

# Programm-Doku

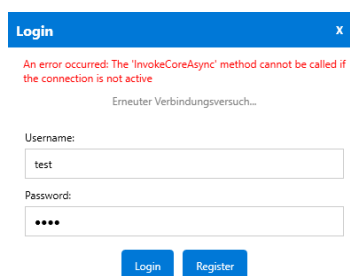
von Tom und Fabian

Dieses Projekt implementiert ein verteiltes Chat- und Dateimanagementsystem mit einer Client-Server Architektur. Das System ermöglicht es den Benutzern, sowohl in privaten Chats als auch in Gruppenchats zu kommunizieren. Die Benutzer können auch Dateien hochladen, herunterladen und versionieren oder auch Plugins laden.

## Private Chat / Gruppen Chat

Das Projekt ist in verschiedene Projekte unterteilt:

- Server
  - Setzt das Backend mit ASP.NET CORE und SignalR für die Echtzeitkommunikation um.
- Client
  - Der Client ist eine WPF Applikation.
- Shared
  - Enthält Modelle und Interfaces, die vom Client, Server oder einem Plugin benötigt werden.
- ModerationPlugin
  - Ist ein Plugin, das verschiedene Chatbefehle wie z.B. /time ermöglicht.
- WhiteboardPlugin
  - Ist ein Plugin, das es Benutzern ermöglicht, ein Whiteboard zu öffnen und mit anderen Benutzern zu zeichnen.

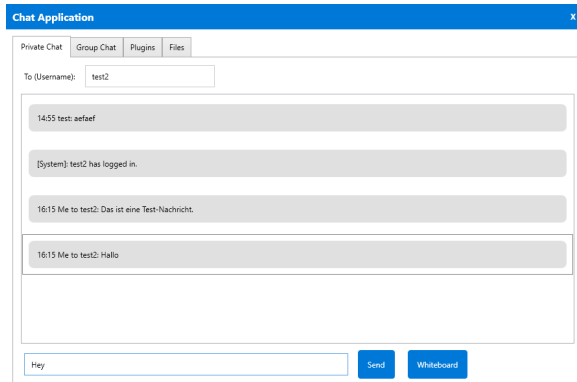


Nach dem Start des Programms erscheint ein Login-Fenster, in dem sich der Benutzer entweder einloggen oder registrieren kann. Das Passwort wird dabei mittels SHA256-Hash in der Datenbank gespeichert und beim Login zur Validierung herangezogen.

Sollte kein Server online sein oder die Verbindung nicht funktionieren, wird dem Benutzer eine Fehlermeldung zusammen mit einem Countdown angezeigt. Alle 10 Sekunden wird automatisch ein neuer Verbindungsversuch gestartet.

Nachdem der Benutzer sich erfolgreich eingeloggt oder registriert hat, startet die eigentliche Anwendung. In dieser kann er zwischen den Bereichen Private Chats, Gruppen Chats, Plugins und Dateiverwaltung wechseln.

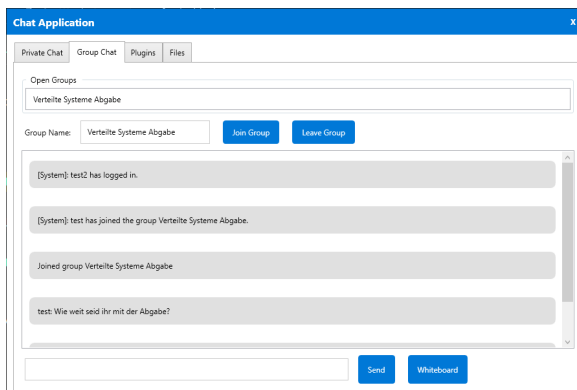
## Private Chat / Gruppen Chat



Im Private Chat kann ein Benutzer direkt mit einem anderen Benutzer kommunizieren. Sobald im entsprechenden Textfeld ein Zielbenutzer eingegeben wird, wird über das "TextChanged"-Event die Methode zum Laden der Chat-Historie aufgerufen. Konkret wird, wenn der Benutzer im Feld „To (Username)“ einen Zielbenutzernamen eingibt, das Event ausgelöst, das die Methode LoadPrivateChatHistory aufruft. Diese Methode sendet über die SignalR-Verbindung einen Aufruf an die Server-Methode GetPrivateChatHistory. Falls der Zielbenutzer

existiert, werden die bisherigen Nachrichten geladen und in chronologischer Reihenfolge angezeigt, sodass der Benutzer alle alten Nachrichten inklusive Zeitstempel sieht.

