RAPPORT SAÉ302

Groupe:**2**

Ronaldo **Nganga** **Adam**

OBJECTIF

Réaliser une application client/serveur (CHAT DE RÉSEAUX SOCIAL)utilisant les fonctions communicantes permettant de concevoir un protocole applicatif au-dessus de la pile de communication TCP/IP.

Nos avons utilise la langage de programmation Java et comme IDE Netbeans . La présence de certaines fonctions dans le langage Java simplifie la gestion des sockets et des flux de données, offrant ainsi un environnement favorable à la mise en œuvre de protocoles de communication avancés.

On a choisi le protocole TCP à cause de sa fiabilité inhérente et sa capacité à assurer un transfert de données robuste entre appareils distants. En tant que protocole orienté connexion, TCP crée des canaux stables entre le client et le serveur, garantissant l'intégrité des données échangées. Cette fonctionnalité est essentielle pour les applications où la perte ou la corruption de données est inacceptable.

Description

Cette application de réseau social permet aux utilisateurs de se connecter, d'échanger des messages. Les messages suivent le format "login,commande,parametre1,parametre2,...", avec le serveur indiquant "serveur". L'authentification se fait par login et mot de passe, assurant la sécurité. L'application prend en charge la communication en temps réel et stocke les messages pour les utilisateurs hors ligne. L'architecture client-serveur gère toutes les interactions, offrant une expérience interactive et sécurisée.

Fonctionnalités

Nous avons mis les fonctionnalités suivantes dans notre application :

- Interface Utilisateur
- Authentifications sécurise
- Messagerie instantanée
- Gestion des amis
- Stockage des Messages

Approche des Sockets pendant le Projet

L'un des piliers fondamentaux de notre recherche et développement a été la compréhension approfondie des sockets, étant donné leur importance dans la mise en œuvre d'une communication efficace entre le client et le serveur.

Phase de Recherche : Au cours de la phase initiale du projet, nous avons consacré un temps significatif à l'étude des concepts essentiels liés aux sockets. Cela incluait la compréhension des différents types de sockets, des protocoles de communication tels que TCP et UDP, ainsi que des défis et avantages associés à chaque approche.

Implémentation Pratique : L'application pratique de ces connaissances a impliqué la création d'un système de sockets robuste et efficace. Nous sommes allés au-delà des bases, explorant des nuances telles que la communication synchrone et asynchrone pour répondre aux exigences spécifiques de notre plateforme.

Défis Surmontés : Tout au long du processus, nous avons rencontré et surmonté des défis liés aux délais, à la manipulation des exceptions et à la scalabilité. Ces expériences ont contribué de manière significative à l'amélioration de la stabilité et des performances du système.

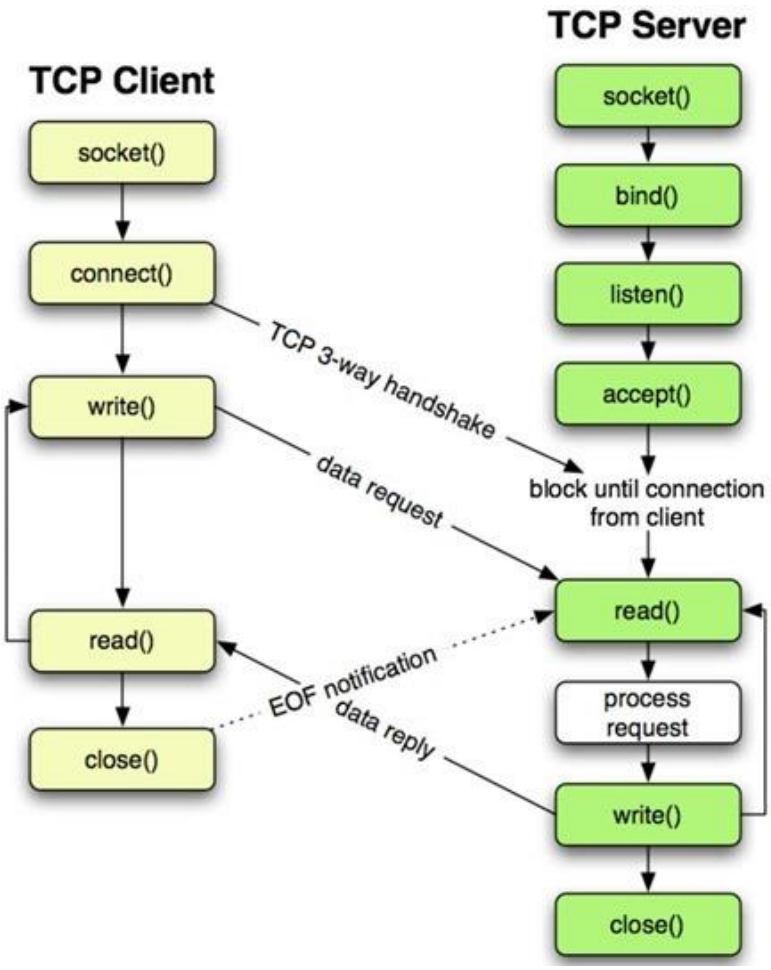
La compréhension approfondie des sockets a non seulement fondé la structure du projet, mais a également fourni une base solide pour la résolution efficace des problèmes et les améliorations continues tout au long du développement.

