



R408 Groupe7 Docker

SUJET : TWITTER / X

HUGO RAGUIN

EMILIEN ROUKINE

QUENTIN LAVAL

Table des matières

| | |
|---|----|
| Introduction..... | 1 |
| Choix Techniques..... | 2 |
| API : Modèles et Contrôleurs..... | 4 |
| Modèles..... | 4 |
| Contrôleurs..... | 4 |
| Exemple de Code..... | 5 |
| Base de Données..... | 6 |
| Explication des Tables..... | 6 |
| PMA..... | 7 |
| Modèle Conceptuel de Données (MCD)..... | 8 |
| Bonnes Pratiques..... | 9 |
| Commits Git..... | 9 |
| Organisation du Projet..... | 10 |
| Respect Environnemental..... | 11 |
| Fonctionnalités Réalisées et Non Réalisées..... | 12 |
| Fonctionnalités Réalisées :..... | 12 |
| Fonctionnalités Non Réalisées :..... | 12 |
| Conclusion..... | 13 |

Introduction

Dans le cadre de notre projet de virtualisation, notre équipe s'est lancée dans l'aventure ambitieuse de recréer une version simplifiée de **Twitter**, désormais appelée **X**.

Contrairement à l'approche qui aurait consisté à repenser entièrement l'interface et l'expérience utilisateur, nous avons choisi de rester fidèles au design original de Twitter.

Ce choix délibéré nous a permis de concentrer nos efforts sur la maîtrise des aspects techniques de la reconstruction d'une application existante, tout en naviguant dans les complexités du développement full-stack et de la virtualisation.

L'objectif était de construire une application fonctionnelle et interactive, capable de simuler les fonctionnalités clés de Twitter, avec un focus particulier sur une base de données robuste, une interface utilisateur réactive et une API performante pour gérer les interactions entre le client et le serveur.

Dans ce rapport, nous détaillerons les choix technologiques que nous avons faits, les raisons qui sous-tendent ces choix, le tout orchestré avec Docker Compose.

Choix Techniques

Base de Données: MariaDB

Pourquoi MariaDB? Choisi pour sa compatibilité avec MySQL, MariaDB est une base de données relationnelle open-source offrant d'excellentes performances pour les applications web, facilitant la gestion des données utilisateurs, des tweets, et des abonnements. Elle est également reconnue pour sa facilité d'utilisation avec Docker.

Alternatives Considérées: PostgreSQL pour sa robustesse et MongoDB pour sa flexibilité en tant que base de données NoSQL. MariaDB a été préférée pour sa simplicité et son efficacité dans notre contexte d'utilisation.

Front-end: Vue.js

Pourquoi Vue.js? Nous avons décidé de pivoter vers Vue.js en raison de sa facilité d'intégration, de sa réactivité, et de son système progressif qui le rend particulièrement adapté pour des projets de toutes tailles. Vue.js est célèbre sa capacité à faciliter le développement d'interfaces utilisateur dynamiques et interactives, parfait pour simuler l'expérience utilisateur de Twitter.

Alternatives Considérées: Nous avons envisagé d'utiliser la partie front end de Laravel car déjà utiliser pendant la SAE 3 et en raison de son intégration transparente avec le back-end et de sa capacité à fournir une solution full-stack en PHP. Flutter était une autre alternative considérée pour le développement front-end, en particulier pour ses performances élevées et son approche de développement multiplateforme. Cependant, en raison du manque de familiarité et d'expertise de notre équipe avec Flutter. Vue.js a été choisi pour sa simplicité et sa flexibilité, permettant une mise en place rapide et efficace du front-end.

Back-end: Laravel

Pourquoi Laravel? Laravel est un framework PHP pour le développement web qui permet de développer des applications web complexes de manière rapide et sécurisée. Il fournit un écosystème riche, intégrant une suite complète pour le développement back-end, de l'authentification à la gestion des bases de données, en passant par le routage et le cache. Pour notre projet, Laravel permet une structuration claire du code et facilite l'intégration avec MariaDB ainsi que la création d'une API RESTful robuste ainsi que le fait de notre expérience dans son utilisation.

Alternatives Considérées: Une alternative que nous avons envisagée était de développer le back-end en utilisant PHP natif en combinaison avec AJAX pour les appels asynchrones. Bien que cette approche soit possible, elle peut rapidement

devenir complexe à mesure que le projet grandit. De plus, Laravel offre des fonctionnalités supplémentaires telles que la gestion des migrations de base de données et la création d'API RESTful qui simplifient le développement et accélèrent le processus.

