Dokumentation

Matthias Vonend Jan Grübener Patrick Mischka
Michael Angermeier Troy Keßler Aaron Schweig

25. März 2020

Inhaltsverzeichnis

1 Basisanforderungen

Test

Die Anwendung soll //TODO

Aus diesem Grund wird die Anwendung von mehreren Gleichwertigen Servern bedient. Jeder dieser sog. Nodes hat dabei eine vollständige Kopie sämtlicher dem System bekannten Daten. Eine Veränderung des Datenbestandes muss dem entsprechend an alle anderen Nodes weiter gegeben.

Jeder Client kann sich zu einer Node verbinden. Möchte dieser eine Nachricht an einen weiteren Client senden, schickt er diese an seine verbundene Node und überlässt die Zustellung dieser. Da alle Nachrichten aus Persitenzgründen an alle Nodes verteilt werden müssen, brauchen die Nodes keine Information über die Clients anderer Nodes. Im Falle einer solchen Anforderung (z. B. Abfrage ob ein anderer Nutzer aktiv ist) könnte ein Protokoll ähnlich zu Routing-Tabellen implementiert werden. Empfängt eine Node eine Nachricht, egal ob von einem Client oder von einer anderen Node, überprüft diese ob die Nachricht für einen ihr bekannten Client bestimmt war und sendet diese gegebenenfalls an diesen.

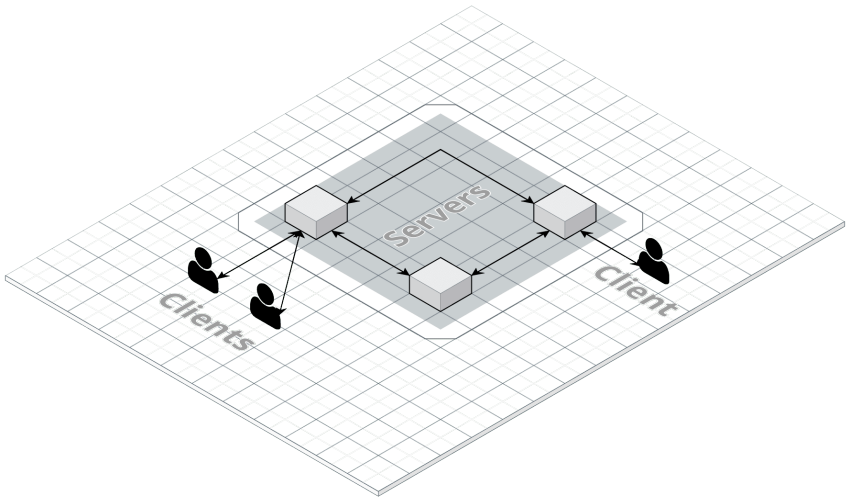

Abbildung 1: Architecture

2 Erweiterungen

2.1 Grafische Oberfläche bei den Nutzern

2.2 Verwendung von Emojis

2.3 Mehrere Chatverläufe pro Nutzer

2.4 Persistentes Speichern der Chatverläufe

Sämtliche der Node bekannten Informationen (Nutzer, Chats, Nachrichten) werden in einem Warehouse verwaltet. Um diese Informationen zwischen Node-Neustarts zu persistieren muss demnach das Warehouse als Datei gespeichert werden und bei Node-Start wieder geladen werden. Existiert noch kein Speicherstand wird die Node mit einem leeren Warehouse initialisiert. Während die Node ausgefallen ist, können andere Nodes gleichzeitig neue Informationen erhalten haben. Sobald die Node wieder verfügbar ist müssen andere Nodes dies bemerken und die ausgefallene Node mit Informationen versorgen. Um den Ausfall festzustellen ist ein Heart-Beat vorgesehen. Dieser pingt jede Sekunde alle benachbarten Nodes an um sicherzustellen, dass die Verbindung noch existiert. Sollte eine Unterbrechung festgestellt, versucht die Node die Verbindung wieder aufzubauen. Gelingt dies, so wird das Warehouse übersendet. Jede Node prüft ob sie alle Informationen des empfangenen Warehouses bereits besitzt und fügt neue Informationen hinzu. Sofern die Node neue Informationen erhalten haben, broadcastet diese ihren neuen Stand an alle benachbarten Nodes um diese auch auf den neuesten Stand zu bringen. Der Speichervorgang kann je nach System und Warehousegröße längere Zeiten in anspruch nehmen. In dieser Zeit können in der Regel keine weiteren Anfragen verarbeitet werden. Um diese Zeit zu minimieren kümmert sich ein eigener Thread um die Speicherung und speichert das Warehouse in Intervallen. Zu beachten ist der Fall, dass eine Node zusammenbricht während der Speichervorgang in Arbeit ist. In diesem Fall würde die node

sämtliche Informationen verlieren, da die Speicherdatei korrupt ist. Um dies zu verhindern wird der Speicherstand zunächst in eine Tempdatei geschrieben und nach erfolgreicher speicherung an den Zielort verschoben.

2.5 Verschlüsselte serverseitige Speicherung der Chats

2.6 Gruppenchats

2.7 Verschlüsselte Übertragung der Chat-Nachrichten

3 Anhang

```
  @Override
  public void run() {
    Thread.currentThread().setName("Connection Handler");
    while (!this.context.isCloseRequested().get()) {
      System.out.println("Handling");
      try {
        var object = inputStream.readObject();
        var packet = (Packet) object;
        this.handlePacket(packet);
      } catch (IOException | ClassNotFoundException e) {
        break;
      }
    }
    try {
      this.client.close();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }

  private void handlePacket(final Packet packet) throws IOException
     {
    for (var listener : this.context.getListeners()) {
      try {
        var methods = listener.getClass().getDeclaredMethods();
        for (var method : methods) {
          if (method.getName().equals("next") && !method.
    isSynthetic()) {
            var packetType = method.getParameters()[0];
            if (packetType.getType().isAssignableFrom(packet.
    getClass())) {
              var retu = (Packet) method.invoke(listener, packet,
    this.context, this);
              this.pushTo(retu);
            }
          }
        }

      } catch (SecurityException | IllegalArgumentException |
    IllegalAccessException
          | InvocationTargetException e) {
        e.printStackTrace();
      }
```

../src/main/java/vs/chat/server/ConnectionHandler.java

Test