

23/06/2025

Développement Web Avancé

Projet Breezy

FISA INFO A3



Groupe 5

CALVO – PINGET – JOASSON – MAILLARD-SALIN

Table des matières

1. Résumé :	2
2. Introduction :	3
3. Fonctionnalités :	3
4. Architecture micro-services :	4
5. Backend :	6
5.1 Docker :	6
5.2 Auth service :	8
5.3 Users service :	9
5.4 Posts service :	9
5.5 Feed service :	11
5.6 Traefik :	11
5.7 API Gateway :	12
5.8 Load Balancer :	13
6. Gestion des sessions :	13
7. Front-end :	15
8. Intégration du projet :	20
9. Gestion des autorisations :	24
10. Conclusion :	24

1. Résumé :

Le projet Breezy consiste à développer un réseau social léger et réactif, inspiré de Twitter/X, optimisé pour des environnements à faibles ressources. L'objectif est de permettre aux utilisateurs de publier des messages courts et d'interagir de manière fluide.

Le back-end utilise Node.js et Express pour gérer les requêtes via une API RESTful, avec une authentification sécurisée par JWT. Le front-end, développé en React, adopte une approche mobile-first pour une interface intuitive.

Les fonctionnalités principales incluent la création de comptes, la publication de messages, et les interactions sociales comme les likes et commentaires. Des fonctionnalités optionnelles, telles que les tags et notifications, sont également prévues.

L'architecture repose sur des microservices, avec Docker pour le déploiement. Breezy vise à offrir une solution innovante et accessible pour les utilisateurs dans des environnements à ressources limitées.

2. Introduction :

À l'ère du numérique, les réseaux sociaux sont essentiels pour la communication, mais beaucoup ne sont pas adaptés aux environnements à ressources limitées. C'est dans ce contexte que le projet Breezy émerge, visant à créer une plateforme légère et réactive inspirée de Twitter/X, optimisée pour des conditions de connectivité restreinte.

Ce rapport explore le développement de Breezy, détaillant les choix technologiques et les fonctionnalités mises en œuvre pour répondre aux besoins des utilisateurs. Nous aborderons les défis rencontrés, les solutions adoptées, et les perspectives d'amélioration.

En mettant l'accent sur la performance et l'accessibilité, Breezy se positionne comme une alternative viable aux réseaux sociaux traditionnels.

3. Fonctionnalités :

- **Fx1. Création de comptes utilisateurs avec validation**
 - En tant que visiteur, je veux créer un compte utilisateur afin de pouvoir accéder aux fonctionnalités de la plateforme.
- **Fx2. Authentification sécurisée**
 - En tant qu'utilisateur, je veux pouvoir me connecter de manière sécurisée pour protéger mes informations personnelles.
- **Fx3. Publication de messages courts**
 - En tant qu'utilisateur, je veux publier des messages courts (ex : 280 caractères) pour partager mes idées avec mes abonnés.
- **Fx4. Affichage des messages sur le profil**
 - En tant qu'utilisateur, je veux voir tous mes messages affichés sur mon profil afin de les consulter ou les modifier.
- **Fx5. Flux chronologique des messages des utilisateurs suivis**
 - En tant qu'utilisateur, je veux un fil d'actualités avec les messages des utilisateurs que je suis pour rester à jour avec leur contenu.
- **Fx6. Liker un post**
 - En tant qu'utilisateur, je veux liker un post pour montrer mon appréciation.
- **Fx7. Répondre à un post sous forme de commentaire**

- En tant qu'utilisateur, je veux répondre à un post pour partager mes réactions ou avis.
- **Fx8. Répondre à un commentaire sur un post**
 - En tant qu'utilisateur, je veux répondre à un commentaire sur un post pour participer à une discussion.
- **Fx9. Suivre ou être suivi par d'autres utilisateurs**
 - En tant qu'utilisateur, je veux pouvoir suivre d'autres utilisateurs pour voir leur contenu dans mon fil d'actualités.
- **Fx10. Profil utilisateur avec informations de base**
 - En tant qu'utilisateur, je veux une page de profil affichant mon nom, ma biographie courte et ma photo de profil pour me présenter aux autres.
- **Fx11. Liste des messages publiés par l'utilisateur sur le profil**
 - En tant qu'utilisateur, je veux voir la liste de mes messages publiés sur ma page de profil.

4. Architecture micro-services :

Voici une section sur la justification de l'architecture microservices pour votre projet Breezy :

Justification de l'Architecture Microservices

L'adoption d'une architecture microservices pour le projet Breezy est motivée par plusieurs facteurs clés qui répondent aux besoins spécifiques de notre application de réseau social léger et réactif. Voici les principales raisons qui justifient ce choix architectural :

1. Scalabilité :

- Les microservices permettent de faire évoluer indépendamment les différents composants de l'application. Cela signifie que nous pouvons allouer plus de ressources aux services qui en ont besoin, comme ceux gérant les interactions utilisateur ou le traitement des messages, sans avoir à mettre à l'échelle l'ensemble de l'application.

2. Flexibilité Technologique :

- Chaque microservice peut être développé et déployé en utilisant les technologies les plus adaptées à sa fonction spécifique. Par exemple, un service peut utiliser Node.js pour sa légèreté et sa rapidité dans la gestion des requêtes I/O, tandis qu'un autre peut utiliser un langage différent mieux adapté à des tâches de traitement de données intensives.

3. Déploiement et Mise à Jour Simplifiés :

- Avec les microservices, les mises à jour et les corrections de bugs peuvent être déployées de manière indépendante pour chaque service. Cela réduit les risques associés aux déploiements et permet des mises à jour plus fréquentes et ciblées sans interruption majeure du service.

4. Résilience et Fiabilité :

- Une architecture microservices améliore la résilience de l'application. En cas de défaillance d'un service, les autres peuvent continuer à fonctionner normalement, limitant ainsi l'impact sur l'expérience utilisateur globale.

5. Optimisation des Ressources :

- Les microservices permettent une meilleure optimisation des ressources, car chaque service peut être dimensionné et configuré en fonction de ses besoins spécifiques. Cela est particulièrement important pour une application comme Breezy, conçue pour fonctionner dans des environnements à ressources limitées.