Um eine neue Nachricht zu senden, gibt der Benutzer einen Text in das Eingabefeld ein und klickt entweder auf den „Send“-Button oder drückt die Enter-Taste. Beim Absenden wird die Nachricht an die Methode ProcessMessageThroughPlugins weitergeleitet, wodurch Plugins, wie beispielsweise das ModerationPlugin, die Nachricht verarbeiten können, bevor sie an den Server gesendet wird.



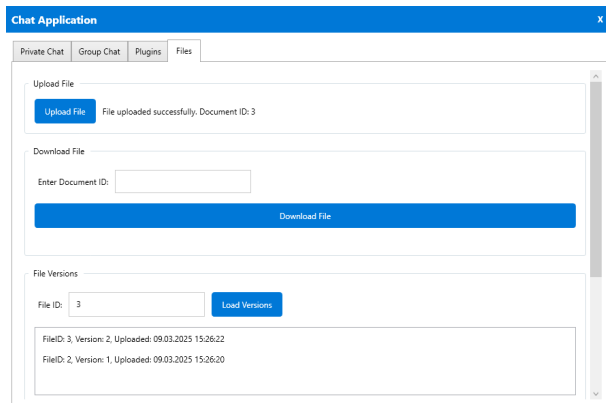
Der Gruppen-Chat funktioniert ähnlich wie der Private Chat, bietet jedoch die Möglichkeit, mit mehreren Benutzern gleichzeitig zu kommunizieren. In diesem Bereich sieht der Benutzer alle derzeit offenen Gruppen in einer übersichtlichen Liste. Eine Gruppe kann der Benutzer entweder beitreten, indem er den Gruppennamen in ein entsprechendes Feld eingibt, oder indem er einen Doppelklick auf den gewünschten Gruppennamen in der Liste ausführt und anschließend auf den „Join Group“-Button klickt. Über die entsprechenden Buttons kann er zudem Gruppen erstellen oder

verlassen.

Sobald eine Nachricht im Gruppen-Chat eingegeben und gesendet wird, wird sie – ebenfalls optional durch Plugins bearbeitet – an die Server-Methode SendGroupMessage übermittelt. Der Server leitet die Nachricht dann an alle Mitglieder der Gruppe weiter, sodass alle Teilnehmer in Echtzeit die neuen Nachrichten empfangen.

Das System meldet in den Chats auch diverse Informationen, wie zum Beispiel, wenn ein Benutzer einer Gruppe beitrifft oder diese verlässt, oder wenn sich jemand im Programm ein- oder ausloggt.

## Dateiverwaltung

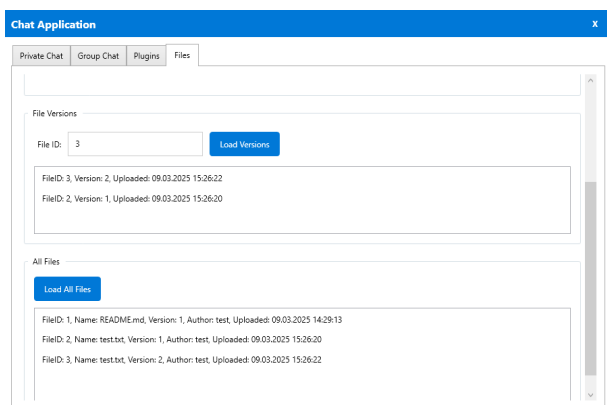


Der Benutzer kann im Tab "Files" Dateien hochladen. Beim Klick auf den „Upload File“-Button öffnet sich ein Dateiauswahldialog, in dem der Benutzer die gewünschte Datei auswählt. Das System ermittelt daraufhin den Dateipfad, liest den Dateiinhalt als Byte-Array ein und wandelt diesen in einen Base64-kodierten String um. Zusammen mit dem Originaldateinamen, dem Benutzernamen als Autor und optionalen Metadaten wird der Inhalt über die SignalR-Verbindung an die Server-Methode UploadDocument übermittelt. Auf der Serverseite wird der Dateiinhalt zunächst in dem vordefinierten

