Die Folien sind für den persönlichen Gebrauch im Rahmen des Moduls gedacht.
Eine Veröffentlichung oder Weiterverteilung an Dritte ist nicht gestattet. (A. Claßen)

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Konzepte moderner Programmiersprachen (KMPS)

Praktikum:
Asynchrone Websocket Kommunikation

Wintersemester 2022/2023

Prof. Dr. Andreas Claßen
Fachbereich 5 Elektrotechnik und Informationstechnik
FH Aachen

Hinweis zum Testatsystem

Für den "Prof. Faßbender" Teil sind im Testatsystem P1 - P3 vorgesehen.

Dieses Praktikum hier wird im Testatsystem als Praktikum Nr. 4 verbucht werden.

Wahrscheinlich wird es insgesamt zwei Praktika im „Prof. Claßen" Teil geben…

Methodik der Ergebnisabgabe für dieses Praktikum

Erstellen Sie eine .zip Datei, in der folgende Dateien enthalten sind:

- Die Source Code Dateien von Client und Server
- Die individuelle Konsolen-Log-Datei (Eingaben & Ausgaben) jedes beteiligten Systems bei mindestens drei beteiligten Clients (und einem Server)
- Ferner eine (wahrscheinlich "per Hand erstellte") Textdatei, in der sie die Teile der Konsolen-Logs aller beteiligten Systeme in zeitlicher Chronologie eingefügt haben.

Laden Sie diese .zip Datei nach Ilias hoch (in die "Übung" zu diesem Praktikum).

***Einzelabgaben**! Teamarbeit ist erlaubt in dem Sinne, dass Sie die "technischen Knackpunkte" der Lösungen diskutieren und die Lösungsidee gemeinsam entwerfen. Aber die Lösungen sollten dann eine individuelle Umsetzung dieser gemeinsam entwickelten Ideen sein. D.h. Lösungen dürfen in Ansätzen ähnlich sein, aber nicht identisch.*

Ich habe letztes Jahr eine ganze Reihe von Lösungen nicht anerkannt, weil sie kopiert waren.
Ich würde / werde das auch dieses Jahr wieder machen…

Aufgabe / Anwendungsfall

Real-Time Kommunikation mit Web-Clients via Websockets, als ein relevanter Anwendungsfall für Concurrency: Asynchrones Handling der individuellen Websocket-Connections.

Anwendungsgebiete, in denen diese Technologie relevant ist:
Messaging Applikationen, Multiplayer Games, Collaboration Applikationen, Infrastruktursysteme mit Near-Real-Time Event Notifications (z.B. "Build Job xyz wurde erfolgreich beendet"), Applikationen mit Near-Real Time Updates von Location-Daten (aktueller Standort mobiler Clients) etc.

Gedachte Anwendung in diesem Praktikum:
Simulierte Tickets in einem Trouble Ticket System
=> Real-Time Kommunikation von: „wer kümmert sich um welches Ticket“, mittels "Selbst-Zuweisung". Siehe auch Beispiel-Ablauf.

Prgsprache der Realisierung: Go für Backend (Server), JavaScript für Clients.

Aufgabe / Anwendungsfall

Ausgabe Server	Benutzereingabe Server	Ausgabe Client 1	Benutzereingabe Client1	Ausgabe Client 2	Benutzereingabe Client2	Ausgabe Client 3	Benutzereingabe Client3
Server gestartet Keine Tickets n: neues Ticket. q: quit							
		Client gestartet Bitte Client-ID eingeben:					
			client1				
		Client: client1 Keine Tickets *number: Selbstzuw. q: quit*					
Neuer Client: client1 Keine Tickets n: neues Ticket. q: quit							
				Client gestartet Bitte Client-ID eingeben:			
					client2		
				Client: client2 Keine Tickets *number: Selbstzuw. q: quit*			
Neuer Client: client2 Keine Tickets n: neues Ticket. q: quit							
						Client gestartet Bitte Client-ID eingeben:	
							client3
						Client: client3 Keine Tickets *number: Selbstzuw. q: quit*	

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Anforderungen / Randbedingungen

Client: JavaScript Code, der entweder unter Node.js als Kommandozeilen-Client läuft oder als simpler Webclient im Browser mit (zusätzlichen) textuellen Ausgaben mittels `console.log` (Liste der Tickets und Zuweisungen) für ihre Abgabedatei.
Der Client muss auf jeden Fall u.a. auf irgendeine Art Eingaben erfragen (für Client-ID & für Selbstzuweisungen der Tickets), beim Webclient also über das Web-UI.
Es ist o.k., wenn beim Webclient die Webseite nicht per HTTP vom Backend Server ausgeliefert wird, sondern per File => Open in den Browser geladen werden muss …

Server: Go Kommandozeilenprogramm. *Es soll zur Laufzeit möglich sein, mittels Tastatureingabe neue Tickets hinzuzufügen. Z.B. mittels "n" => Server vergibt dann selbst eine neue sequentielle Nummer für das neue Ticket und setzt es in den Zustand "noch nicht zugewiesen" und kommuniziert es dann an die Clients als Teil des neuen "Gesamt-Datenzustands"*

"Richtige" Websockets (kein Long Polling, keine Server-Sent Events) und eine möglichst einfache Library zur Unterstützung in der Programmiersprache.
D.h. keine "Hybrid"-Lösungen wie socket.io und keine "Full Feature Plattformen", welche die ganze Kommunikation und den Code in Server und Client "intern handhaben" und wo man nur noch den Applikationscode "einhängen" muss.