API : Modèles et Contrôleurs

MODÈLES

Les modèles représentent la structure de données de notre application. Ils définissent la manière dont les données sont stockées, récupérées et manipulées dans la base de données. Chaque modèle correspond à une table dans la base de données et a donc un nom explicite.

- **User.php**: Représente les utilisateurs de votre application. Il contient des propriétés comme l'ID de l'utilisateur, le nom, l'email, et les méthodes pour interagir avec les données utilisateur (ex : authentification, récupération des informations de l'utilisateur).
- **Tweet.php**: Gère les tweets postés par les utilisateurs. Ce modèle contient des champs tels que l'ID du tweet, le contenu du tweet, l'ID de l'utilisateur qui l'a posté, et des méthodes pour ajouter, supprimer, ou récupérer des tweets.
- **Comment.php**: Gère les commentaires faits sur les tweets. Il contient des attributs comme l'ID du commentaire, l'ID du tweet associé, le commentaire lui-même, et l'ID de l'utilisateur qui a commenté.
- **Followers.php**: S'occupe de la relation de suivi entre les utilisateurs, avec des champs tels que l'ID du suiveur, et l'ID de l'utilisateur suivi.
- **Like.php** et **Retweet.php**: Gèrent respectivement les "likes" et les "retweets" des tweets par les utilisateurs. Ces modèles contiennent des informations telles que l'ID de l'utilisateur, l'ID du tweet, et éventuellement le timestamp de l'action.

CONTRÔLEURS

Les contrôleurs prennent en charge la logique de l'application. Ils agissent comme intermédiaires entre les modèles et les vues, en traitant les requêtes entrantes, en manipulant les données via les modèles, et en renvoyant les réponses au client.

- **AuthController.php**: Gère l'inscription, la connexion et la déconnexion des utilisateurs. Il utilise le modèle **User** pour vérifier les identifiants et créer de nouveaux utilisateurs.

- **UserController.php**: Fournit des fonctionnalités liées à la gestion des utilisateurs, comme la mise à jour des profils et la récupération des données utilisateur. Interagit avec le modèle **User**.
- **TweetController.php**: Gère la gestion des tweets. Il permet aux utilisateurs de poster, supprimer, et récupérer des tweets en utilisant le modèle **Tweet**.
- **CommentController.php**: Permet aux utilisateurs d'ajouter, de supprimer et de récupérer des commentaires sur les tweets via le modèle **Comment**.
- **LikeController.php** et **RetweetController.php** (hypothétiques): Génèrent les "likes" et les "retweets" pour les tweets, utilisant les modèles **Like** et **Retweet**.

Ces contrôleurs et modèles ensemble permettent de structurer l'application en séparant clairement la logique de l'application de sa représentation et de sa persistance des données. Cette séparation des responsabilités facilite la maintenance, le test, et l'évolution de l'application.

EXEMPLE DE CODE

Voici un exemple de code tiré du **TweetController.php** pour illustrer l'interaction entre un contrôleur et son modèle :

```
// TweetController.php
public function postTweet(Request $request) {
    $tweet = new Tweet();
    $tweet->content = $request->content;
    $tweet->user_id = Auth::user()->id;
    $tweet->save();
    return response()->json(['message' => 'Tweet successfully posted!'], 200);
}
```

Base de Données

EXPLICATION DES TABLES

Tables Principales :

- **users**: Contient les informations des utilisateurs, telles que l'identifiant, le prénom, le nom, le pseudonyme, la biographie, l'email, le mot de passe, la date de naissance et la photo de profil. Chaque utilisateur est identifié par un ID unique.
- **tweets**: Stocke les tweets publiés par les utilisateurs. Chaque tweet est associé à un utilisateur via **user_id** et contient le texte du tweet ainsi que la date de création.
- **followers**: Enregistre les relations d'abonnement entre les utilisateurs, indiquant qui suit qui. Chaque relation est définie par un **follower_id** et un **followed_id**.
- **comment**: Conserve les commentaires que les utilisateurs font sur les tweets. Les commentaires sont liés à l'utilisateur qui les a faits (**user_id**) et au tweet commenté (**tweet_id**).
- **like**: Tient compte des "j'aime" que les utilisateurs donnent aux tweets. Chaque "like" est caractérisé par l'ID de l'utilisateur qui l'a donné et l'ID du tweet aimé.
- **retweet**: Similaire à la table **like**, cette table gère les retweets des utilisateurs, permettant de voir qui a retweeté quels tweets.
- **picture**: Contient des liens vers des images associées aux tweets. Chaque entrée fait référence à un **tweet_id**.