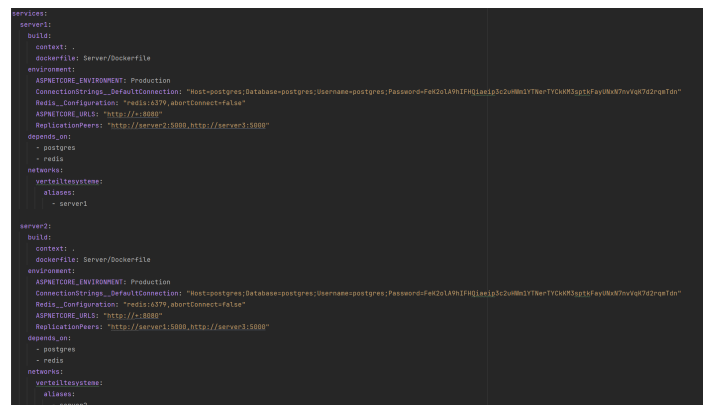
Verzeichnis abgelegt. Dabei wird ein eindeutiger Dateiname erzeugt, indem vor den Originalnamen eine GUID gestellt wird, um Namenskollisionen zu vermeiden. Anschließend wird in der Datenbanktabelle documents ein Eintrag erstellt, der neben dem Dateinamen auch Angaben zum Autor, zum Upload-Zeitpunkt, zum Dateipfad, zu den Metadaten und zur Versionsnummer speichert. Falls bereits ein Eintrag mit demselben Dateinamen und Autor existiert, wird die Versionsnummer automatisch um eins erhöht.

Beim Download gibt der Benutzer im entsprechenden Bereich eine Dokument-ID ein, wodurch die Server-Methode DownloadDocument aufgerufen wird. Zunächst wird versucht, die Datei anhand des in der Datenbank gespeicherten Pfads lokal zu finden. Ist die Datei nicht vorhanden, wird eine in der Konfiguration hinterlegte Liste von Replikations-Peers verwendet, an die eine HTTP-Anfrage an den Endpunkt /replicate mit der Dokument-ID gesendet wird. Wird die Datei von einem Peer gefunden, wird der Base64-kodierte Inhalt zurückgegeben, lokal gespeichert und dem Benutzer zum Download angeboten.

Außerdem kann der Benutzer den Versionsverlauf einer Datei einsehen. Mithilfe der Methode GetDocumentVersionsById werden alle Versionen einer Datei, basierend auf Dateiname und Autor, in absteigender Reihenfolge angezeigt, sodass bei Bedarf auch auf ältere Versionen zugegriffen werden kann. Zusätzlich steht die Funktion LoadAllFiles zur Verfügung, mit der der Benutzer einen vollständigen Überblick über alle im System gespeicherten Dateien erhält. In dieser Ansicht werden alle relevanten Metainformationen wie Datei-ID, Dateiname, Version, Autor und Upload-Zeitpunkt übersichtlich dargestellt. Dadurch kann der Benutzer nicht nur die einzelnen Versionen einer Datei nachvollziehen, sondern auch schnell erkennen, welche Dateien aktuell vorhanden sind und welche zusätzlichen Informationen zu den Dokumenten gespeichert wurden.