6. Facilité de Développement et de Maintenance :

- En divisant l'application en petits services autonomes, les équipes de développement peuvent travailler plus efficacement et en parallèle sur différentes parties de l'application. Cela facilite également la maintenance, car les problèmes peuvent être isolés et traités au niveau du service concerné.

7. Adaptabilité aux Changements :

- Les microservices permettent une plus grande adaptabilité aux changements des besoins métiers ou technologiques. De nouveaux services peuvent être ajoutés, et les services existants peuvent être modifiés ou remplacés sans affecter l'ensemble du système.

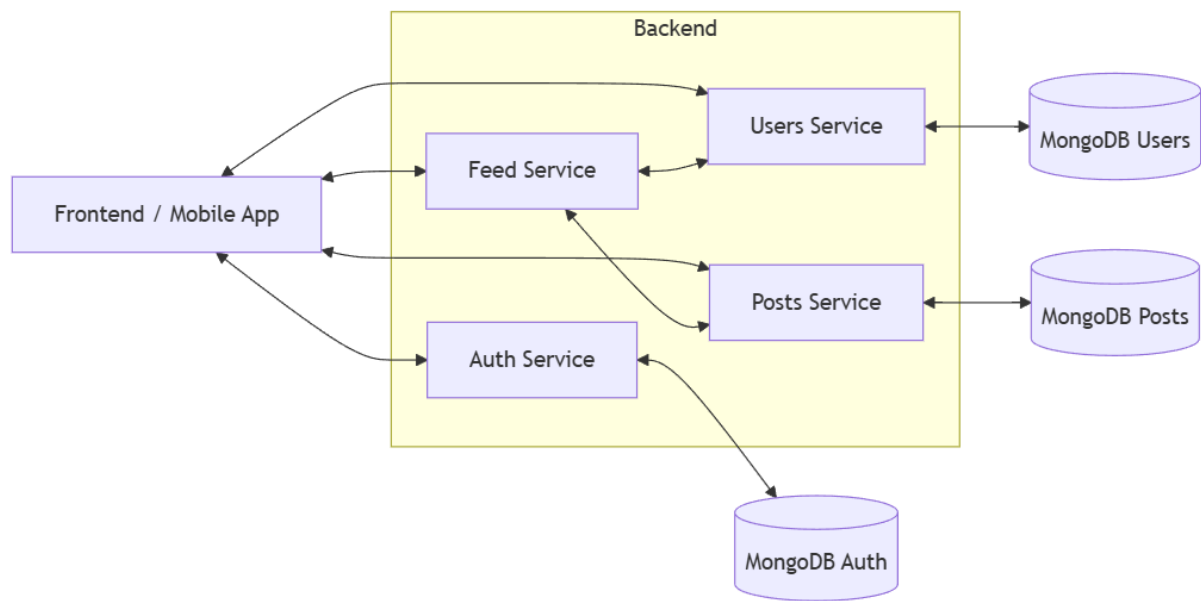


Figure 1 : Architecture

5. Backend :

Le backend de l'application Breezy est conçu selon une architecture microservices, permettant une gestion efficace et modulaire des différentes fonctionnalités de l'application. Cette approche facilite la scalabilité, la maintenance et le déploiement indépendant des services. Voici une description des principaux services qui composent le backend de Breezy :

Technologies utilisées :

- Node.js
- JWT
- Express
- MongoDB
- Mongoose

5.1 Docker :

Dans le cadre de notre projet, Docker a joué un rôle central en facilitant le déploiement, la gestion et l'isolation des différents services et bases de données. Voici comment Docker a été utilisé et les avantages qu'il a apportés à notre architecture.

Conteneurisation des Services

1. Isolation des Environnements :

- Chaque service de notre application, tel que le service d'authentification, le service utilisateur et le service de publication, est exécuté dans un conteneur Docker séparé. Cela permet d'isoler les environnements d'exécution, garantissant que les dépendances et configurations d'un service n'interfèrent pas avec celles des autres.

2. Déploiement Simplifié :

- Docker permet de créer des images contenant tout ce qui est nécessaire pour exécuter un service, y compris le code, les bibliothèques, les variables d'environnement et les outils système. Ces images peuvent être facilement déployées sur n'importe quel environnement prenant en charge Docker, ce qui simplifie grandement le processus de déploiement.

3. Gestion des Dépendances :

- Grâce à Docker, nous pouvons définir précisément les dépendances et les versions des logiciels nécessaires pour chaque service dans un fichier Dockerfile. Cela assure la cohérence entre les environnements de développement, de test et de production.

Orchestration avec Docker Compose

1. Définition des Services et Réseaux :

- Nous avons utilisé Docker Compose pour définir et orchestrer les différents services de notre application. Le fichier docker-compose.yml décrit la configuration de chaque service, y compris les images à utiliser, les ports à exposer, les volumes à monter et les réseaux à connecter.

2. Réseaux Personnalisés :

- Un réseau personnalisé, nommé breezy-network, a été créé pour permettre la communication entre les différents conteneurs. Cela facilite l'interconnexion des services et des bases de données tout en maintenant l'isolation par rapport à d'autres applications exécutées sur le même hôte.

3. Gestion des Volumes :

- Docker Compose permet également de définir des volumes pour la persistance des données. Par exemple, les bases de données MongoDB utilisent des volumes pour stocker les données de manière persistante, assurant ainsi que les données ne sont pas perdues lorsque les conteneurs sont redémarrés ou recréés.

5.2 Auth service :

- **Rôle** : Ce service est chargé de l'authentification et de l'autorisation des utilisateurs. Il gère la connexion, la déconnexion, et la validation des sessions utilisateur à l'aide de JSON Web Tokens (JWT).
- **Base de Données** : Les informations d'authentification sont stockées dans une base de données MongoDB dédiée.
- **Interactions** : L'Auth Service travaille en étroite collaboration avec le Users Service pour valider les informations d'identification et gérer les sessions utilisateur.