PMA

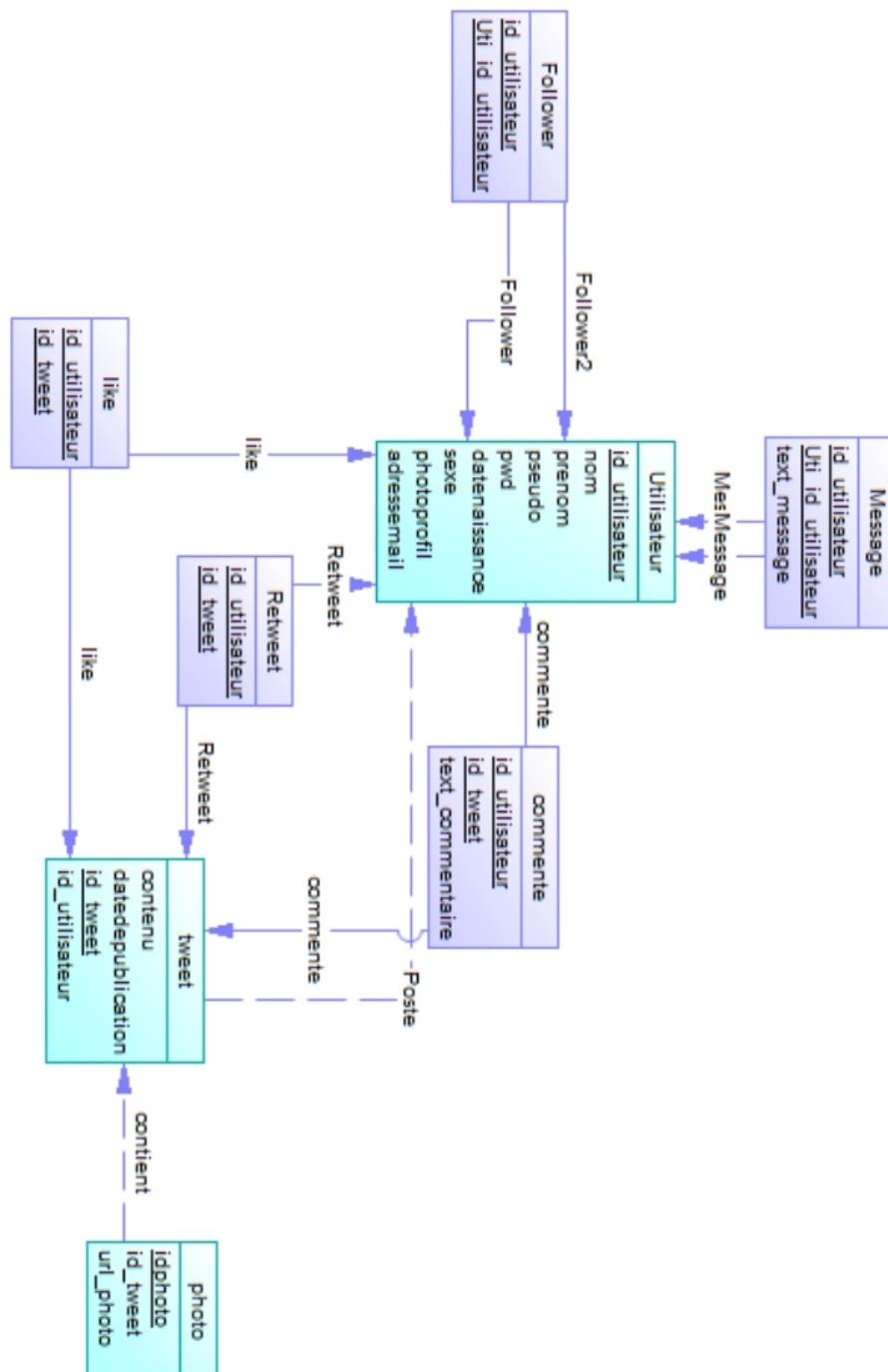
Accès Automatique: Après le démarrage du Docker, phpMyAdmin est accessible automatiquement via un navigateur web à l'adresse localhost:1313. Cela permet une connexion rapide à l'interface d'administration de la base de données.

