# Aufgabe 1: Einführung in Git, Schlanker Chat-Client

## 1.1 Schlanker Chat-Client

In dieser Aufgabe soll der Client einer minimalistischen Chatanwendung weiterentwickelt werden. Aus Nutzersicht soll die Anwendung wie in Abbildung 1 aussehen. Das Interface wird in HTML beschrieben und wird durch die Datei `index.html` vorgegeben, die Sie in Ihrem Repository finden.

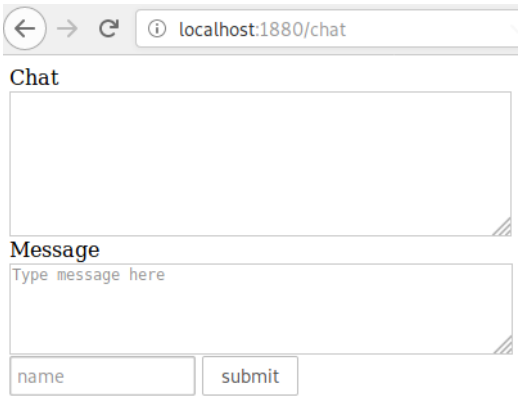


Abbildung 1: UI der Chatanwendung

Der Server ist mit der Datei `server.json` als Node-RED Flow (siehe Tafelübung) vorgegeben. Laden Sie Node-RED mit `module load node-red` bevor Sie den Server mit `node-red server.json` im Ordner `01_client/` im Git-Repository auf den Rechnern im CIP starten. Der Server muss nicht bei jeder Änderung neu gestartet werden, Änderungen werden automatisch neu ausgeliefert. Der Server liefert unter `localhost:1880/chat` den Inhalt der Datei `index.html` aus. Bitte überprüfen Sie dies, indem Sie `localhost:1880/chat` in Ihrem Browser öffnen.

Arbeiten Sie mit Git! Zum Ende der Abgabefrist muss Ihre Lösung auf dem GitLab Server gepusht sein. Wir empfehlen *regelmäßige* Commits und Pushes um Versionsstände zu sichern.

### 1.1.1 Testen des Clients, Benutzung von Browser Development Tools

Der Client wird zwar ohne client-seitige Logik ausgeliefert, ist aber bereits funktionsfähig. Testen Sie den Client, indem Sie zwei Browsertabs oder -fenster öffnen und Nachrichten in beide Richtungen austauschen. Dabei müssen Sie die Seite mit F5 aktualisieren, um neue Nachrichten abzurufen!

- Obwohl der Client im Auslieferungszustand kein JavaScript Code enthält, werden neue Nachrichten bei dem sendenden Client automatisch angezeigt, weil der Browser die Seite neu lädt. Analysieren Sie die Informationen im *Network* Tab der Browser Developer Tools (siehe Tafelübung) um nachzuvollziehen, warum dies funktioniert! Achten Sie dabei insbesondere auf die HTTP Status Codes.

### 1.1.2 Asynchrone HTTP Requests

Mit dem Entwicklungskonzept Ajax (siehe Tafelübung) können Webanwendungen HTTP Requests und Responses *asynchron* versenden und empfangen. Üblicherweise wird dann der DOM-Tree der bereits dargestellten Website manipuliert um beispielsweise neue Inhalte anzuzeigen. Der Grundbaustein für Ajax ist das JavaScript-Objekt `XMLHttpRequest`[1], das von allen gängigen Browsern implementiert wird.

Implementieren Sie das folgende Verhalten, indem Sie die Datei `index.html` um JavaScript-Code erweitern, der dann an die Browser ausgeliefert wird: Der Client soll einmal pro Sekunde einen HTTP `GET`-Request mit der Location `/savedMessages` an den Server senden, um so neue Nachrichten abzurufen. Der Inhalt der Response soll in das HTML Element `chatHistory` eingefügt werden. Verwenden Sie zur Implementierung die in der Tafelübung vorgestellte `setInterval`[2]-Funktion (alternativ auch `setTimeout`). Nutzen sie dazu Promises, entweder mit der `async/await` oder then Syntax um die asynchronen Requests zu behandlen.

**Zusätzliche Hinweise:**

- Neben den Manpages gibt es weitere hilfreiche Dokumentation über Git: Cheat Sheet von GitHub[3], Interaktiver Cheat Sheet[4], Umfassende Git Dokumentation als E-Book[5]
- Eine gute Ressource Rund ums Web ist `https://developer.mozilla.org/en-US/docs/Web#documentation_for_web_developers`
- Ihr könnte remote auf den CIP Rechnern Arbeiten mit `ssh -L 1880:localhost:1880 cip1e3.cip.cs.fau.de`

[1] `https://developer.mozilla.org/docs/Web/API/XMLHttpRequest`
[2] `https://developer.mozilla.org/docs/Web/API/WindowOrWorkerGlobalScope/setInterval`
[3] `https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf`
[4] `www.ndpsoftware.com/git-cheatsheet.html`
[5] `https://git-scm.com/book/`

**Letzter Commit bis zum 11. November.**

Präsentation der fertigen Lösung in der Woche der Abgabefrist in der Rechnerübung!