Introduction

Contexte du projet

Dans un monde où la confidentialité des données est devenue une préoccupation majeure, la nécessité de solutions de communication sécurisées est plus pressante que jamais. Les plateformes de messagerie traditionnelles sont souvent la cible d'attaques telles que l'interception de données, le vol d'identifiants ou encore l'usurpation d'identité.

Avec l'essor du télétravail et de la collaboration en ligne, il est essentiel de garantir la protection des échanges, que ce soit pour les particuliers ou les entreprises. C'est dans ce contexte que notre projet **XXX** a été initié. L'objectif est de fournir une plateforme de messagerie instantanée qui allie sécurité, confidentialité et facilité d'utilisation.

Notre solution repose sur une infrastructure locale (**on-premise**) permettant un contrôle total sur les données échangées, tout en intégrant des mécanismes avancés de sécurité, notamment l'**authentification multi-facteurs (MFA)** et le **chiffrement de bout en bout**.

Problématiques à résoudre

Le projet vise à répondre aux défis suivants :

1. **Manque de sécurité dans les échanges de données**
    - Les solutions de messagerie classiques utilisent des mesures de sécurité insuffisantes, exposant les données des utilisateurs aux interceptions et violations de confidentialité.
2. **Usurpation d'identité et accès non autorisé**
    - L'utilisation exclusive d'un mot de passe n'est plus suffisante pour protéger les comptes des utilisateurs contre les tentatives de piratage (brute force, phishing, etc.).
3. **Gestion des sessions utilisateurs sécurisées**
    - Garantir que les utilisateurs restent connectés de manière sécurisée sans risque de détournement de session.
4. **Contrôle total des données par l'utilisateur ou l'organisation**
    - Les services cloud ne garantissent pas toujours une transparence totale sur la gestion et la localisation des données sensibles.
5. **Absence de gestion des clés de chiffrement par l'utilisateur**

- La plupart des plateformes gèrent les clés côté serveur, ce qui représente un risque en cas de compromission du serveur central.

Objectifs généraux

Notre projet a pour ambition de :

1. **Offrir une solution de communication sécurisée**
    - Développer une application de chat qui garantit la confidentialité des échanges grâce à des protocoles de chiffrement robustes.
2. **Garantir l'authentification forte des utilisateurs avec MFA**
    - Implémenter un processus d'authentification en plusieurs étapes (mot de passe + code 2FA) pour renforcer la sécurité de l'accès aux comptes.
3. **Assurer la confidentialité et l'intégrité des messages**
    - Mettre en œuvre un chiffrement de bout en bout pour que seuls les utilisateurs autorisés puissent lire les messages échangés.
4. **Faciliter l'utilisation tout en maintenant un haut niveau de sécurité**
    - Concevoir une interface utilisateur moderne et intuitive sans compromis sur les fonctionnalités de sécurité.
5. **Garantir une infrastructure adaptable et évolutive**
    - Permettre l’hébergement de la solution en local et prévoir une transition future vers le cloud si nécessaire.

I. Description du Projet

1. Fonctionnalités principales

Le projet **XXX** repose sur plusieurs fonctionnalités clés visant à assurer la sécurité, la confidentialité et la facilité d'utilisation de la plateforme.

Authentification multi-facteurs (MFA)

L'authentification des utilisateurs repose sur un mécanisme de **MFA (Multi-Factor Authentication)**, combinant plusieurs éléments pour renforcer la sécurité des comptes.

Cette authentification repose sur :

- **Mot de passe sécurisé** : Les mots de passe des utilisateurs sont stockés de manière sécurisée via un algorithme de hachage robuste (bcrypt).
- **Code de vérification temporaire (OTP)** : Un code à usage unique généré par une application d'authentification (Google Authenticator, Authy).
- **Gestion des sessions utilisateur** : Implémentation d'une gestion des sessions sécurisées pour empêcher l'usurpation d'identité.

Flux de travail de l'authentification MFA :

1. L'utilisateur saisit ses identifiants (nom d'utilisateur/mot de passe).
2. Le serveur MFA génère un QR code pour la configuration du 2FA.
3. L'utilisateur scanne le QR code pour activer le MFA.
4. À chaque connexion, l'utilisateur doit fournir le code OTP pour finaliser l'accès.

Messagerie instantanée chiffrée

La communication entre les utilisateurs sera protégée par un **chiffrement de bout en bout (E2EE)** pour garantir la confidentialité des échanges.

- **Génération de clés publiques/privées côté client** : Chaque utilisateur génère ses propres clés de chiffrement lors de l'inscription.
- **Échange sécurisé des clés publiques** : Le serveur stocke uniquement les clés publiques pour permettre le chiffrement des messages.
- **Transmission sécurisée des messages** : Tous les messages envoyés sont chiffrés avant d'être transmis via des sockets sécurisés.