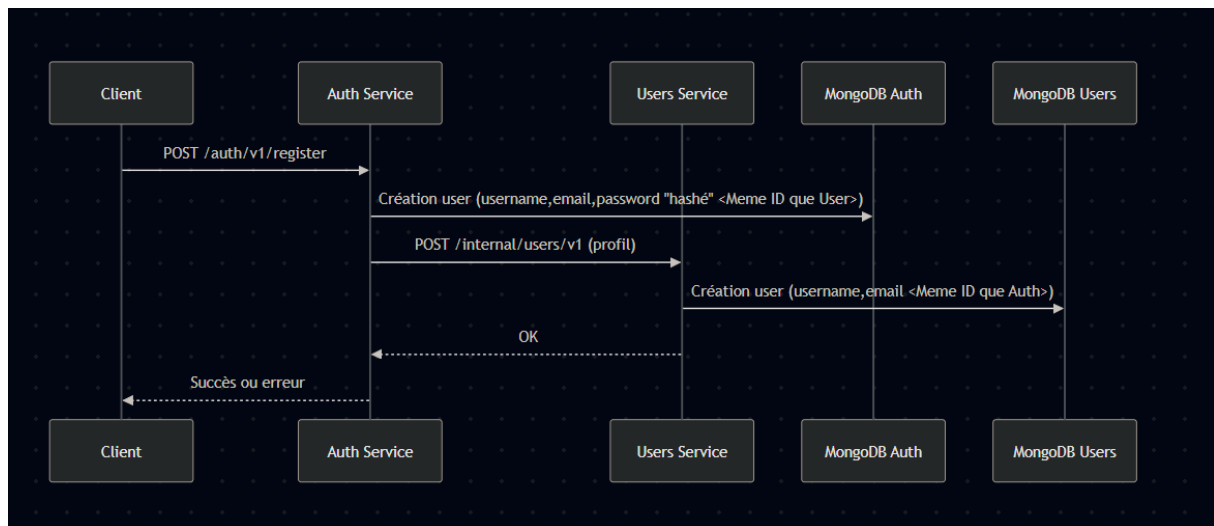


Figure 2 : Diagramme de séquence d'authentification

La donnée entrée par l'utilisateur dans le service d'authentification est dupliquée dans la base de données Users en plus de la base de données Auth. Cela permet de séparer le stockage du mot de passe du reste des données de l'utilisateur.

Structure de la donnée enregistrée :

- Username
- Email
- Password
- createdAt (Date de création de l'utilisateur)

Le service d'authentification permet l'utilisation d'un refresh token pour obtenir un nouvel access token après expiration sans demander à l'utilisateur de se réauthentifier.

Structure du refresh token :

- userId

- token
- expiresAt (Expiration du jeton)
- revoked (booléen d'état du jeton)

5.3 Users service :

- **Rôle** : Ce service gère toutes les opérations liées aux utilisateurs, y compris la création de comptes, la gestion des profils utilisateurs, et la récupération des informations utilisateur.
- **Base de Données** : Les données des utilisateurs sont stockées dans une base de données MongoDB dédiée.
- **Interactions** : Le Users Service communique avec le Feed Service pour fournir les informations nécessaires sur les utilisateurs suivis et avec l'Auth Service pour la gestion des sessions utilisateur.

Structure d'un utilisateur :

- `_id` : Identifiant unique attribué à un utilisateur
- Username
- Email
- Followed : Liste qui contient les utilisateurs suivis

5.4 Posts service :

- **Rôle** : Ce service s'occupe de la gestion des messages publiés par les utilisateurs. Cela inclut la création, la modification, la suppression et la récupération des messages.
- **Base de Données** : Les messages sont stockés dans une base de données MongoDB dédiée, appelée MongoDB Posts, permettant une gestion efficace des données de messages.
- **Interactions** : Le Posts Service interagit avec le Feed Service pour fournir les messages à afficher dans le flux des utilisateurs.

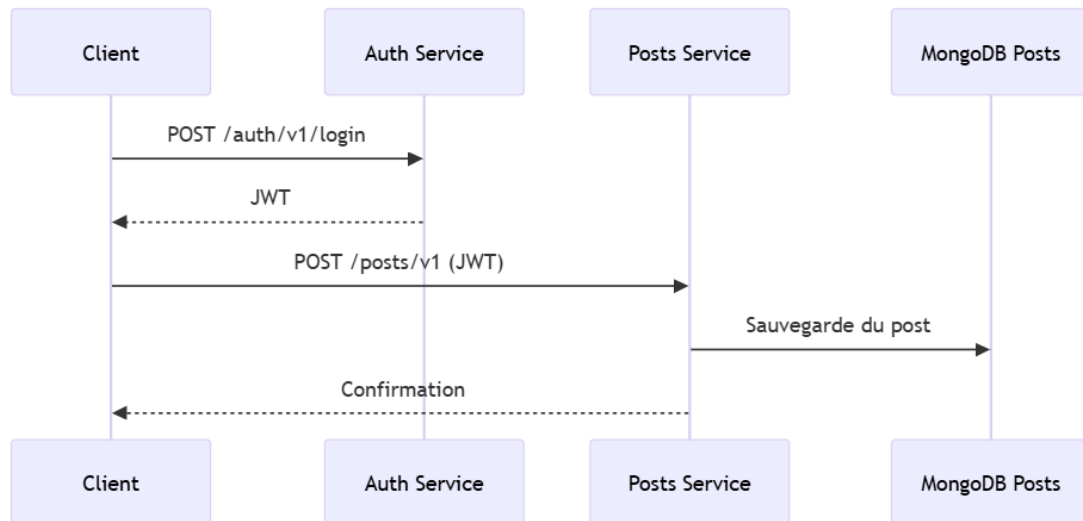


Figure 3 : Diagramme de séquence des posts

Structure d'un post :

- userId : Lie le post à l'utilisateur
- username : Même fonction que userId
- title : Titre du post
- content : Contenu du post
- likes : liste des utilisateurs ayant likés le post
- comments : liste de Schéma des commentaires

Structure d'un commentaire :

- _id : Identifiant du commentaire
- userId : Lie le commentaire à l'utilisateur
- username : Même fonction que userId
- content : Contenu de la réponse
- createdAt : Date de création
- replies : liste de Schéma de réponses

Structure d'une réponse :

- userId : Lie la réponse à l'utilisateur
- username : Même fonction que userId
- content : Contenu de la réponse
- createdAt : Date de création
- _id : Identifiant de la réponse

5.5 Feed service :

- **Rôle** : Ce service est responsable de la gestion du flux de messages affichés aux utilisateurs. Il récupère les messages des utilisateurs suivis et les organise de manière chronologique pour être affichés sur le front-end.
- **Interactions** : Le Feed Service interagit principalement avec le Posts Service pour récupérer les messages et avec le Users Service pour obtenir les informations sur les utilisateurs suivis.