File-Versioning und Ansicht aller Dateien



docker-compose.yml mit den ReplicationPeers

## Plugins

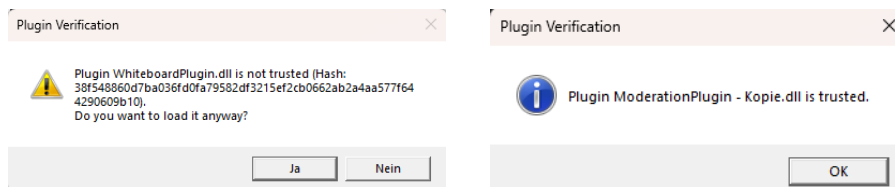
Im Tab „Plugins“ werden alle im Plugin-Verzeichnis hinterlegten Erweiterungen geladen. Der PluginLoader sucht beim Start nach vorhandenen Plugin-DLLs und lädt diese in einen separaten AssemblyLoadContext, um eine Isolation vom Hauptsystem zu gewährleisten. Dadurch wird vermieden, dass Fehler oder unerwünschte Seiteneffekte des Plugin-Codes die restliche Anwendung beeinträchtigen.

```
private bool VerifyPlugin(string pluginFile)
{
    _logger.LogInformation("Verifying plugin: {File}", pluginFile);
    try
    {
        using (var stream = File.OpenRead(pluginFile))
        using (var sha256 = SHA256.Create())
        {
            var hashBytes = sha256.ComputeHash(stream);
            var hashString = BitConverter.ToString(hashBytes).Replace("-", "").ToLowerInvariant();

            var allowedHashes = new HashSet<string>
            {
                "F972afafe05a130aef39caef54ab87b77626d8a0cdea82cc5a51693b" // ModerationPlugin
            };

            if (allowedHashes.Contains(hashString))
            {
                _logger.LogInformation("Plugin {File} is trusted. Hash: {Hash}", pluginFile, hashString);
                MessageBox.Show($"Plugin {File} is trusted.", "Plugin Verification",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
                return true;
            }
        }
    }
}
```

Bevor ein Plugin geladen wird, erfolgt eine Sicherheitsprüfung. Dabei wird der SHA256-Hash der Plugin-Datei berechnet und mit einem vordefinierten Satz vertrauenswürdiger Hash-Werte verglichen. Stimmt der ermittelte Hash überein, gilt das Plugin als vertrauenswürdig und wird automatisch geladen. Sollte der Hash jedoch nicht den erwarteten Werten entsprechen, wird dem Benutzer über eine Warnmeldung die Möglichkeit gegeben, selbst zu entscheiden, ob er das Plugin trotzdem laden möchte.



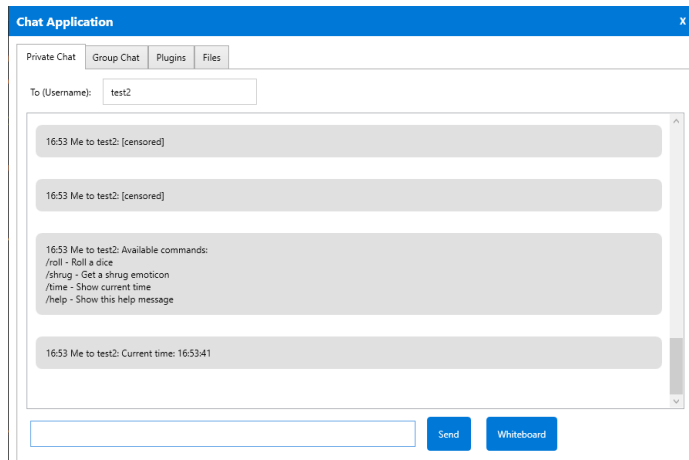
Ein weiterer Aspekt des Plugin-Konzepts ist das dynamische Nachladen zur Laufzeit. Wird beispielsweise ein Videocall initiiert, und besitzt Client B das dafür benötigte Plugin noch nicht, erfolgt folgender Ablauf:

