develop :**
    - Branche d'intégration pour les développements en cours.
    - Toutes les branches de fonctionnalité se fusionneront d'abord dans cette branche avant d'être intégrées dans la branche main.
- **Branches de Fonctionnalité (Feature Branches) :**
    - Chaque nouvelle fonctionnalité, correction ou tâche doit démarrer sur sa propre branche (ex. : `feature/posts-route`, `feature/user-authentication`).
    - Ces branches sont créées à partir de la branche develop.

Exemple de Workflow :

1. Créer une nouvelle branche pour une fonctionnalité à partir de la branche develop.
2. Effectuer des commits réguliers et pousser les modifications.
3. Ouvrir une pull request contre la branche develop.
4. Soumettre le code à une revue par les responsables désignés avant la fusion.

Propriété des Branches et Contrôle des Permissions

- **Gestion des Permissions sur GitHub :**
    - Utilisez les règles de protection de branches de GitHub pour contrôler qui peut pousser ou fusionner des modifications dans les branches sensibles (main et develop).
    - Seuls certains membres (ex. : Chef, Adjoint et autres responsables assignés) auront le droit de fusionner dans ces branches.
- **Enforcement :**
    - Exiger des revues de code, des vérifications d'état et, si nécessaire, des validations (commit sign-offs) pour toute modification sur les branches protégées.
    - Les branches de fonctionnalités peuvent être collaboratives, mais leur fusion dans les branches protégées doit obligatoirement se faire via des pull requests contrôlées.

7. Directives de Sécurité

La sécurité est une priorité absolue. Les mesures suivantes et les bonnes pratiques doivent être rigoureusement respectées :

Bonnes Pratiques OWASP

- Exécuter régulièrement des analyses automatisées de sécurité.
- Suivre les directives OWASP pour le codage sécurisé, notamment en ce qui concerne la validation et la gestion des erreurs.

Prévention des Injections SQL

- **Règle Critique :**
  - Utilisez systématiquement des requêtes paramétrées ou les méthodes intégrées de Sequelize pour éviter toute injection SQL.
- **Bonnes Pratiques :**
  - Ne jamais concaténer des entrées utilisateurs brutes dans des requêtes SQL.
  - Utiliser les méthodes ORM de Sequelize (ex. : `findOne`, `findAll`, etc.) qui gèrent automatiquement l'échappement des données.
  - Valider et assainir toutes les entrées côté backend et frontend.

Mesures Générales de Sécurité

- **Validation et Assainissement des Entrées :**
  - Vérifiez que toutes les données fournies par les utilisateurs correspondent aux formats attendus.
  - Assainissez les données pour éliminer tout code malveillant ou caractère inattendu.
- **Revue de Code :**
  - Procédez régulièrement à des revues de code en mettant l'accent sur l'identification des vulnérabilités potentielles.
- **Documentation :**
  - Conservez les directives et bonnes pratiques de sécurité dans le dossier `/docs` pour que tous les membres puissent s'y référer.

8. Schéma de la Base de Données en Détail

La définition précise du schéma de la base de données est cruciale. Le schéma proposé ci-dessous doit être respecté et documenté dans le dossier `/docs` pour garantir la cohérence et la performance.

Table des Utilisateurs (Users)

- **id :** Clé primaire (UUID ou incrémentation)
- **username :** Chaîne de caractères, unique
- **email :** Chaîne de caractères, unique
- **password_hash :** Hachage sécurisé du mot de passe
- **role :** (Optionnel) Rôle ou niveau d'autorisation
- **created_at / updated_at :** Dates de création et de mise à jour

Table des Articles (Posts)

- **id :** Clé primaire
- **user_id :** Clé étrangère référant à `Users.id`
- **title :** Chaîne de caractères
- **content :** Texte
- **created_at / updated_at :** Dates de création et de mise à jour
- **status :** (Optionnel) Enum ou indicateur définissant si l'article est publié ou en brouillon

Table des Votes (Votes)

- **id :** Clé primaire
- **user_id :** Clé étrangère référant à `Users.id`
- **post_id :** Clé étrangère référant à `Posts.id`
- **vote :** Entier (ex. : +1 pour un upvote, -1 pour un downvote)
- **created_at :** Date et heure du vote
- **Contrainte Unique :** Association unique (user_id, post_id) pour éviter les votes en double