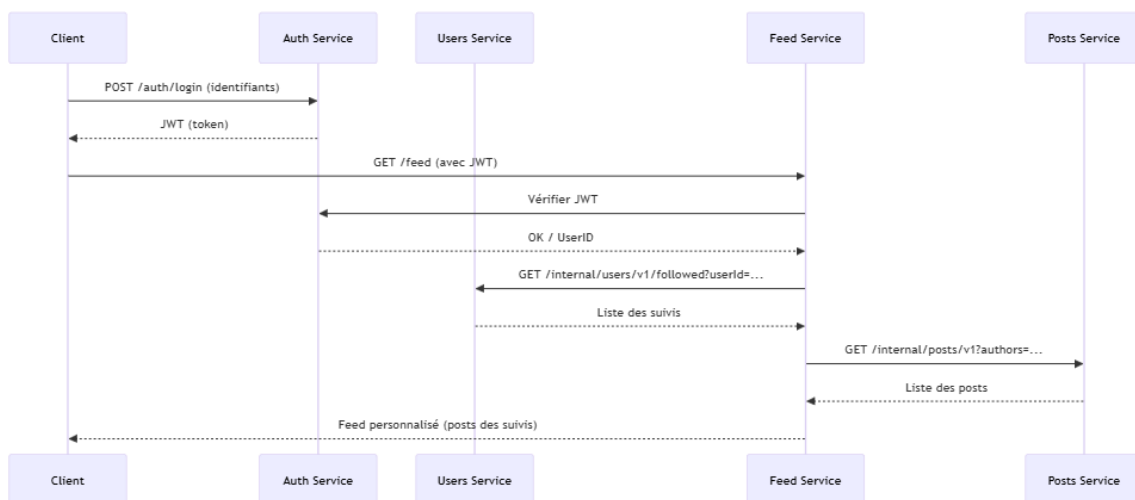


Figure 4 : Diagramme de flux du feed

5.6 Traefik :

Traefik est un reverse proxy et un load balancer moderne et dynamique, conçu pour faciliter le déploiement de microservices. Il est particulièrement apprécié pour sa capacité à s'intégrer naturellement avec les environnements conteneurisés, comme Docker et Kubernetes. Voici quelques points clés sur l'utilisation de Traefik :

Configuration et Déploiement

1. Intégration avec Docker :

- Traefik peut être configuré pour découvrir automatiquement les services Docker et les exposer en tant que routes. Cela se fait en utilisant les labels Docker pour spécifier les règles de routage.

2. Configuration de Base :

- Traefik utilise un fichier de configuration pour définir les entrées (entrypoints), les fournisseurs (providers), et les middlewares.

Fonctionnalités Avancées

1. Load Balancing :

- Traefik peut répartir la charge entre plusieurs instances d'un service pour améliorer la performance et la disponibilité.

2. Middleware :

- Les middlewares permettent d'ajouter des fonctionnalités comme l'authentification, la compression, et la gestion des en-têtes.

5.7 API Gateway :

Dans notre architecture, Traefik joue le rôle d'une API Gateway, un point d'entrée unique pour toutes les requêtes client. Cette configuration centralise la gestion des requêtes et simplifie le processus de routage vers les différents services de notre application.

Fonctionnalités de l'API Gateway

1. Routage des Requêtes :

- Traefik est configuré pour diriger les requêtes vers les services appropriés en fonction du chemin d'URL. Par exemple, les requêtes avec le préfixe /auth/v1 sont automatiquement acheminées vers le service d'authentification. Cela est réalisé grâce à des labels Docker qui définissent des règles de routage spécifiques pour chaque service.

2. Découverte Automatique des Services :

- Grâce à son intégration avec Docker, Traefik découvre dynamiquement les services disponibles et configure les routes en conséquence. Cela permet une mise à jour automatique des routes lorsque de nouveaux services sont ajoutés ou que des services existants sont modifiés.

3. Tableau de Bord :

- Traefik offre un tableau de bord accessible via traefik.localhost sur le port 8080. Ce tableau de bord fournit une interface utilisateur pour surveiller et gérer les routes, les services et les middlewares configurés.

4. Sécurité et Middlewares :

- Des middlewares sont utilisés pour ajouter des fonctionnalités supplémentaires, telles que l'authentification et la gestion des en-têtes.

Par exemple, un middleware d'authentification est appliqué pour vérifier les requêtes avant qu'elles n'atteignent les services protégés.

5.8 Load Balancer :

Traefik agit également comme un Load Balancer, répartissant les requêtes entrantes entre plusieurs instances d'un même service pour améliorer la performance et la disponibilité de notre application.

Fonctionnalités du Load Balancer

1. Répartition de Charge :

- Traefik répartit les requêtes entre les différentes instances d'un service. Cela permet de gérer efficacement le trafic et d'éviter la surcharge d'une seule instance. Par exemple, le service d'authentification est déployé avec deux réplicas, et Traefik répartit les requêtes entre ces réplicas.

2. Sticky Sessions :

- Pour maintenir l'affinité de session, Traefik utilise des cookies pour s'assurer que les requêtes d'un même client sont toujours dirigées vers la même instance de service. Cela est particulièrement utile pour les applications qui nécessitent une session utilisateur cohérente.

3. Haute Disponibilité :

- En répartissant la charge entre plusieurs instances, Traefik améliore la disponibilité de l'application. Si une instance tombe en panne, les requêtes sont automatiquement redirigées vers les instances restantes, assurant ainsi une continuité de service.

6. Gestion des sessions :

Dans notre architecture, la gestion des sessions et l'authentification des utilisateurs sont assurées par l'utilisation de JSON Web Tokens (JWT). Cette approche permet de sécuriser les échanges entre les clients et les services tout en maintenant une expérience utilisateur fluide et sécurisée.