Flux de travail du chiffrement :

1. L'expéditeur chiffre son message avec la clé publique du destinataire.
2. Le message chiffré est envoyé via le serveur backend.
3. Le destinataire le déchiffre localement avec sa clé privée.

Gestion des utilisateurs

Le projet intègre une gestion complète des utilisateurs, permettant :

- **Création de compte** : Inscription avec nom d'utilisateur, mot de passe et activation du MFA.
- **Modification des informations personnelles** : Mise à jour des informations de profil.
- **Suppression de compte** : Permet à un utilisateur de se désinscrire en supprimant définitivement ses données.

Gestion des sessions sécurisées

Une fois l'utilisateur connecté, une session sécurisée est établie et maintenue avec des mécanismes de protection avancés :

- **Sessions basées sur des cookies sécurisés et expirations dynamiques.**
- **Vérification des jetons JWT pour une gestion efficace de l'accès.**
- **Expiration automatique des sessions après une période d'inactivité.**

Historique des connexions (logs)

Pour assurer la traçabilité et la sécurité des accès, chaque connexion utilisateur est enregistrée dans un journal des événements incluant :

- **Date et heure de connexion.**
- **Adresse IP de l'utilisateur.**
- **Événements de connexion et d'échec d'authentification.**
- **Tentatives de connexion suspectes.**

Ces logs permettent aux administrateurs de surveiller l’activité et d'identifier toute tentative d'accès non autorisé.

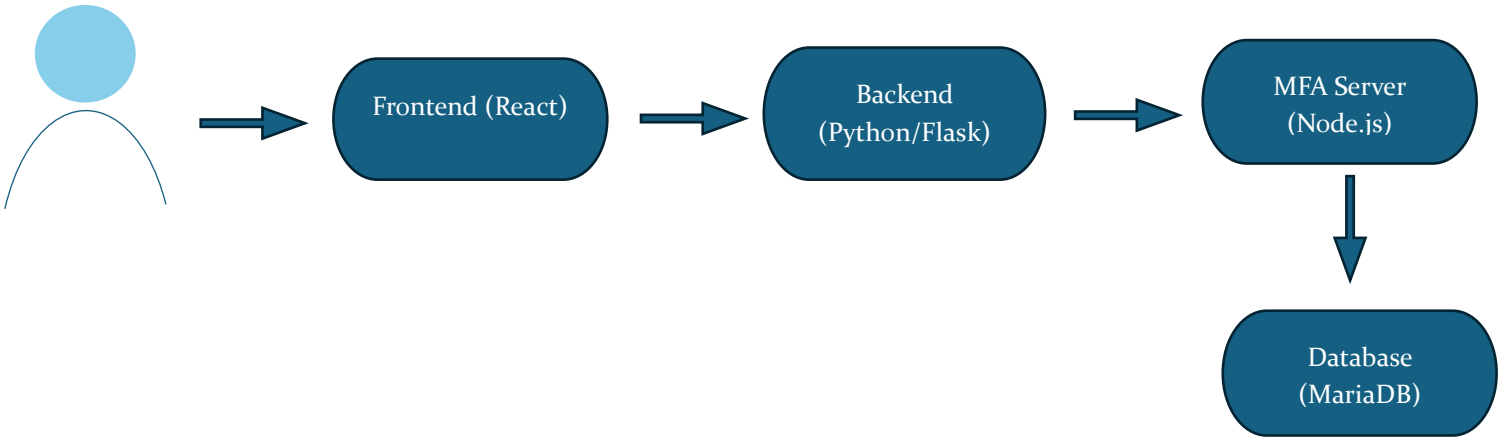
2. Architecture globale

L'architecture du projet repose sur une séparation claire des composants pour améliorer la sécurité, les performances et la maintenabilité.

Présentation des différents serveurs

1. **Serveur MFA (Javascript, VM)**
    - Fournit les services d'authentification MFA.
    - Gère l'enregistrement et la vérification des utilisateurs.
    - Génère et stocke les QR codes pour la configuration 2FA.
    - Communication via une API REST sécurisée (HTTPS).
2. **Serveur Backend (Python, VM)**
    - Responsable de la gestion des utilisateurs, des sessions et des messages.
    - Implémente les sockets pour les échanges en temps réel.
    - Connecté à la base de données pour stocker les messages et utilisateurs.
3. **Serveur Base de données (MariaDB, VM)**
    - Stocke toutes les données utilisateur et les messages chiffrés.
    - Séparé du backend pour une sécurité renforcée.
    - Tables principales : users, messages, mfa_logs.
4. **Serveur Frontend (React/Vite, VM)**
    - Interface utilisateur accessible via un navigateur web.
    - Communication avec le backend via des appels API sécurisés.
    - Fournit les fonctionnalités d'inscription, de connexion et de chat.