Qu'est-ce que sont les sockets ?

Actuellement, il existe plusieurs technologies permettant la communication entre les machines. L'une des technologies ou mécanismes les plus courants pour cette communication sont les sockets.

Un socket est un mécanisme de communication couramment utilisé pour implémenter un modèle client/serveur, permettant l'échange de messages entre les processus d'une machine ou d'une application serveur et d'une machine ou d'une application cliente.

Comme illustré dans l'image ci-dessous, ce modèle de communication implique un serveur et un client. Le serveur est démarré et attend les demandes des clients sur un port spécifique.



Ensuite, le client établit une connexion (en utilisant l'IP du serveur et le port), et si tout se passe comme prévu, le client et le serveur commencent à échanger des informations entre eux.

À la fin de l'échange de messages, le client et le serveur ferment la communication.

Flux de communication

Connexion: Lorsqu'un utilisateur tente de se connecter, le module de connexion envoie au serveur un message contenant le nom d'utilisateur et le mot de passe.

Le serveur traite les informations, authentifie l'utilisateur et répond au module Login en indiquant si l'authentification a réussi.

Enregistrer :Lors de l'enregistrement d'un nouvel utilisateur, le module Register envoie un message au serveur contenant les informations d'enregistrement (nom d'utilisateur et mot de passe).

Le serveur valide et stocke ces informations, confirmant ainsi le succès de l'enregistrement.

Scénarios de connexion et d'enregistrement des utilisateurs

Connexion réussie :

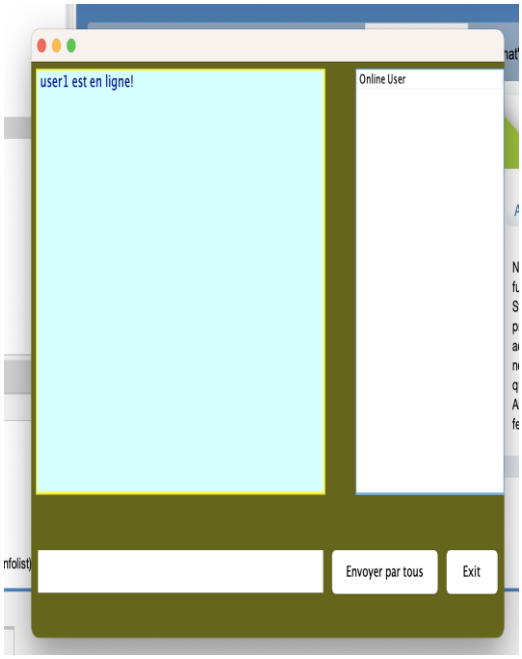
*Le client (module de connexion) envoie les informations d'identification au serveur.

*Le serveur authentifie les informations et répond par un message indiquant le succès.

*Le client reçoit le message de réussite et autorise l'accès au système.