Mise en œuvre de JWT dans notre Projet

1. Authentification et Génération de Token :

- Lorsque l'utilisateur se connecte avec succès, le service d'authentification génère un JWT contenant des informations sur l'utilisateur, telles que l'identifiant utilisateur et le nom d'utilisateur. Ce token est signé avec une clé secrète connue uniquement du service d'authentification.
- Le JWT est ensuite renvoyé au client, qui le stocke dans un cookie sécurisé.

2. Utilisation du Token pour les Requêtes Ultérieures :

- Pour chaque requête ultérieure à un service protégé, le client inclut le JWT dans l'en-tête Authorization sous la forme Bearer <token>.
- Traefik, en tant qu'API Gateway, intercepte ces requêtes et utilise un middleware pour vérifier la validité du token. Le middleware redirige la requête vers le service d'authentification pour validation.

3. Validation du Token :

- Le service d'authentification vérifie la signature du JWT à l'aide de la clé secrète. Si le token est valide, le service d'authentification renvoie les informations utilisateur nécessaires pour traiter la requête.
- Si le token est invalide ou expiré, la requête est rejetée avec une réponse d'erreur appropriée, telle qu'une erreur 401 Non autorisé.

4. Gestion des Sessions :

- Grâce aux JWT, nous pouvons gérer les sessions utilisateur de manière sécurisée et sans état (stateless). Cela signifie que le serveur n'a pas besoin de stocker des informations de session, ce qui simplifie la mise à l'échelle de l'application.
- Les tokens JWT ont une durée de vie limitée, ce qui réduit le risque de compromission à long terme. Lorsqu'un token expire, l'utilisateur doit se réauthentifier pour obtenir un nouveau token.

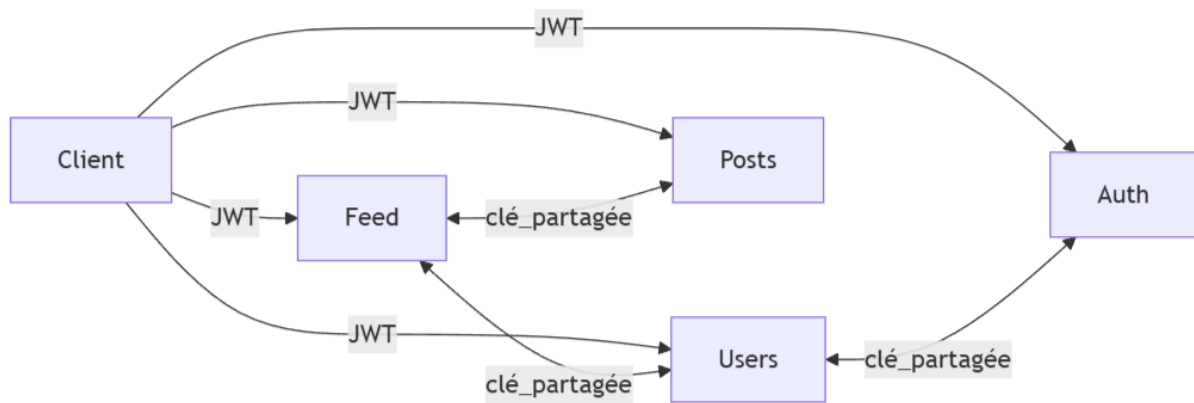


Figure 5 : Utilisation du JWT par les services

7. Front-end :

Dans notre projet, le front-end a été développé en utilisant une combinaison de technologies modernes pour offrir une expérience utilisateur fluide, réactive et sécurisée. Voici un aperçu des technologies utilisées et des approches adoptées pour la conception et le développement du front-end.

Technologies utilisées :

- React.js
- Next.js
- Tailwind CSS
- Axios
- React Router

7.1 Réactivité et UX/UI

1. Design Responsive :

- Grâce à Tailwind CSS, nous avons pu créer un design entièrement responsive qui s'adapte à différentes tailles d'écran. Les utilitaires de Tailwind nous ont permis de définir des styles spécifiques pour différentes largeurs d'écran, assurant ainsi une expérience cohérente sur tous les appareils.

2. Gestion des Erreurs UI :

- Nous avons mis en place une gestion robuste des erreurs pour informer les utilisateurs de manière claire et utile lorsqu'une erreur se produit. Cela inclut l'affichage de messages d'erreur conviviaux et la fourniture de suggestions pour résoudre les problèmes rencontrés.

7.2 Gestion des Sessions

1. Stockage des JWT :

- Pour gérer les sessions utilisateur, nous stockons les tokens JWT (JSON Web Tokens) dans un cookie côté client après une authentification réussie. Cela permet de maintenir l'état de connexion de l'utilisateur et d'effectuer des requêtes authentifiées vers notre back-end.

2. Redirection après Authentification :

- Après une authentification réussie, l'utilisateur est redirigé vers une page protégée de l'application. Cela est géré par React Router, qui nous permet de naviguer de manière programmatique vers différentes routes en fonction de l'état d'authentification de l'utilisateur.

3. Protection des Routes :

- Nous avons mis en place des mécanismes pour protéger les routes qui nécessitent une authentification. Si un utilisateur non authentifié tente d'accéder à une route protégée, il est redirigé vers la page de connexion. Cela est réalisé en vérifiant la présence et la validité du JWT stocké.

