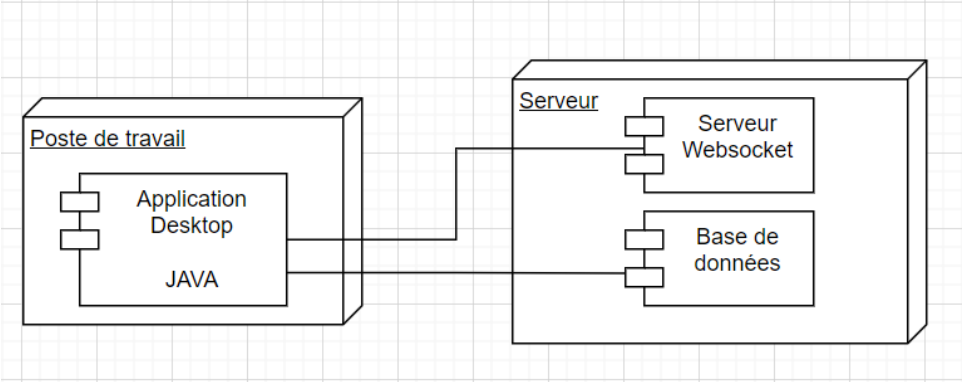
Projet 2

Concept : Développement d'une application de tchat permettant la communication en temps réel entre différents utilisateurs, intégrant à la fois des discussions individuelles et de groupe, ainsi que des canaux publics ou privés.

Architecture :



Fonctionnalités :

- **Sélectionner un destinataire :**

  - Affichage de la liste des contacts ou des groupes existants.
  - Option de recherche pour filtrer les contacts.

- **Envoyer un message :**

  - Interface de saisie de texte.
  - Bouton d'envoi qui transmet le message au serveur.
  - Gestion des types de messages (texte, images, fichiers).

- Recevoir un message :

  - Réception en temps réel via WebSocket.
  - Affichage dynamique des messages entrants dans le fil de discussion.

- **Rechercher dans la liste des messages :**

  - Barre de recherche dans une conversation donnée.
  - Indexation et filtrage des messages par mots-clés.

- **Épingler un message :**

  - Option de menu sur un message pour l'épingler.
  - Affichage du message épinglé en haut de la conversation.

- **Accéder à son historique de message :**

  - Chargement des discussions passées.
  - Pagination ou chargement infini pour les anciens messages.

- **Créer un canal :**

  - Permettre aux utilisateurs de créer un canal public ou privé.
  - Définir un nom et une description pour le canal.

- S'abonner / Se désabonner à un canal :

  - Permettre aux utilisateurs de rejoindre ou quitter un canal.
  - Gestion des permissions et notifications selon le type de canal (public/privé).

- Envoyer et recevoir des messages dans un canal :

  - Fonctionnalité similaire à l'envoi/réception dans une discussion individuelle, mais au niveau d'un canal.
  - Support de l'affichage des messages envoyés par divers utilisateurs au sein du canal.

- Rechercher et épingler des messages dans un canal :

  - Rechercher des messages spécifiques dans l'historique d'un canal.
  - Épingler des messages importants pour un accès rapide par tous les membres du canal.

- l'historique des messages d'un canal :

  - Charger les messages passés d'un canal avec pagination ou chargement infini.

- **Mise à jour automatique :**

  - Permettre un système de mise à jour automatique de l'application.

Stack technologique :

- **Base de données : mariadb**
- **Backend Java :**
  - **Framework :** Spring Boot
  - **Communication en temps réel :** Utilisation de Spring WebSocket pour la gestion des messages en temps réel.
  - **Base de données :** Utilisation de JPA (Hibernate) pour interagir avec une base de données relationnelle (PostgreSQL, MySQL, etc.).
- **Frontend Java :**
  - **Framework UI :** JavaFX pour créer une interface de bureau réactive.
  - **Gestion d'état local :** Utilisation des modèles de conception JavaFX (MVC) pour la gestion de l'interface.

Méthodes de déploiement :

Déploiement du serveur

1. Containerisation avec Docker Compose :

- **Isolation et portabilité :**
  Le serveur est containerisé à l'aide de Docker, ce qui permet de garantir la cohérence entre les environnements de développement, de test et de production. Chaque composant (application Spring Boot, base de données MariaDB) est encapsulé dans son propre conteneur.
- **Fichier dev.yaml :**
  Un fichier dev.yaml gère la configuration des services pour le developpement :
  - **Service Backend :**
    Utilise une image Docker construite à partir du Dockerfile du projet Spring Boot. Ce service expose les ports nécessaires et définit les variables d'environnement pour la connexion à la base de données et autres configurations.
  - **Service Base de données :**
    Le conteneur MariaDB est configuré avec les volumes persistants pour assurer la sauvegarde des données, et les paramètres de configuration adaptés (utilisateur, mot de passe, nom de base, etc.).
- **Fichier prod.yaml :**
  Un fichier prod.yaml gère la configuration des services :
  - **Les mêmes services que dans de fichier dev.yaml**
  - **Service de tunnel :**

    Le conteneur cloudflared servira à ne pas exposer directement les ports du serveur de production, permettant de renforcer la sécurité du projet en intégrant le WAF de cloudflare et masquant l'ip publique du serveur.

Déploiement du client

1. Packaging natif de l'application JavaFX :

- **Création de packages natifs :**
  Utilisation de l'outil jpackage pour générer des installateurs natifs adaptés aux différents systèmes d'exploitation (Windows, macOS, Linux). Cela fournit une expérience utilisateur fluide lors de l'installation de l'application.
- **Mise à disposition :**
  Les installateurs sont hébergés sur un site officiel ou un repository interne, permettant aux utilisateurs de télécharger et d'installer facilement la dernière version.

Livrables :

- Document de conception :
    - Liste détaillée des fonctionnalités
    - Maquette des écrans
- Guide pour les utilisateurs :
    - Écrans et instructions
- Repository de code
    - Branche développement à jour
    - Branche main à jour
    - Branche release à jour

Logiciel existant d'inspiration : Telegram