Flux de données entre les composants



1. **Inscription et authentification**
    - Le frontend envoie les informations d'inscription au backend.
    - Le backend communique avec le serveur MFA pour l'activation 2FA.
    - Une fois validée, les informations sont enregistrées dans la base de données.
2. **Envoi et réception de messages**
    - L'utilisateur envoie un message via le frontend.
    - Le backend chiffre et stocke le message dans la base de données.
    - Le destinataire reçoit le message en le déchiffrant avec sa clé privée.
3. **Gestion des connexions**
    - Chaque utilisateur connecté dispose d'une session active suivie par le backend.
    - Les logs d'accès sont stockés dans la base de données pour audit.

II. Exigences Fonctionnelles

Cette section décrit les fonctionnalités essentielles du projet **XXX**, en mettant l'accent sur la sécurité et l'expérience utilisateur.

1. Inscription et Connexion des utilisateurs

L'inscription et la connexion des utilisateurs doivent être sécurisées et inclure un mécanisme d'authentification forte (MFA) pour garantir la protection des comptes.

Inscription des utilisateurs

L'inscription d'un utilisateur suit les étapes suivantes :

1. **Saisie des informations personnelles** :
    - Nom d'utilisateur (unique).
    - Mot de passe sécurisé (haché côté serveur avant stockage).
2. **Configuration du MFA** :
    - Génération d'un QR code envoyé au frontend pour activation dans une application d'authentification (Google Authenticator, Authy).
    - L'utilisateur scanne le QR code pour activer le 2FA.
3. ***Génération des clés directement côté client sans intervention du serveur*.** :
    - les clés privées resteront exclusivement sur le client, tandis que seules les clés publiques seront stockées sur le serveur pour chiffrer les messages.
4. **Validation et stockage** :
    - Les informations sont envoyées au backend pour stockage sécurisé dans la base de données.

Critères d'acceptation :

- Le processus d'inscription ne doit être validé qu'après l'activation du 2FA.
- Le mot de passe et la clé privée ne doivent jamais être stockés en clair sur le serveur.

Connexion sécurisée via 2FA

Le processus de connexion suit les étapes suivantes :

1. **Saisie des identifiants (nom d'utilisateur + mot de passe).**
2. **Validation des identifiants dans la base de données.**
3. **Envoi de la demande de validation au serveur MFA.**
4. **Saisie du code OTP généré par l'application d'authentification.**
5. **Si le code est valide, ouverture d'une session sécurisée avec génération d'un jeton JWT.**

Critères d'acceptation :

- Un utilisateur ne peut accéder à son compte qu'après validation complète du MFA.
- Une session doit être maintenue via un jeton JWT et une expiration automatique après une période d'inactivité.

2. Gestion des messages

L'application doit permettre aux utilisateurs d'échanger des messages en toute sécurité grâce au chiffrement de bout en bout.

Envoi et réception chiffrée de bout en bout

1. **Chiffrement du message côté client avec la clé publique du destinataire.**
2. **Transmission sécurisée via le backend en utilisant des sockets sécurisés.**
3. **Stockage du message chiffré dans la base de données MariaDB.**
4. **Déchiffrement côté destinataire avec sa clé privée locale.**

Critères d'acceptation :

- Aucun message ne doit être stocké en clair dans la base de données.
- La communication entre les serveurs et les clients doit être sécurisée via SSL/TLS.
- Les messages doivent être lisibles uniquement par l’expéditeur et le destinataire.

Historique des messages stockés de manière sécurisée

1. **Enregistrement des messages chiffrés avec des métadonnées (expéditeur, destinataire, horodatage).**
2. **Possibilité pour l'utilisateur de consulter son historique de discussion.**
3. **Suppression automatique des messages après une période définie.**

Critères d'acceptation :

- L'historique des messages ne peut être consulté que par l'utilisateur authentifié.
- Les messages supprimés doivent être irrécupérables du côté du serveur.

3. Gestion des clés de chiffrement

La gestion des clés de chiffrement est essentielle pour garantir la sécurité et la confidentialité des communications.

Génération et stockage des clés privées/publiques côté client

1. **Génération dynamique des clés dans le navigateur à l'aide de bibliothèques de cryptographie côté client, comme *Web Crypto API*..**
2. **Stockage sécurisé de la clé privée sur le client dans un espace sécurisé du navigateur (localStorage, IndexedDB, ou secure enclave des appareils mobiles).**
3. **Envoi de la clé publique au backend pour partage avec d'autres utilisateurs.**

Critères d'acceptation :

- La clé privée ne doit jamais être exposée au serveur.
- La clé publique doit être accessible pour les autres utilisateurs afin de permettre le chiffrement.