4. Refresh Token :

- L'access token est stocké dans redux et le refresh token est stocké dans un cookie qui appelle la route refresh quand le serveur retourne un message d'expiration du cookie

7.3 Interface utilisateur :

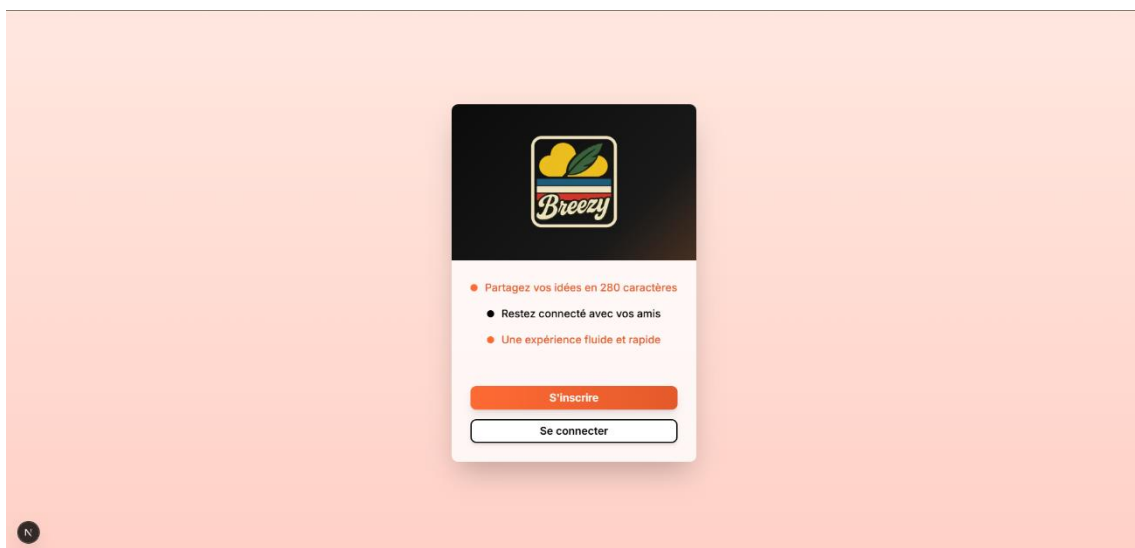


Figure 6 : Page d'accueil

Créer un compte

Rejoignez Breezy pour partager vos idées avec le monde

Nom d'utilisateur

Email

Mot de passe

Confirmer le mot de passe

S'inscrire

Vous avez déjà un compte? [Se connecter](#)

Figure 7 : Création du compte

Se connecter

Connectez-vous pour accéder à votre compte Breezy

Email

alban.calvo1@viacesi.fr


Mot de passe

••••

Se connecter

[Vous n'avez pas de compte? S'inscrire](#)

Figure 8 : Login

 Breezy

Accueil

Recherche

Notifications

Profil

Paramètres

jeudi 26 juin

22:58

23°C

Publier

@admin_principal

Utilisateur

Fil d'actualités

Titre

Quoi de neuf ?