Zunächst wird Client B über ein entsprechendes Event (z. B. `ReceiveWhiteboardPluginRequest`) informiert und erhält die Frage, ob das fehlende Plugin automatisch nachgeladen werden soll. Akzeptiert der Benutzer diese Anfrage, ruft Client B über die bestehende SignalR-Verbindung die Methode `RequestPluginFile` auf. Der Client, der das Plugin bereits besitzt, reagiert darauf, indem er die entsprechende Plugin-Datei (z. B. `WhiteboardPlugin.dll`) aus seinem Plugin-Verzeichnis einliest, in einen Base64-kodierten String umwandelt und diesen an Client B übermittelt (mittels der Methode `SendPluginFile`). Auf der Empfängerseite dekodiert Client B den Base64-String, speichert die Plugin-Datei im eigenen Plugin-Verzeichnis und informiert den Benutzer über die erfolgreiche Installation. Dadurch kann das Plugin ohne Neustart der Software dynamisch nachgeladen und direkt verwendet werden. Er muss dann nur auf Load Plugins im Plugin Tab gehen und kann dann in den Chat gehen, den Usernamen eingeben, auf den Whiteboard Button klicken und schon hat man ein gemeinsames Whiteboard mit zwei Usern.

## ModerationPlugin

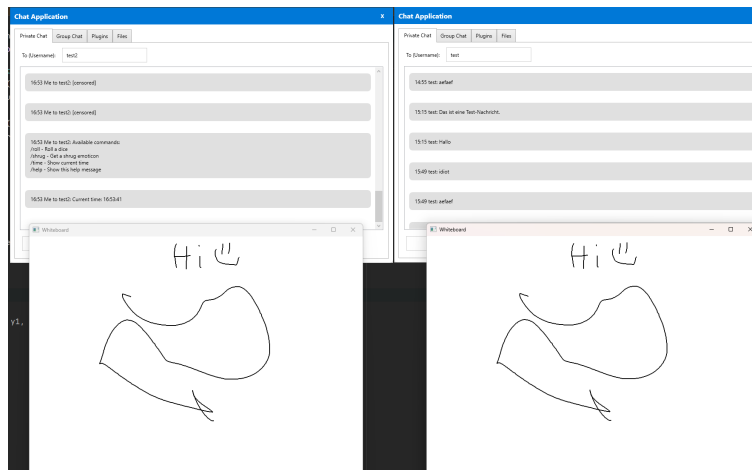
Dieses Plugin erweitert den Chat um Moderationsfunktionen. Es implementiert die Schnittstelle `IPlugin` und arbeitet im Hintergrund, ohne eine eigene Benutzeroberfläche bereitzustellen. In der Methode `ProcessMessage` wird zunächst geprüft, ob die eingegebene Nachricht als Befehl interpretiert werden soll – erkennbar daran, dass sie mit einem Schrägstrich (/) beginnt. Unterstützt werden Befehle wie `"/roll"` zum Würfeln, `"/shrug"` für einen Emoji, `"/time"` zur Anzeige der aktuellen Uhrzeit und `"/help"` zur Ausgabe einer Befehlsübersicht. Handelt es sich nicht um einen Befehl, erfolgt eine automatische Moderation, bei der mithilfe regulärer

Ausdrücke bestimmte Beleidigungswörter (zum Beispiel "idiot", "stupid", "dumb", ...) durch den Platzhalter "[censored]" ersetzt werden.



### WhiteboardPlugin

Dieses Plugin ermöglicht es den Benutzern, ein Whiteboard zu öffnen und gemeinsam in Echtzeit zu zeichnen. Beim Aufruf des Plugins wird ein neues Fenster geöffnet, in dem ein Whiteboard integriert ist. Der Benutzer kann mit der Maus zeichnen, wobei die gezeichneten Linien sofort lokal dargestellt und über die SignalR-Verbindung an andere Teilnehmer übertragen werden. Das Plugin unterstützt dabei sowohl den privaten Modus als auch den Gruppenmodus.