Anforderungen / Randbedingungen

Server soll an jeden Client via Websocket immer auch eine oder mehrere Datenstruktur(-en) schicken mit Daten über alle Tickets und alle aktuellen Selbstzuweisungen.
D.h. es ist o.k., wenn ihr Client keinen state (Liste der Tickets bisherige Selbstzuweisungen) speichert.
Dies soll die Programmierung des Clients vereinfachen.
Sie können aber, wenn Sie möchten, den Client alternativ auch "stateful" machen und den Server nur die Updates/Änderungen schicken lassen...

Die Daten sollen alle als JSON Daten über die Websockets geschickt werden, nicht URL-encoded auf ein REST Interface o.ä.
D.h. es gibt nur eine URL und keine URL Parameter.
Einzige mögliche Ausnahme: Beim Verbindungsaufbau darf Information über den Client über URL Parameter eingebettet werden, falls nötig.

Anforderungen / Randbedingungen

Es ist o.k., wenn Fehlersituationen von ihrem Code nicht sauber gehandhabt werden. Es soll nur auf jeden Fall der „positiv-funktionierende Ablauf“ programmiert sein, bei dem mehrere Clients zum zentralen Server connecten und Daten-Updates (erfolgte Ticket-Selbstzuweisung der Clients) an den Server schicken und die "aktuelle Datensituation" (Gesamtliste aller Tickets inkl. neuer Tickets, Selbstzuweisungen aller Clients) in "near real time" von dort beziehen.

Vom Benutzer eingegebene Client-IDs sollen immer korrekt sein, d.h. kein Schutz gegen doppelt vergebene Client-IDs nötig.
Eingabe der Ticketnummer gegen Falscheingabe sichern, der Client soll möglichst nicht "abschmieren", wenn man sich bei der Ticketnummer mal vertippt
Sie können aber gerne voraussetzen, dass die Eingabe zumindest eine ganze Zahl ist...

Keine Authentifizierung (d.h. kein "komplizierter Login" im Client o.ä.).

Anforderungen / Randbedingungen

Die Daten-Updates werden vom Server zum Client über die Websocket-Verbindung gepushed, ohne dass der Client fragt. Dazu muss der Server aber eine Liste der verbundenen Clients pflegen.

Da Go bei mehreren Prozessoren „echt parallel“ arbeitet, sollen Sie ihre zentralen Datenstruktur (im Server) für die Ticketzuweisungen dadurch gegen konkurrierende Updates schützen, dass die ändernden Zugriffe darauf über Channel Kommunikation an eine Goroutine geleitet werden und nur diese eine Goroutine ändern darf. Das erspart die Einführung von Locks/Semaphoren zum Schutz der Datenstruktur.

Websocket Programmierung im JS Client

Siehe z.B.:

https://developer.mozilla.org/de/docs/WebSockets

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications

Websocket Programmierung im Go Server

Siehe:

https://github.com/gorilla/websocket
… oder …
https://pkg.go.dev/golang.org/x/net/websocket

JSON im Go Server

Siehe:

https://gobyexample.com/json

https://pkg.go.dev/encoding/json

Erfahrungen aus vergangenen Praktika

Wenn Sie eine HTML Seite als Formular ("FORM") designen, dann wird beim POST (d.h. beim Klicken auf die Buttons wie OK) die Seite neu geladen. Dabei wird dann aber die alte WebSocket Verbindung beendet und eine ganz neue WebSocket Verbindung aufgebaut. Das stört die Logik ihrer Anwendung, da diese davon ausgeht, dass Client und Server in einer durchgehenden Session miteinander kommunizieren, in der die Daten auf beiden Seiten erhalten bleiben und aktualisiert werden... Nicht das "FORM" Tag verwenden und bei den Buttons den Tag "BUTTON" verwendet (statt SUBMIT), dann bleibt das eine übergreifende Session.

Wenn Sie im Go Code mit "range" durch eine Liste iterieren, bekommen Sie als Werte *Kopien* der enthaltenen Elemente. Das hat dazu geführt, dass ihre Zuweisungen von Tickets nicht gespeichert wurden. "range" liefert aber auch den Index mit, über den man dann auf das Originalelement des Arrays zugreifen kann, wenn man dieses dauerhaft ändern möchte. Diese Problematik tritt auch in anderen Programmiersprachen immer wieder auf, wenn man sich nicht bewusst macht, ob man mit dem Original oder mit einer Kopie arbeitet...