0/280 caractères

Publier

Pour vous

Abonnements

A

admin_principal

@admin_principal

· il y a environ 7 heures

test

dfsdfsdfst

1

U

julie_art

@julie_art

· il y a environ 7 heures

MongoDB vs PostgreSQL

Analyse comparative pour choisir la bonne base de données

1

U

julie_art

@julie_art

· il y a environ 7 heures

Figure 9 : Fil d'actualités

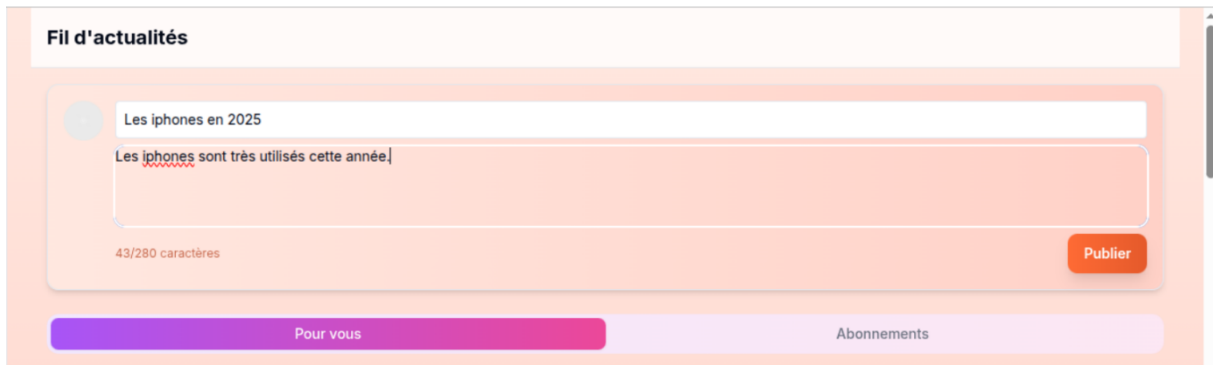


Figure 10 : Post

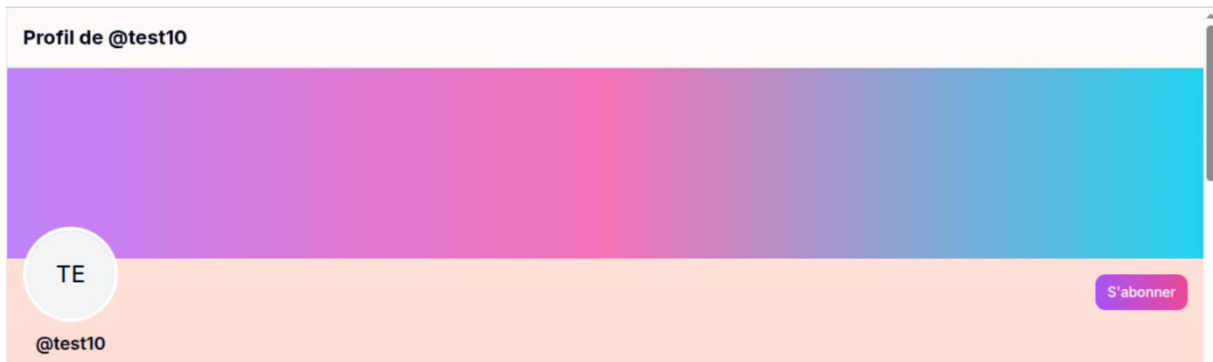


Figure 11 : Suivre des utilisateurs

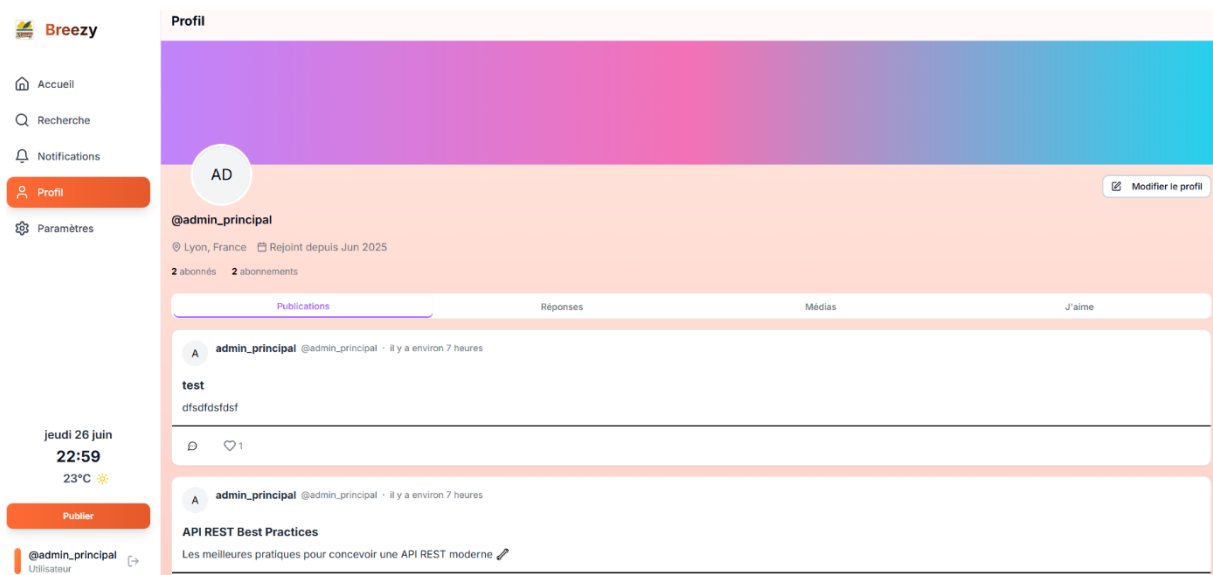


Figure 12 : Profil

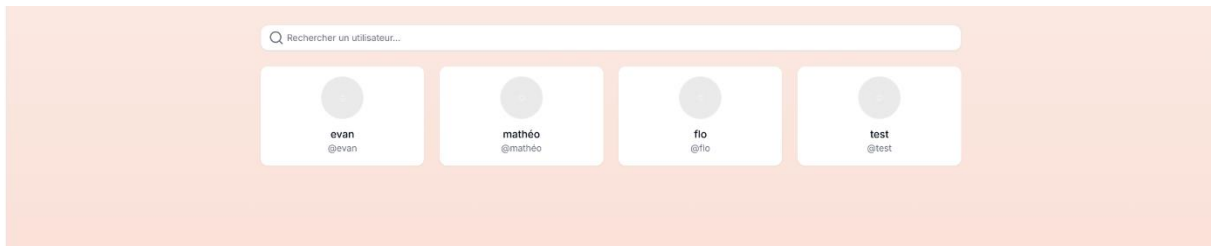


Figure 13 : Recherche d'utilisateurs

8. Intégration du projet :

Dans le cadre de notre projet, la mise en place d'un pipeline CI/CD (Intégration Continue / Livraison Continue) a été essentielle pour garantir la qualité, la cohérence et la sécurité de notre code. Voici comment nous avons configuré et intégré ces processus dans notre workflow de développement.

Linting et Cohérence du Code

Pour assurer une haute qualité de code et une cohérence à travers notre base de code, nous avons intégré des outils de linting dans notre pipeline CI/CD. Voici les étapes et les outils utilisés :

1. Configuration d'ESLint :

- **ESLint** est un outil de linting pour JavaScript et TypeScript qui permet d'identifier et de corriger les problèmes de style de code et les erreurs de programmation courantes. Nous avons configuré ESLint avec un ensemble de règles personnalisées pour répondre aux normes de codage de notre équipe.
- **Intégration dans le Pipeline CI/CD** : Un job spécifique a été ajouté à notre pipeline CI/CD pour exécuter ESLint à chaque commit. Ce job génère un rapport détaillé des erreurs et des avertissements, ainsi que des suggestions pour les corriger.

```
✓ Vérification du formatage avec Prettier 1s

1 ▶ Run cd backend
11 Checking formatting...
12 [warn] auth-service/src/app.js
13 [warn] auth-service/src/controllers/authController.js
14 [warn] auth-service/src/models/User.js
15 [warn] auth-service/src/routes/auth.js
16 [warn] posts-service/src/app.js
17 [warn] posts-service/src/controllers/posts.controller.js
18 [warn] posts-service/src/models/posts.model.js
19 [warn] users-service/src/app.js
20 [warn] users-service/src/controllers/users.controller.js
21 [warn] users-service/src/middleware/auth.middleware.js
22 [warn] users-service/src/models/users.model.js
23 [warn] users-service/src/routes/users.route.js
24 [warn] Code style issues found in 12 files. Run Prettier with --write to fix.
25 ✖ Des problèmes de formatage ont été détectés. Veuillez corriger les erreurs de formatage
    localement.
26 Pour corriger, exécutez les commandes suivantes :
27   cd backend
28   npx prettier --config ../../github/config/.prettierrc.js --ignore-path
    ../../github/config/.prettierignore --fix **/*.js npx prettier --config
    ../../github/config/.prettierrc.js --ignore-path ../../github/config/.prettierignore --fix **/*.json
29 Error: Process completed with exit code 1
```

Figure 14 : Alerte de vérification de formatage

2. Automatisation et Rapports :

- Le job de linting est exécuté automatiquement lors de chaque push vers notre dépôt de code. Les résultats sont affichés directement dans l'interface de notre outil CI/CD, ce qui permet aux développeurs de voir rapidement les problèmes et de les corriger avant qu'ils ne soient fusionnés dans la branche principale.
- **Avantages** : Cette approche permet de maintenir un code propre et cohérent, réduit les erreurs potentielles et facilite la collaboration entre les membres de l'équipe en assurant que tout le monde suit les mêmes normes de codage.