Image.Login

image. Authentification réussie



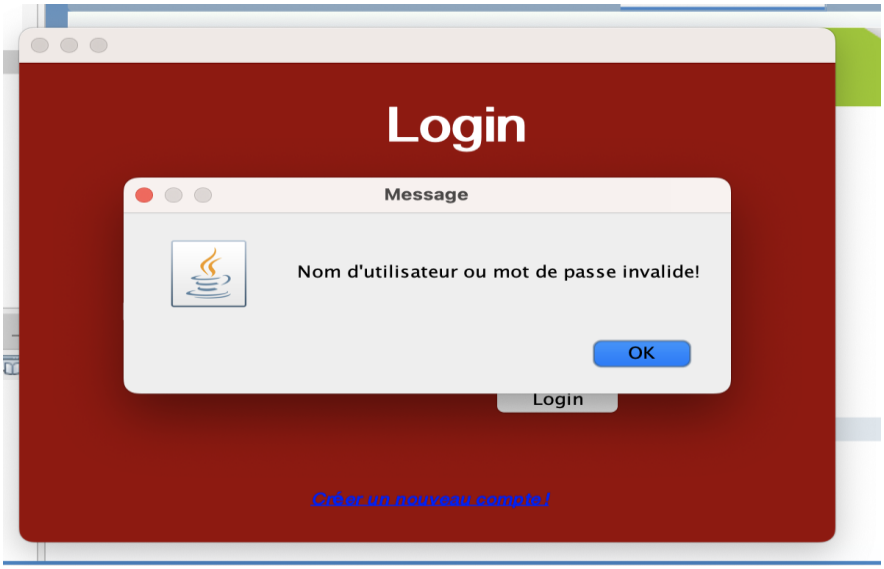
Connexion non réussie :

*Le client (module de connexion) envoie les informations d'identification au serveur.

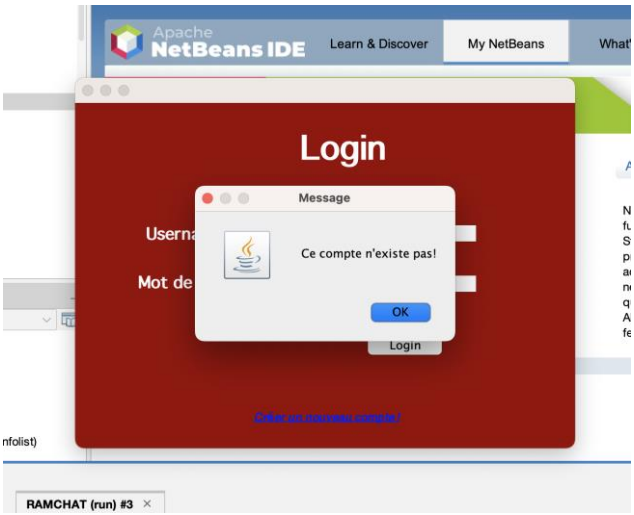
*Le serveur ne trouve aucune correspondance avec les données d'identification et répond par un message indiquant l'échec.

*Le client affiche un message d'erreur et demande de nouvelles données d'identification.

réponse du serveur : les informations d'identification ne sont pas correct, mais les utilisateurs existent



réponse du serveur : les utilisateurs n'existent pas sur le serveur



Enregistrement réussi :

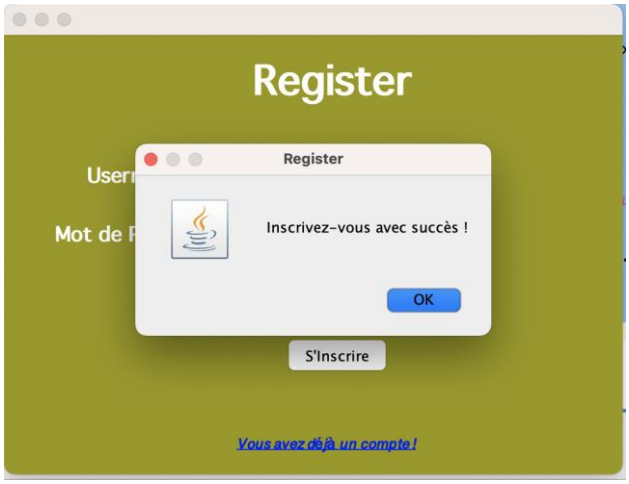
*Le client (module d'enregistrement) envoie les informations d'enregistrement au serveur.

*Le serveur valide et stocke les informations et répond par un message de réussite.

*Le client affiche le message de réussite et autorise la connexion avec les nouvelles données d'identification.