Mécanisme de renouvellement des clés – optionel –

1. **L'utilisateur peut demander un renouvellement manuel de ses clés.**
2. **Un mécanisme automatique de renouvellement des clés doit être mis en place tous les X mois.**
3. **Les anciennes clés doivent être révoquées et supprimées après une migration réussie des nouvelles clés.**

Critères d'acceptation :

- Les nouvelles clés doivent être appliquées à tous les nouveaux messages.
- Les anciennes clés doivent être utilisées uniquement pour la consultation de l'historique.

III. Exigences Techniques

Cette section détaille les choix technologiques ainsi que les exigences de sécurité, de performance et de scalabilité du projet **XXX**, garantissant un fonctionnement fiable et sécurisé.

1. Technologies choisies

Le projet repose sur un ensemble de technologies modernes et éprouvées, garantissant sécurité, performance et maintenabilité. 😊 ***Ca pu du GTP la*** 😊

Backend MFA

- **Technologie :** Node.js
- **Framework :** Express
- **Modules utilisés :**
    - **Passport.js** : Gestion de l'authentification utilisateur avec support des sessions.
    - **Speakeasy** : Implémentation de l'authentification multi-facteurs (TOTP).
    - **Express-session** : Gestion des sessions sécurisées.

Backend Principal

- **Technologie :** Python
- **Framework :** Flask
- **Modules utilisés :**
    - **Flask-SocketIO** : Gestion des connexions en temps réel via WebSockets.
    - **Cryptography** : Chiffrement des données sensibles.
    - **JWT (PyJWT)** : Gestion des jetons de session sécurisés.

Base de données

- **Système :** MariaDB
- **Motivation du choix :**
    - Performances élevées pour les requêtes complexes.
    - Sécurisation des données avec options avancées d'authentification et de chiffrement.
    - Facilité de migration vers des solutions cloud (AWS RDS, Google Cloud SQL).

Frontend

- **Technologie :** React.js
- **Build tool :** Vite (pour un développement rapide et une performance optimisée).
- **Bibliothèques utilisées :**
    - **Axios** : Communication avec les API backend.
    - **React Router** : Gestion de la navigation.
    - **Bootstrap/Tailwind CSS** : Interface utilisateur réactive et moderne.
    - Web Crypto API ou CryptoJS : pour garantir une gestion sécurisée des clés

2. Sécurité

La sécurité est un aspect fondamental du projet. Plusieurs mesures ont été adoptées pour assurer la protection des données et des communications.

Certificats SSL/TLS

- Mise en place de certificats auto-signés pour l'environnement de développement.
- Intégration de certificats SSL/TLS valides lors du déploiement en production.
- Protection des communications entre :
    - Le frontend et le backend.
    - Le backend principal et le serveur MFA.

Chiffrement des données

- **Données en transit** :
    - Chiffrement avec SSL/TLS.
    - Utilisation des WebSockets sécurisés (wss://).
- **Données au repos** :
    - Hashage des mots de passe avec **bcrypt**.
    - Chiffrement des messages avec RSA/ AES avant stockage.

Sécurisation des sessions

- **Jetons JWT sécurisés** :
    - Signés avec une clé secrète (HS256).
    - Expiration automatique après une période définie.
- **Cookies sécurisés** :
    - HttpOnly pour empêcher les accès via JavaScript.
    - Secure pour transmission uniquement sur HTTPS.
    - SameSite pour éviter les attaques CSRF.

Critères d'acceptation :

- Toute tentative d'accès non autorisée doit être détectée et enregistrée dans les logs de sécurité.
- Aucun mot de passe ou clé privée ne doit être stocké en clair.

3. Performances

L'application doit être conçue pour offrir une expérience utilisateur fluide et réactive, même sous forte charge.

Gestion des connexions simultanées

- Implémentation d'un système de gestion efficace des WebSockets pour gérer les connexions multiples.
- Optimisation de l'utilisation de la mémoire et des threads pour éviter les surcharges du serveur.

Temps de réponse optimal

- Mise en cache des requêtes fréquentes pour réduire la latence (ex. Redis, Memcached).
- Réduction du temps de génération des QR codes lors de la configuration du 2FA.
- Temps de réponse cible pour :
    - Inscription : < 2 secondes.
    - Connexion : < 1,5 seconde.
    - Envoi de message : < 500 ms.