## Server-Infrastruktur und Robustheit

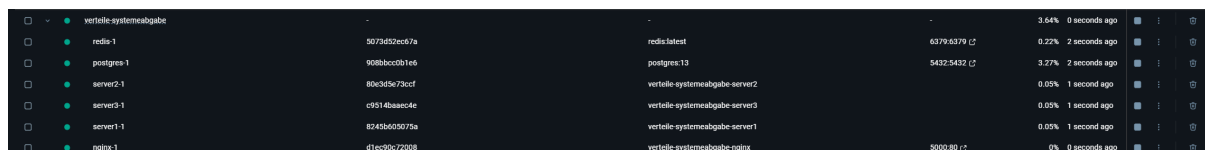
Unser System ist so gemacht, dass der Ausfall einzelner Knoten den laufenden Betrieb nicht beeinträchtigt. Im Kern besteht die Server-Infrastruktur aus drei identischen Servern, die das ASP.NET Core-Backend implementieren und über SignalR eine Echtzeitkommunikation mit den Clients ermöglichen. Alle drei Server greifen auf dieselbe PostgreSQL-Datenbank zu, in der Benutzerdaten, Chatnachrichten und Dateimetadaten persistent gespeichert werden.

Um die Kommunikation zwischen den Servern zu erleichtern, ist Redis als Zwischenspeicher integriert. Redis fungiert als Message Broker. Wenn ein Server eine Nachricht oder einen Befehl sendet, sorgt Redis dafür, dass diese Information an alle anderen Server weitergeleitet wird. Dadurch wird gewährleistet, dass auch bei einem Ausfall einer einzelnen Instanz alle übrigen Server synchron bleiben und die Echtzeitkommunikation ungestört fortgesetzt wird.

Als zentraler Zugangspunkt für die Clients dient ein Nginx-Loadbalancer, der in einem eigenen Container betrieben wird. Die Nginx-Konfiguration definiert einen Upstream-Bereich, in dem alle drei Server aufgeführt sind. Dabei wird der eingehende Traffic, beispielsweise für den Endpunkt /chatHub, gleichmäßig auf die verfügbaren Server verteilt. Sollte ein Server nicht mehr erreichbar sein, übernimmt Nginx automatisch den gesamten Datenverkehr und leitet ihn an die verbleibenden, funktionierenden Instanzen weiter.

Die gesamte Infrastruktur wird über Docker Compose verwaltet. In der docker-compose.yml-Datei sind sämtliche Container definiert

- Server (server1, server2, server3): Jeder Server wird aus demselben Dockerfile gebaut, wobei wichtige Umgebungsvariablen wie der Connection String zur PostgreSQL-Datenbank, die Redis-Konfiguration (z. B. "redis:6379,abortConnect=false") und eine Liste von Replikations-Peers gesetzt werden. Diese Peers ermöglicht es dem FileManagementService, bei fehlenden Dateien diese von anderen Knoten zu beziehen.
- PostgreSQL: Die Datenbank läuft in einem eigenen Container, nutzt ein persistentes Volume (postgres-data) und wird durch regelmäßige Healthchecks überwacht, sodass ihre Verfügbarkeit stets sichergestellt ist.
- Redis: Redis stellt einen zentralen Message Broker dar, der in einem eigenen Container betrieben wird. Er sorgt dafür, dass alle Nachrichten, die über SignalR versendet werden, schnell und zuverlässig zwischen den Servern verteilt werden.
- Nginx: Der Nginx-Container fungiert als Loadbalancer. Die zugehörige Konfiguration (nginx.conf) legt fest, dass eingehende Anfragen an den Endpunkt /chatHub über den definierten Upstream-Bereich an die drei Server weitergeleitet werden. Dabei wird sichergestellt, dass der Traffic auch bei Ausfall einer Instanz automatisch an die anderen Server verteilt wird.