Écran d'enregistrement

réponse du serveur lorsque l'enregistrement est réussi



Mise en œuvre du stockage des utilisateurs sur le serveur

Le stockage des utilisateurs sur le serveur est un élément crucial du système, car il permet de suivre et de gérer les clients connectés. Dans ce rapport, nous nous concentrerons sur la classe ServerHandle, qui joue un rôle central dans cette fonctionnalité.

en dessous, nous avons mis une partie du code de cette classe:

```
public class ServerHandle {
    static final int PORT = 4445;
    public static List<AccountInfo> accountinfolist;

    public List<AccountInfo> getAccountinfolist() {
        return accountinfolist;
    }

    public void setAccountinfolist(List<AccountInfo> accountinfolist) {
        this.accountinfolist = accountinfolist;
    }
    public static void addUser(AccountInfo item){
        accountinfolist.add(item);
    }
    public static void removeUser(AccountInfo item){
        try{
            for(int i=0; i<= accountinfolist.size(); i++){
                if(accountinfolist.get(i).getUsername().equals(item.getUsername())){
                    accountinfolist.remove(i);
                    break;
                }
            }
        }catch(Exception ex){
            accountinfolist.clear();
        }

    }
```

La classe ServerHandle possède une liste statique appelée accountinfolist, qui est initialisée comme une liste vide. Cette liste stocke des objets de la classe AccountInfo, représentant des informations sur les utilisateurs connectés, y compris leurs noms d'utilisateur et les sockets associées.

La méthode addUser est chargée d'ajouter un utilisateur à la liste lorsqu'il se connecte au serveur. Elle reçoit un objet AccountInfo en paramètre et ajoute simplement cet objet à la liste.

La méthode removeUser est chargée de supprimer un utilisateur de la liste lorsqu'il se déconnecte du serveur. Elle recherche l'utilisateur dans la liste sur la base de son nom d'utilisateur et le supprime lorsqu'elle le trouve. Si une exception se produit au cours du processus, la liste est effacée pour garantir la cohérence.

Ces méthodes sont invoquées à des moments précis du cycle de vie du serveur, par exemple pendant le processus d'authentification (info) et lorsqu'un utilisateur décide de se déconnecter (Exit). Le contrôle actif de cette liste permet au serveur de gérer les utilisateurs connectés et de fournir des fonctions personnalisées telles que le chat de groupe, la messagerie privée et le contrôle de la présence en ligne.

En résumé, la classe ServerHandle est l'épine dorsale du système de stockage des utilisateurs. Elle fournit des méthodes cruciales pour ajouter et supprimer dynamiquement des utilisateurs au fur et à mesure qu'ils interagissent avec le serveur. Cela favorise une architecture flexible et efficace pour la gestion des utilisateurs dans l'environnement du serveur.

***Observation

Côté serveur (ServerHandle) :

Configuration du serveur :

Le serveur est démarré sur le port 4445.

Une liste statique accountinfolist est initialisée pour stocker les informations sur les utilisateurs connectés.

```
static final int PORT = 4445;
public static List<AccountInfo> accountinfolist;
```

Initialisation du serveur :

Le serveur attend les connexions et crée un nouveau fil d'exécution (ServerThread) pour chaque client connecté.

Le fil d'exécution du serveur (ServerHandle) gère l'ajout et la suppression d'utilisateurs dans la liste.

```
public static void main(String args[]) {
    accountinfolist = new ArrayList<>();
    Socket s=null;
    ServerSocket ss2=null;
    Transfer transfer = new Transfer();
    Thread tTransfer = new Thread(transfer);
    tTransfer.setName("tTransferfile");
    tTransfer.start();
    System.out.println("Serveur en écoute sur port "+PORT+"......");
    try{
        ss2 = new ServerSocket(PORT);

    }
    catch(IOException e){
        System.out.println("Erreur du serveur");
    }

    while(true){
        try{
            synchronized(ss2) {
                s= ss2.accept();
                System.out.println("connexion établie : Adresse "+ s);
                ServerThread st=new ServerThread(s,ServerHandle.accountinfolist);
                st.start();

            }
        }

    catch(Exception e){
        //System.out.println("Connection Error");

    }
    }
}
```

Communication du client avec le serveur :

La classe ClientHandle gère la communication avec le client, en envoyant les informations de connexion au serveur.

```
    public static void InitClient(String username){

        try {
            //System.out.println("_InitClient");
            //System.out.println("0:"+username);
            user = username;
            address=InetAddress.getLocalHost();
            System.out.println("address: "+address);
            socket=new Socket(address, 4445); // You can use static final constant POR
            br= new BufferedReader(new InputStreamReader(System.in));
            is=new BufferedReader(new InputStreamReader(socket.getInputStream()));
            os= new PrintWriter(socket.getOutputStream());
            //System.out.println("1:"+username);
            sendInfo(username);
            //System.out.println("2:"+username);
        }
        catch (IOException e){
            e.printStackTrace();
            System.err.print("IO Exception");
        }
    }
```