Vérification des Failles de Sécurité

La sécurité est une préoccupation majeure dans le développement de logiciels. Pour identifier et corriger les vulnérabilités potentielles, nous avons intégré Snyk dans notre pipeline CI/CD. Voici comment cela a été mis en œuvre :

1. Scan des Dépendances Node.js :

- **Snyk** est un outil de sécurité qui permet de scanner les dépendances de nos projets Node.js pour identifier les vulnérabilités connues. Nous avons configuré Snyk pour analyser nos fichiers package.json et package-lock.json à chaque exécution du pipeline.
- **Rapports de Vulnérabilité** : Snyk génère des rapports détaillés des vulnérabilités trouvées, incluant des descriptions des problèmes, leur niveau de gravité et des suggestions pour les corriger. Ces rapports sont accessibles directement dans notre outil CI/CD.

2. Scan des Images Docker :

- En plus des dépendances Node.js, nous avons également configuré Snyk pour scanner les images Docker utilisées dans notre projet. Cela permet d'identifier les vulnérabilités au niveau du système d'exploitation et des bibliothèques installées dans les conteneurs.
- **Intégration dans le Pipeline** : Le scan des images Docker est exécuté automatiquement lors de la construction des images dans le pipeline CI/CD. Les résultats sont affichés dans l'interface de l'outil CI/CD, permettant aux développeurs de prendre des mesures correctives avant le déploiement.

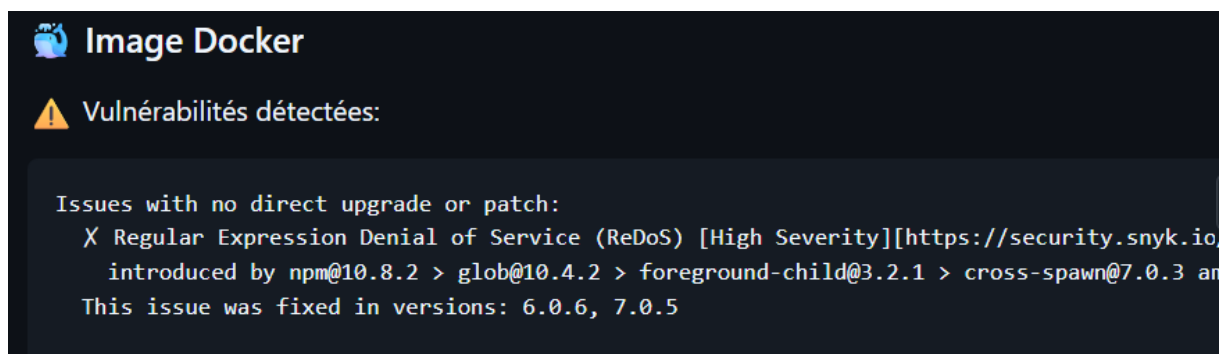



Figure 15 : Alerte de vérification des vulnérabilités sur une image Docker

3. Correction des Vulnérabilités :

- Les rapports générés par Snyk incluent des recommandations pour corriger les vulnérabilités identifiées. Cela peut inclure la mise à jour de dépendances, la modification de configurations ou l'application de correctifs spécifiques.
- **Avantages** : L'intégration de Snyk dans notre pipeline CI/CD permet de s'assurer que notre code est sécurisé avant le déploiement, réduisant ainsi les risques de failles de sécurité en production.

Dépendances Node.js

Total des dépendances: 0

 5 vulnérabilités trouvées:

- Regular Expression Denial of Service (ReDoS) [medium]
Package: null
Version: 4.17.15
Fix: 4.17.21
- Code Injection [high]
Package: null
Version: 4.17.15
Fix: 4.17.21
- Prototype Pollution [high]
Package: null
Version: 4.17.15
Fix: 4.17.20
- Prototype Pollution [high]
Package: null
Version: 4.17.15
Fix: 4.17.17
- Prototype Pollution [high]
Package: null
Version: 4.17.15
Fix: 4.17.17

Figure 16 : Vulnérabilités des dépendances

9. Gestion des autorisations :

Fonctionnalité (Fx)	Visiteur	Utilisateur	Modérateur	Administrateur
Fx1. Création de comptes utilisateurs	✓	✗	✗	✓
Fx2. Authentification sécurisée	✗	✓	✓	✓
Fx3. Publication de messages courts	✗	✓	✓	✓
Fx4. Affichage des messages sur le profil	✗	✓ (seulement le sien)	✓	✓
Fx5. Flux chronologique des messages	✗	✓	✓	✓
Fx6. Liker un post	✗	✓	✓	✓
Fx7. Répondre à un post sous forme de commentaire	✗	✓	✓	✓
Fx8. Répondre à un commentaire sur un post	✗	✓	✓	✓
Fx9. Suivre ou être suivi par d'autres utilisateurs	✗	✓	✓	✓
Fx10. Profil utilisateur avec informations de base	✗	✓	✓	✓
Fx11. Liste des messages publiés par l'utilisateur sur le profil	✗	✓	✓	✓

Figure 17 : Matrice des permissions

Le tableau ci-dessus indique que pour accéder aux fonctionnalités du site, il faut créer un compte au préalable

10. Conclusion :

En conclusion, ce projet a été une aventure enrichissante qui nous a permis de mettre en pratique une variété de technologies et de méthodologies modernes pour construire une application robuste, scalable et sécurisée. Grâce à l'utilisation de Docker pour la conteneurisation et l'orchestration des services, nous avons pu assurer une isolation efficace, une portabilité accrue et une gestion simplifiée des dépendances. L'intégration de Traefik comme API Gateway et Load Balancer a grandement facilité la gestion du trafic réseau et l'équilibrage de charge, assurant ainsi une haute disponibilité et une performance optimale de notre application.

Le développement du front-end avec React et Next.js a permis de créer une interface utilisateur dynamique et réactive, offrant une expérience utilisateur fluide et intuitive. L'approche mobile-first et l'utilisation de Tailwind CSS ont été déterminantes pour garantir une expérience cohérente et agréable sur tous les types d'appareils. La gestion des sessions avec JWT a renforcé la sécurité de notre application, tout en maintenant une expérience utilisateur personnalisée et sécurisée.

L'intégration de la CI/CD avec des outils de linting et de vérification des failles de sécurité a été cruciale pour maintenir la qualité et la sécurité de notre code. Ces processus automatisés nous ont permis de détecter et de corriger rapidement les problèmes, assurant ainsi un développement plus efficace et sécurisé.