|            |              |                                |               |       |               |   |   |   |
|------------|--------------|--------------------------------|---------------|-------|---------------|---|---|---|
| redis-1    | 5073d52ec67a | redis:latest                   | 6379.6379 (C) | 3.64% | 0 seconds ago | ■ | ! | ⊗ |
| postgres-1 | 908bacc0b1e6 | postgres:13                    | 5432:5432 (C) | 0.22% | 2 seconds ago | ■ | ! | ⊗ |
| server2-1  | 80e3d5e73ccf | verteilt-systemeabgabe-server2 |               | 3.27% | 2 seconds ago | ■ | ! | ⊗ |
| server3-1  | c8514baacc4e | verteilt-systemeabgabe-server3 |               | 0.05% | 1 second ago  | ■ | ! | ⊗ |
| server1-1  | 8245d653075a | verteilt-systemeabgabe-server1 |               | 0.05% | 1 second ago  | ■ | ! | ⊗ |
| nginx-1    | d1ec90c72008 | verteilt-systemeabgabe-nginx   | 5000.80 (C)   | 0%    | 0 seconds ago | ■ | ! | ⊗ |

Alle diese Container sind in einem gemeinsamen Netzwerk (verteilt-systeme) miteinander verbunden. Das System bleibt also beständig, auch wenn einer oder zwei Knoten ausfallen.

## Logging

Überall im System (Client, Server, ...) ist Logging eingebaut, damit Aktionen, Ereignisse aber auch Fehler leicht zu überwachen sind. In den Server-Komponenten, wie beispielsweise in der Authentifizierung, im FileManagementService oder in den Chat-Hubs, werden Logeinträge erstellt. So wird beispielsweise beim Login-Versuch eines Benutzers, beim erfolgreichen Zugriff auf die Datenbank oder beim Senden einer Nachricht detailliert protokolliert. Auch Fehler oder ungewöhnliche Zustände werden mit Logmeldungen erfasst, sodass bei Problemen eine schnelle Diagnose möglich ist.

```
2025-03-09 17:00:21 server1-1 | info: Server.Hubs.ChatHub[0]
2025-03-09 17:00:21 server1-1 |   ChatHub initialized.
2025-03-09 17:00:21 server1-1 | info: Server.Hubs.ChatHub[0]
2025-03-09 17:00:21 server1-1 |   Sending whiteboard line to group test2: [428, 4] to [430, 3]
2025-03-09 17:00:33 server1-1 | info: Server.Services.FileManagementService[0]
2025-03-09 17:00:33 server1-1 |   Using existing file storage directory at /app/UploadedFiles
2025-03-09 17:00:33 server1-1 | info: Server.Hubs.ChatHub[0]
2025-03-09 17:00:33 server1-1 |   ChatHub initialized.
2025-03-09 17:00:38 server1-1 | info: Server.Services.FileManagementService[0]
2025-03-09 17:00:38 server1-1 |   Using existing file storage directory at /app/UploadedFiles
2025-03-09 17:00:38 server1-1 | info: Server.Hubs.ChatHub[0]
2025-03-09 17:00:38 server1-1 |   ChatHub initialized.
2025-03-09 17:00:38 server1-1 | info: Server.Hubs.ChatHub[0]
2025-03-09 17:00:38 server1-1 |   Login attempt for user: test2
2025-03-09 17:00:38 server1-1 | info: Server.Services.AuthService[0]
2025-03-09 17:00:38 server1-1 |   Login attempt for user: test2
2025-03-09 17:00:38 server1-1 | info: Server.Services.AuthService[0]
2025-03-09 17:00:38 server1-1 |   Login for user test2 successful
2025-03-09 17:00:38 server1-1 | info: Server.Hubs.ChatHub[0]
2025-03-09 17:00:38 server1-1 |   User test2 logged in successfully. Connection ID fcvV4ZZxNH7hqe4iBNe9vw added.
2025-03-09 17:00:38 server1-1 | info: Server.Services.FileManagementService[0]
2025-03-09 17:00:38 server1-1 |   Using existing file storage directory at /app/UploadedFiles
2025-03-09 17:00:38 server1-1 | info: Server.Hubs.ChatHub[0]
2025-03-09 17:00:38 server1-1 |   ChatHub initialized.
2025-03-09 17:00:39 server1-1 | info: Server.Services.FileManagementService[0]
2025-03-09 17:00:39 server1-1 |   Using existing file storage directory at /app/UploadedFiles
```

(Anleitung für die Installation befindet sich in der README.md)