Table des Commentaires (Comments)

- **id :** Clé primaire
- **user_id :** Clé étrangère référant à `Users.id`

- **post_id :** Clé étrangère référant à `Posts.id`
- **content :** Texte
- **created_at / updated_at :** Dates de création et de mise à jour

Considérations Supplémentaires

- **Indexation :**
  - Créer des index sur les clés étrangères (ex. : `user_id` dans les tables Posts, Votes et Comments) pour optimiser les requêtes.
- **Associations dans Sequelize :**
  - Définir clairement les associations (ex. : `User.hasMany(Post)`, `Post.belongsTo(User)`, etc.) dans les modèles.
- **Migrations et Versionnage :**
  - Utiliser les migrations Sequelize pour versionner toutes les modifications du schéma.
  - Documenter chaque script de migration et procéder à des revues de code afin de garantir leur bon fonctionnement.

9. Défis Potentiels et Stratégies d'Atténuation

Conflits de Fusion et Problèmes d'Intégration

- **Défi :**
  - Les modifications chevauchantes dans les fichiers de configuration partagés ou dans les contrats d'API peuvent générer des conflits de fusion.
- **Stratégies d'Atténuation :**
  - Organiser des réunions d'intégration régulières pour synchroniser les équipes.
  - Utiliser des pull requests détaillées avec des revues de code systématiques.
  - Documenter clairement les interfaces et les contrats d'API dans le dossier `/docs`.

Gestion des Tâches et Transition vers les Rôles Additionnels

- **Défi :**

    - Le passage des membres affectés aux tâches additionnelles vers leurs rôles spécialisés peut temporairement créer des lacunes dans le développement des fonctionnalités de base.
- **Stratégies d'Atténuation :**
    - Prévoir des périodes de chevauchement pendant lesquelles les membres continuent de soutenir le développement principal.
    - Assurer des sessions de formation croisée et une documentation détaillée pour faciliter la transition.

Modèle de Branching et Gestion des Permissions

- **Défi :**
    - Une mauvaise communication ou des modifications non autorisées peuvent survenir si la gestion des branches n'est pas rigoureusement contrôlée.
- **Stratégies d'Atténuation :**
    - Appliquer strictement les règles de protection des branches via GitHub.
    - Exiger des revues de code et des validations pour toute fusion dans les branches protégées (main et develop).

Risques de Sécurité

- **Défi :**
    - Les vulnérabilités telles que l'injection SQL, une mauvaise gestion des entrées ou d'autres failles de sécurité.
- **Stratégies d'Atténuation :**
    - Sensibiliser l'équipe aux pratiques de codage sécurisé.
    - Intégrer des tests automatisés de sécurité dans les workflows CI/CD.
    - Mettre à jour régulièrement les mesures de sécurité conformément aux recommandations OWASP.

10. Conclusion

Cette documentation détaillée fournit une feuille de route complète pour le développement de notre application de blog. En respectant les lignes directrices ci-dessous, notre équipe pourra :

- Maintenir une structure de répertoire claire et organisée.

- Utiliser Sequelize pour une gestion dynamique et versionnée de la base de données.
- Mettre en place des workflows CI/CD modulaires assurant la qualité du code.
- Répartir les tâches de manière claire entre les développeurs principaux et les membres avec tâches additionnelles.
- Adopter un modèle de branching qui minimise les conflits et protège le code de production.
- Respecter les bonnes pratiques de sécurité, notamment contre les injections SQL, et assurer une validation rigoureuse des entrées.
- Suivre un schéma de base de données détaillé garantissant la cohérence et la performance.

Chaque membre de l'équipe est invité à se familiariser avec ce document et à s'y référer tout au long du développement. Grâce à une communication claire, des revues de code rigoureuses et le respect de ces directives, nous construirons ensemble une application de blog robuste, sécurisée et maintenable.

Merci de lire attentivement cette documentation. Pour toute question ou commentaire, n'hésitez pas à en discuter lors de notre prochaine réunion d'équipe.

Bon développement à tous !