Configuration des Identifiants: Lors de la connexion à phpMyAdmin, les identifiants de connexion sont automatiquement configurés en utilisant les variables d'environnement définies dans le fichier *docker-compose.yml*, telles que `${DB_USERNAME}`, `${DB_PASSWORD}` et `${DB_DATABASE}`, simplifiant ainsi le processus d'authentification.

Gestion des Bases de Données: Une fois connecté à phpMyAdmin, les utilisateurs ont accès à une interface conviviale pour gérer les bases de données, les tables et les requêtes SQL. Ils peuvent effectuer des opérations telles que la création, la modification et la suppression de bases de données, ainsi que l'exécution de requêtes pour interroger ou modifier les données.




Grâce à l'utilisation de phpMyAdmin dans notre configuration Docker, la gestion de la base de données MariaDB est simplifiée, offrant une interface accessible pour les administrateurs et les développeurs. Cela permet un développement plus efficace et une maintenance plus facile de notre application.

MODÈLE CONCEPTUEL DE DONNÉES (MCD)



Bonnes Pratiques

COMMITTS GIT

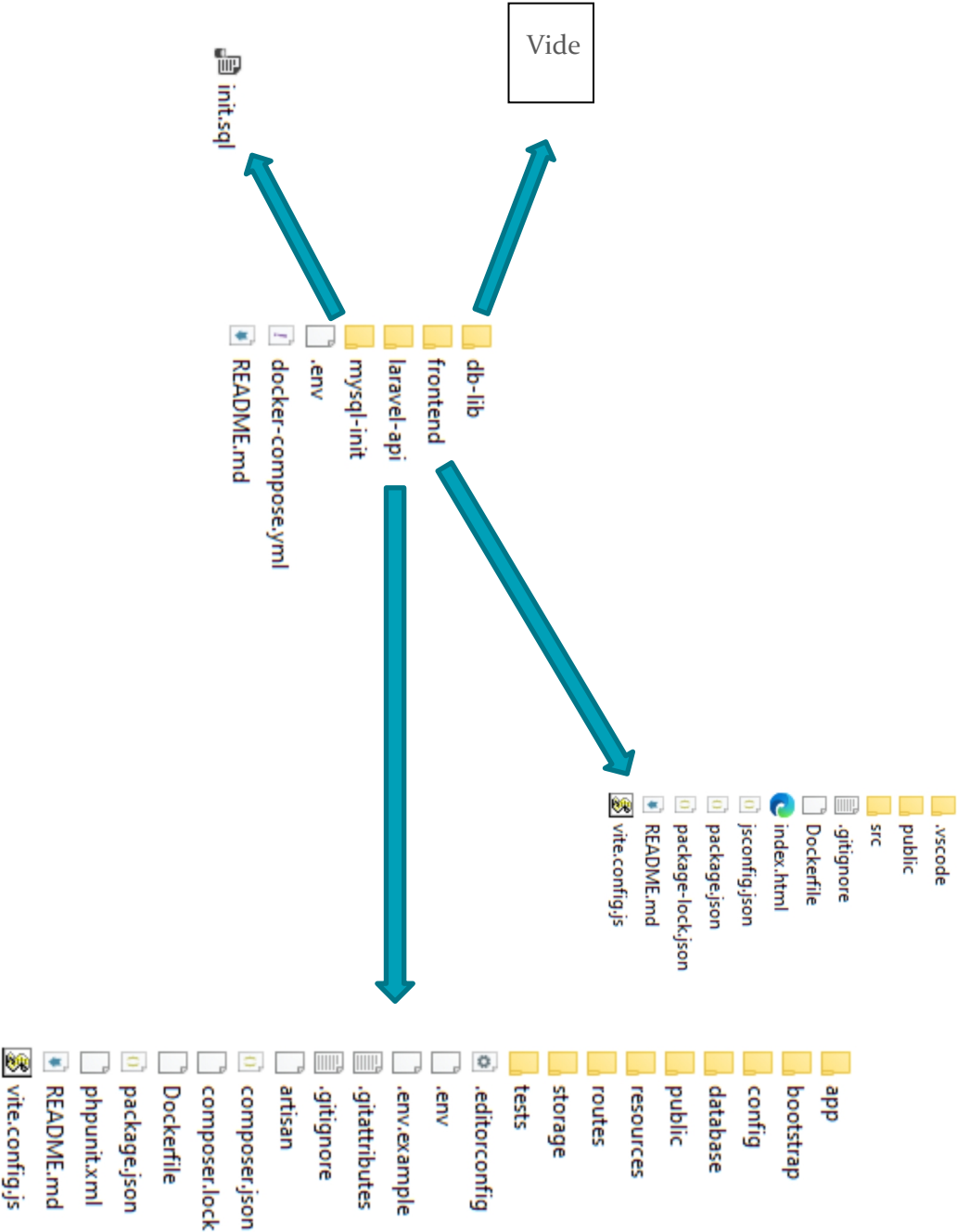
| |
|--|
| Update view account (affichage de s infos)  Styxql committed 14 minutes ago |
| Update README  Styxql committed 50 minutes ago |
| Ajout css + page compte  Styxql committed 1 hour ago |