Tests de charge 

- Simulation de 1000 utilisateurs simultanés pour évaluer les performances.
- Surveillance du temps de réponse moyen sous forte charge.

4. Scalabilité

L'application est conçue pour s'adapter à une croissance future et supporter de nombreux utilisateurs.

Évolutivité horizontale et verticale

- Ajout possible de nouveaux serveurs pour distribuer la charge en utilisant un load balancer (Nginx, HAProxy).
- Optimisation des ressources serveur (RAM, CPU) en fonction de la demande.

Déploiement Cloud-ready

- Intégration future avec des services cloud tels que :
    - **AWS** : Utilisation d'EC2 pour le backend et RDS pour la base de données.
    - **Google Cloud** : Utilisation de Compute Engine et Cloud SQL.
    - **Azure** : Déploiement sur App Services avec gestion des identités via Azure AD.

Conteneurisation

- Dockerisation des services pour assurer la portabilité et la reproductibilité de l'environnement.
- Orchestration avec Kubernetes pour la gestion automatique du scaling.

Critères d'acceptation :

- L'architecture doit permettre d'ajouter de nouveaux services sans impacter l'existant.
- Le projet doit pouvoir être déployé en local comme sur le cloud sans modifications majeures.

IV. Infrastructure et Hébergement

Cette section décrit l'architecture matérielle et logicielle mise en place pour héberger les différents services du projet **XXX**, ainsi que les plans d'évolution future vers le cloud.

1. Serveurs utilisés

L'infrastructure actuelle repose sur trois serveurs distincts, chacun ayant un rôle spécifique afin d'assurer la modularité, la sécurité et la performance du système.

Serveur	Système d'exploitation	Rôle	Composants hébergés
Serveur MFA	... Linux	Gestion de l'authentification MFA	Backend MFA (Node.js, Express, Passport.js)
Serveur Base de données	... Linux	Gestion des données utilisateur	MariaDB (Stockage sécurisé)
Serveur Principal	Windows / Linux	Backend et Frontend principaux	Flask (Backend) + React (Frontend)

Serveur MFA (Linux)

- **Rôle :** Gestion de l’authentification des utilisateurs via MFA.
- **Services hébergés :**
    - Node.js avec Express pour gérer les routes d'authentification.
    - Passport.js pour la gestion des sessions et de l'authentification locale.
    - Speakeasy pour la génération et la vérification des codes 2FA.
- **Ports ouverts :**
    - 7002 pour l'API d'authentification.

Serveur Base de données (Linux)

- **Rôle :** Stockage sécurisé des informations utilisateur et des messages.
- **Services hébergés :**
    - MariaDB pour la gestion des données utilisateurs, messages et logs MFA.
- **Sécurisation :**
    - Accès restreint par adresse IP autorisée uniquement.
    - Connexions sécurisées via TLS/SSL.
- **Ports ouverts :**
    - 3306 pour l'accès à la base de données.

Serveur Principal (Windows / Linux)

- **Rôle :** Gestion de l'application principale (authentification, messagerie et UI).
- **Services hébergés :**
    - Backend Flask pour la gestion des sockets et de la logique métier.
    - Frontend React pour l'interface utilisateur.
- **Sécurisation :**
    - JWT pour l'authentification utilisateur côté backend.
    - Chiffrement des communications via HTTPS.

- **Ports ouverts :**
    - 5000 pour le backend Flask.
    - 3000 pour le frontend React.