Réponse du serveur :

Après avoir envoyé les informations de connexion, le client attend une réponse du serveur (succès ou échec). Code pour lire la réponse du serveur (la Classe ReadClient).

```
 */
public class ReadClient extends Thread{
    private static Socket socket;
    public static String mess;
    public ReadClient(Socket socket) {
        this.socket = socket;
    }

    public static String getMess() {
        return mess;
    }

    public static void setMess(String mess) {
        ReadClient.mess = mess;
    }
```

Rétroaction de l'utilisateur :

En fonction de la réponse du serveur, le client affiche des messages de réussite ou d'échec.

```
private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String user = txtUsername.getText();
       String pass = txtPassword.getText();
       int stt = 0;
       List<String> listaccount = ReadWriteAccount.Read();
       for(String item:listaccount){
           String usertmp = item.split(":")[0];
           String passtmp = item.split(":")[1];
           if(user.equals(usertmp))
           {
               stt = 1;
               if(pass.equals(passtmp)){
                   this.setVisible(false);
                   //ChatFrame chatf = new ChatPublic(user);
                   //chatf.setVisible(true);

                   Thread th = new Thread(){
                       public void run(){
                           ChatPublic chatf = new ChatPublic(user);
                           //chatf.setUsername(user);
                           chatf.setVisible(true);
                       }
                   };
                   th.start();

               }else{
                   JOptionPane.showMessageDialog(null,"Nom d'utilisateur ou mot de passe invalide!");
               }
           }
       }
       if(stt==0){
           JOptionPane.showMessageDialog(null,"Ce compte n'existe pas!");
       }

}
```

Cette analyse met en évidence la manière dont le client et le serveur interagissent au cours du processus de connexion, la communication étant gérée par les méthodes de la classe ClientHandle.

Chat public et privé

Envoi de messages au groupe :

Lorsque l'utilisateur tape un message et appuie sur la touche "Entrée" ou clique sur le bouton "Envoyer par tous", le message est envoyé au groupe via la méthode btnSendActionPerformed.

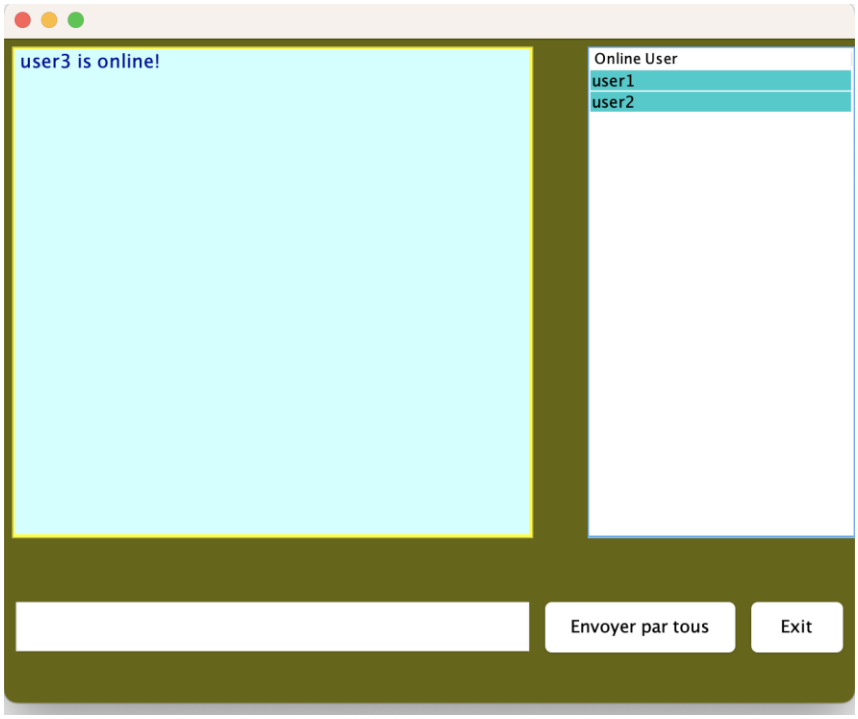
```
private void btnSendActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String type = "ChatGroup";
    String mess = username+":"+txtMess.getText()+":"+type;
    client.WriteClient(mess);
    txtMess.setText("");
}
```