"**Update view account (affichage des infos)**" : Ce commit indique spécifiquement que la vue du compte a été mise à jour pour afficher certaines informations. Cela montre que les modifications apportées concernent l'interface utilisateur et que ces changements sont probablement visuels.

"**Update README**" : Un commit dédié à la mise à jour du README montre que vous maintenez une documentation à jour, ce qui est essentiel pour la compréhension du projet par les nouveaux arrivants et les autres développeurs.

"**Ajout css + page compte**" : Un autre bon exemple montrant que le style CSS et la page de compte ont été travaillés. Il est spécifique sur les éléments ajoutés ou modifiés.

ORGANISATION DU PROJET



Respect Environnemental

Nous avons pris des mesures pour minimiser notre empreinte carbone et favoriser une utilisation efficace des ressources énergétiques :

Optimisation de l'infrastructure : En utilisant Docker pour l'orchestration des services, nous avons rationalisé le déploiement et la gestion des conteneurs, ce qui permet une utilisation plus efficace des ressources matérielles et énergétiques.

Choix de technologies éco-responsables : Nous avons sélectionné des technologies réputées pour leur efficacité énergétique, telles que Vue.js pour le front-end, qui offre une performance optimisée côté client et réduit la charge sur les serveurs.

En prenant des mesures pour réduire notre consommation d'énergie, nous contribuons à protéger l'environnement tout en offrant une bonne expérience utilisateur. Nous croyons que chaque action compte, et en tant que développeurs, nous pouvons encourager des pratiques plus respectueuses de l'environnement.

Fonctionnalités Réalisées et Non Réalisées

FONCTIONNALITÉS RÉALISÉES :

Publication de Tweets : Les utilisateurs peuvent publier des tweets et les partager avec leur réseau.

Likes de Tweets : Les utilisateurs peuvent exprimer leur appréciation en aimant les tweets publiés par d'autres utilisateurs.

Réponses aux Tweets : La fonctionnalité de répondre permet aux utilisateurs d'engager des conversations en répondant à des tweets spécifiques.

Retweet des Tweets : Les utilisateurs peuvent retweeter des tweets qu'ils trouvent intéressants pour les partager avec leurs abonnés.

Système de Connexion Sécurisé : L'application utilise un système de connexion sécurisé avec des mots de passe hashés pour protéger les comptes utilisateur.

Système de Follow : Les utilisateurs peuvent suivre d'autres utilisateurs pour voir leurs tweets dans leur flux d'actualités.

Onglet "Mon Compte" : Les utilisateurs ont accès à un onglet "Mon Compte" qui leur permet de retrouver toutes les informations pertinentes sur leur compte, telles que le nombre de followers, le nombre d'abonnements, leur pseudo, leur nom et prénom, etc.

Recherche : Les utilisateurs peuvent rechercher les tweets en inscrivant dans la barre de recherche au dessus de leur « feed ».

FONCTIONNALITÉS NON RÉALISÉES :

Messagerie Directe : La fonctionnalité de messagerie directe entre les utilisateurs n'a pas été implémentée.

Tendance et Exploration : La possibilité de découvrir des tweets tendances ou d'explorer des sujets spécifiques n'a pas été intégrée.

Conclusion

Le "Projet X Clone" est le résultat de notre travail d'équipe, mettant en avant notre compréhension solide du développement web et de la collaboration. Nous avons travaillé méthodiquement, en nous concentrant sur des commits précis et une organisation soignée du code, tout en fournissant une documentation claire et utile.

En nous appuyant sur des technologies telles que Vue.js pour le front-end et Docker pour l'orchestration des services, nous avons créé une application non seulement fonctionnelle mais aussi adaptée à une évolution et à une maintenance aisée.

En conclusion, nous avons posé les bases d'un projet dynamique et évolutif, qui témoigne de notre capacité à répondre à des exigences complexes et à travailler de manière harmonieuse.