2. Réseautage

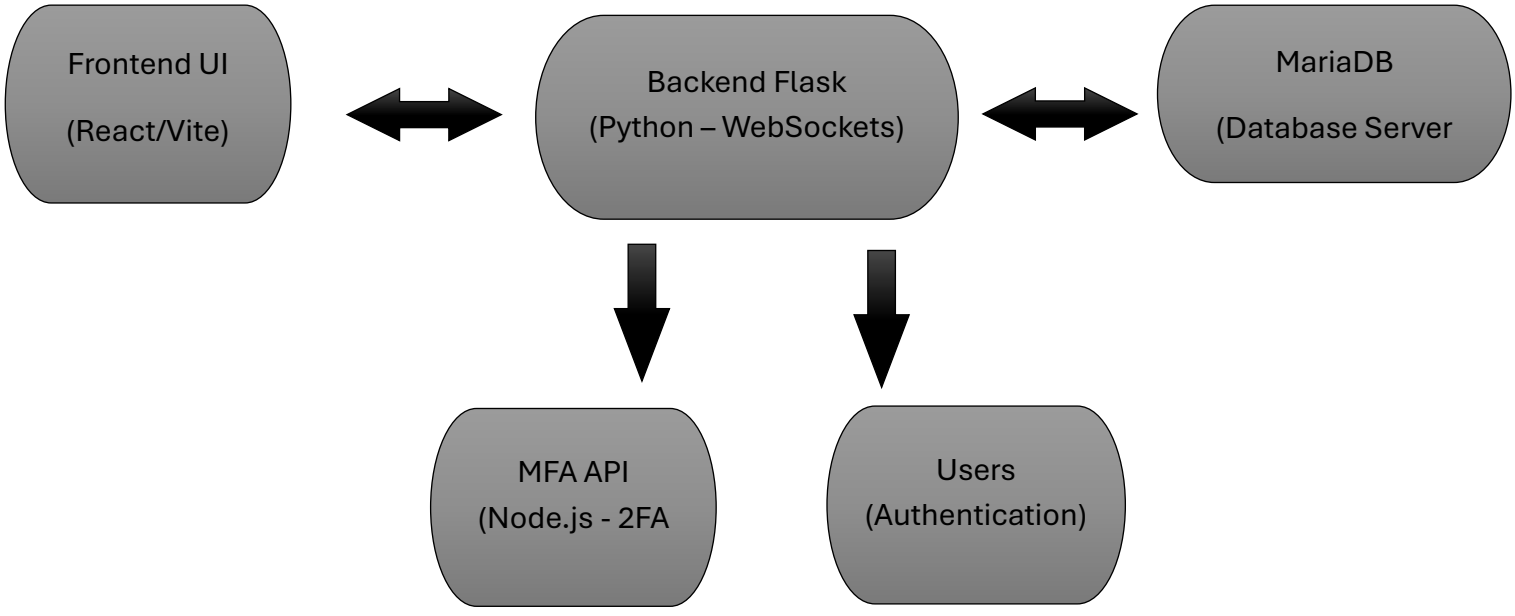
La communication entre les différents serveurs repose sur un réseau local sécurisé, en utilisant des IP privées pour éviter toute exposition non autorisée sur Internet.

Adressage IP interne

Chaque serveur est configuré avec une adresse IP privée statique pour garantir une communication fiable et prévisible :

Serveur	Adresse IP	Description
Serveur MFA	x.x.x.10	Gestion de l'authentification MFA
Serveur Base de données	x.x.x.20	Gestion des données
Serveur Principal	x.x.x.30	Application backend et frontend

Schéma des flux de communication



Sécurité réseau

Des mesures de sécurité sont appliquées au niveau du réseau :

- **Pare-feu (iptables/ufw)** pour restreindre l'accès aux services uniquement aux IP autorisées.
- **Segmentation réseau** pour isoler les différents services et éviter les accès non autorisés.
- **VPN interne** pour les connexions à distance des membres de l'équipe. (hum je ne sais pas comment c'est possible, mais bon va savoir)

3. Plan de migration future vers le cloud

Afin de garantir une évolutivité et une meilleure résilience du système, nous envisageons une migration progressive vers une infrastructure cloud.

Objectifs de la migration cloud

- **Meilleure disponibilité et scalabilité** : Hébergement sur des services tels qu'AWS, Azure, ou Google Cloud.
- **Réduction des coûts de maintenance** : Moins de gestion matérielle et flexibilité accrue.
- **Sécurisation accrue** : Mécanismes de chiffrement avancés et gestion fine des accès.

Scénario de déploiement cloud

Composant	Cloud Service Proposé	Raison du choix
Backend MFA	AWS Lambda / EC2	Déploiement sans gestion d'infrastructure
Backend Principal	AWS ECS / Azure App Service	Gestion automatisée et scalabilité facile
Base de données	AWS RDS / Google Cloud SQL	Base managée avec haute disponibilité
Frontend	AWS S3 / Cloudflare Pages	Distribution rapide et fiable du frontend

Étapes de migration prévues

1. **Phase 1 : Test en environnement cloud hybride**
    - Déploiement du backend sur AWS EC2 tout en maintenant la base de données sur serveur local.
    - Validation des performances et de la sécurité.
2. **Phase 2 : Migration complète**
    - Déplacement progressif des données vers une base cloud (AWS RDS ou GCP SQL).
    - Intégration d'un load balancer pour distribuer la charge entre plusieurs instances backend.
3. **Phase 3 : Optimisation et monitoring**
    - Mise en place de monitoring avec AWS CloudWatch ou Prometheus.
    - Ajustement des ressources en fonction de l'utilisation réelle.

V. Déploiement et Outils de Collaboration

Cette section décrit les outils et processus adoptés pour le développement, les tests et la collaboration au sein de l'équipe du projet **XXX**, ainsi que les étapes clés du déploiement.

1. Environnement de développement

L'environnement de développement est structuré pour assurer une collaboration fluide et un suivi efficace des progrès.