Gestion des utilisateurs en ligne :

La liste des utilisateurs en ligne est mise à jour via la méthode ReadUpdate lorsque le serveur reçoit des messages entrants/sortants d'utilisateurs en ligne.

```
public void ReadUpdate(Socket socket)
{
    BufferedReader is=null;
    String line=null;
    try {
        is=new BufferedReader(new InputStreamReader(socket.getInputStream()));
    }
    catch (IOException e){
        e.printStackTrace();
        System.err.print("IO Exception");
    }
    try{
        while(true){
            line = is.readLine();
            System.out.println("line: "+line);
            String rq = line.split(":")[2];
            String messfromserver = line.split(":")[1];
            String userfromserver = line.split(":")[0];
            String messappendtochatgroup = userfromserver+":"+messfromserver+"\n";
            if(rq.equals("ChatGroup")){
                txtareaChatGroup.append(messappendtochatgroup);
            }
            if (rq.equals("UserOnline") ||rq.equals("UserExit") )
            {
                txtareaChatGroup.append(messfromserver+"\n");
                String ListOnlineUser = line.split(";")[1];
                System.out.println(ListOnlineUser);
                List<String> OnlineUsers = new ArrayList<>();
                String[]datatmp = ListOnlineUser.split(":");
                String[] data = datatmp[1].split(",");
                for(String item:data){
                    if(!item.equals(username)){
                        OnlineUsers.add(item);
                    }
                }
```

ChatPublic



Chat privé :

Si un utilisateur double-clique sur un utilisateur en ligne, une discussion privée est lancée.

```
private void jtbOnlineUserMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:

    if(evt.getClickCount()==2){
        int i = jtbOnlineUser.getSelectedRow();
        TableModel model = jtbOnlineUser.getModel();
        String desUser = model.getValueAt(i, 0).toString();
        String source = username;
        Thread th = new Thread(){
            public void run(){
                ChatPrivate chatPrivatef = new ChatPrivate(source,desUser,client,"");
                ChatPrivateClass chatPrivate = new ChatPrivateClass(chatPrivatef,source,desUser);
                listChatPrivate.add(chatPrivate);
                chatPrivate.getChatprivate().setVisible(true);
            }
        };
        th.setName(source+"-"+desUser);
        th.start();
    }


}
```

Réception et envoi de fichiers :

Il existe une logique de réception et d'envoi de fichiers, utilisant la classe ClientTransferFile et des méthodes telles que Receivefile et SendFile.

la mise en œuvre de l'envoi de fichiers a été motivée par notre curiosité et notre désir d'apprendre, sans qu'il soit nécessaire d'entrer dans les détails techniques spécifiques de la documentation. Cela devrait donner l'idée que l'exploration de ces fonctionnalités a été un choix motivé par notre curiosité et notre désir d'affiner nos compétences.

Algorithme de relation entre amis

Côté serveur, la variable mesAmis est une liste qui stocke les relations d'amitié. Chaque relation d'amitié est représentée comme une concaténation des identifiants des deux clients impliqués. Par exemple, la relation entre les clients 3 et 5 serait représentée par "35" ou "53".

Ceci est fait pour permettre une vérification bidirectionnelle, quel que soit l'ordre dans lequel les clients sont devenus amis.

Malheureusement, nous n'avons pas eu le temps d'implémenter cette logique, mais nous pouvons garantir que cette solution fonctionne, car nous l'avons déjà testée dans une autre application.

Conclusion

En conclusion, ce projet a non seulement fourni un environnement propice à l'apprentissage et au développement de compétences techniques, mais a également renforcé l'importance de la curiosité et de la collaboration. En explorant de nouvelles fonctionnalités, telles que l'envoi de fichiers, nous avons élargi notre compréhension et relevé des défis qui ont renforcé notre équipe. Chaque membre a contribué de manière significative, et les expériences partagées ont été essentielles à notre croissance collective. Cette aventure a non seulement amélioré nos compétences techniques, mais a également souligné l'importance de la curiosité en tant que catalyseur de l'apprentissage continu. En fin de compte, nous célébrons non seulement le succès du projet, mais aussi la richesse de l'expérience que nous avons construite ensemble.