Outils de gestion du code source

- **GitHub** :
    - Hébergement du code source.
    - Suivi des modifications via le versioning Git.
    - Gestion des branches pour le développement de nouvelles fonctionnalités.

- Utilisation des **Pull Requests (PR)** pour examiner et fusionner le code.
- Automatisation CI/CD pour les tests et déploiements via GitHub Actions.

Branches principales :

1. main – Version stable de production.
2. develop – Dernières fonctionnalités en cours de développement.
3. feature/* – Branches dédiées aux nouvelles fonctionnalités.

Outils de communication et coordination

L'équipe utilise plusieurs outils pour assurer une communication efficace et une bonne coordination :

Outil	Utilisation
WhatsApp	Discussions rapides et suivi quotidien
Microsoft Teams	Réunions hebdomadaires et partage de documents
GitHub Issues	Suivi des tâches et des bugs

2. Procédures de tests

Afin d'assurer la qualité et la fiabilité du système, plusieurs niveaux de tests sont mis en place :

Tests unitaires

- Vérification des fonctionnalités individuelles des modules (MFA, messagerie, gestion des utilisateurs).
- Technologies utilisées :
  - **Backend MFA (Node.js)** : Mocha, Chai
  - **Backend Principal (Python Flask)** : PyTest
  - **Frontend (React)** : Jest, React Testing Library

Tests d'intégration

Ces tests assurent que les interactions entre les différents services fonctionnent correctement.

- Tests de communication entre le **backend principal** et le **serveur MFA**.
- Tests de la gestion des sessions et de l'envoi/réception de messages via WebSockets.
- Simulations d’interactions complètes de l'utilisateur (de l'inscription jusqu'à la messagerie).

Tests de sécurité

Des vérifications de sécurité sont effectuées pour garantir la protection des données et des sessions utilisateur :

- Tests contre les attaques **SQL Injection**, **XSS**, et **CSRF**.
- Vérification du chiffrement des données avec **TLS/SSL**.
- Audit des permissions d'accès aux ressources sensibles.

Outils de tests automatisés

Pour automatiser les tests, les outils suivants seront utilisés :

- **Postman** pour les tests d'API automatisés.
- **Selenium** pour tester l'interface utilisateur.
- **JMeter** pour les tests de performance et de charge.

3. Échéancier prévisionnel

L'équipe suit un calendrier de développement en **sprints hebdomadaires**, avec des objectifs précis à atteindre à chaque étape.

Sprint	Objectif	Deadline	Responsable
1	Configuration des serveurs (MFA, DB, Backend)	---	Tapsoba
2	Finalisation du serveur MFA	---	Tapsoba
3	Implémentation de la messagerie sécurisée	---	Ulrich, Othniel et Tapsoba
4	Développement du frontend (React)	---	Nancy
5	Tests d'intégration et sécurité	---	Toute l'équipe
6	Mise en production locale	---	Toute l'équipe

4. Stratégie de déploiement

L'objectif est de garantir un déploiement progressif et contrôlé du système sur l'environnement local, avant une éventuelle migration vers le cloud.

Étapes de déploiement en environnement local

1. **Déploiement du serveur MFA**
    - Hébergement sur le serveur Linux.
    - Configuration de l'authentification sécurisée via HTTPS.
2. **Déploiement de la base de données**
    - Mise en place sur le serveur Debian.
    - Configuration de la connexion avec des accès restreints.
3. **Déploiement du backend principal**
    - Lancement du backend Flask pour gérer les sockets.
    - Tests de connexion entre le backend et le serveur MFA.
4. **Déploiement du frontend**
    - Hébergement du frontend React sur le serveur principal.
    - Tests des fonctionnalités utilisateur complètes.

Stratégie de mise à jour

Chaque mise à jour suivra un processus bien défini :

1. Déploiement en **environnement de test** avant mise en production.
2. Validation via les tests automatisés.
3. Déploiement progressif sur les différents serveurs.
4. Suivi des performances et correction des éventuels bugs.

5. Formation et documentation

Afin d'assurer une bonne prise en main du projet, une documentation détaillée sera fournie à l'équipe :

- **Documentation technique** :
    - Installation et configuration des serveurs.
    - API et endpoints du système.
    - Procédures de déploiement.
- **Documentation utilisateur** :
    - Guide d'utilisation de l'application.
    - Procédures de connexion et de gestion du compte.

VI. Contraintes et Risques

Cette section décrit les défis et les limites du projet, ainsi que les risques identifiés pouvant affecter son bon déroulement.

1. Contraintes de sécurité

Étant donné la nature sensible des communications utilisateur, plusieurs mesures de sécurité doivent être mises en place pour assurer une protection optimale :

Protection contre les attaques web

- **XSS (Cross-Site Scripting)** :
    - Filtrage et évasion des entrées utilisateur côté frontend et backend.
    - Utilisation d'en-têtes HTTP sécurisés (Content-Security-Policy, X-XSS-Protection).
- **CSRF (Cross-Site Request Forgery)** :
    - Implémentation de tokens CSRF pour sécuriser les requêtes sensibles.
    - Restriction des cookies aux mêmes sites avec l'attribut SameSite.

Sécurisation des échanges inter-serveurs

- Chiffrement des communications via **SSL/TLS** pour empêcher l'interception des données.
- Mise en place de **pare-feu et règles d'accès strictes** entre les serveurs pour limiter les connexions non autorisées.
- Utilisation de **JWT (JSON Web Tokens)** pour sécuriser les échanges entre services.

2. Contraintes techniques

Le projet est confronté à plusieurs contraintes liées aux ressources et aux capacités techniques actuelles :

1. **Limitation des ressources matérielles actuelles**
    - Le déploiement initial est réalisé sur des machines locales (Kali, Debian, machine hôte) (Chez Tapsoba).
    - Risque de surcharge en cas d'un grand nombre d'utilisateurs simultanés.
    - Plan de migration vers le cloud nécessaire à moyen terme pour la scalabilité.
2. **Complexité de l'intégration des technologies**
    - Nécessité de coordonner les différentes technologies : **Node.js, Python, MariaDB, React.**
    - Gestion des dépendances et des compatibilités entre les frameworks et bibliothèques.

3. **Maintenance et évolutivité**
    - Anticipation des mises à jour et correctifs de sécurité.
    - Documentation rigoureuse pour faciliter la prise en main par de nouveaux membres.

3. Risques potentiels

Plusieurs risques sont identifiés et des solutions préventives sont envisagées pour éviter d'éventuels retards ou complications.

Risque	Impact potentiel	Stratégie d'atténuation
Retard dans l'intégration des modules	Allongement du calendrier de projet	Suivi rigoureux et réunions hebdomadaires
Problèmes de compatibilité des services	Dysfonctionnements imprévus	Tests continus d'intégration
Vulnérabilités de sécurité découvertes	Perte de données sensibles	Audits de sécurité réguliers
Manque de disponibilité des membres	Ralentissement du développement	Répartition des tâches équilibrée

VII. Critères de Réussite

Pour évaluer la réussite du projet, plusieurs critères doivent être respectés afin de garantir que le produit final répond aux exigences initiales.

1. Fonctionnalité du système

Inscription et connexion sécurisée via MFA

- L'utilisateur doit pouvoir s'inscrire et activer l'authentification à deux facteurs sans difficulté.
- La connexion doit être sécurisée avec un OTP valide et une gestion des sessions efficace.

Messagerie fonctionnelle

- Envoi et réception des messages chiffrés avec un temps de latence minimal.
- Les messages doivent être stockés de manière sécurisée sans fuite d'information.

Gestion efficace des utilisateurs

- Création, suppression et gestion des sessions utilisateur doivent être fluides.
- Logs détaillés des connexions pour garantir la traçabilité.

2. Performance et sécurité

- **Temps de réponse optimisé**
  - Les opérations d'authentification et de messagerie doivent répondre en moins de 1 seconde en environnement local.
- **Sécurisation des données**
  - Aucune donnée sensible ne doit être stockée en clair.
  - Toutes les communications doivent être chiffrées en transit et au repos.
- **Fiabilité de l'infrastructure**
  - Tolérance aux pannes des composants clés (MFA, base de données, messagerie).
  - Système capable de gérer au moins 50 utilisateurs simultanés en environnement local.

3. Expérience utilisateur

- **Interface intuitive et réactive**
  - Le frontend doit être fluide et facile à naviguer sur desktop et mobile.
  - Les utilisateurs doivent recevoir des messages d'erreurs clairs en cas de problème.

- **Accessibilité sur différentes plateformes**
    - Le système doit être compatible avec les navigateurs courants (Chrome, Firefox, Edge).
    - Support minimal pour les appareils mobiles via responsive design.

4. Livrables attendus

Pour garantir la livraison complète du projet, les livrables suivants doivent être fournis :

Livrable	Description	Responsable
Cahier des charges finalisé	Document détaillant toutes les exigences du projet	Tapsoba
Serveur MFA opérationnel	API d'authentification fonctionnelle	Tapsoba
Backend de messagerie fonctionnel	Gestion des connexions et des messages sécurisés	Ulrich, Othniel et Tapsoba
Frontend utilisateur	Interface fonctionnelle et intuitive	Nancy