Die Hausaufgaben müssen von jedem Studierenden einzeln bearbeitet und abgegeben werden. Für die Hausaufgabe sind die aktuellen Informationen vom Blog `https://seblog.cs.uni-kassel.de/ws2021/programming-and-modelling/` zu berücksichtigen.

Abgabefrist ist der 04.02.2021 - 23:59 Uhr

Abgabe

Wir benutzen für die Abgabe der Hausaufgaben Git. Jedes Repository ist nur für den Studierenden selbst sowie für die Betreuer und Korrektoren sichtbar.

Für die Hausaufgabe benötigt ihr ein **neues Repository**, welches über folgenden Link angelegt werden muss:

`https://classroom.github.com/a/KokiWtaz`

Nicht oder zu spät gepushte (Teil-)Abgaben werden mit 0 Punkten bewertet.

Abgaben, welche nicht kompilieren oder Laufzeitfehler beinhalten, werden mit 0 Punkten bewertet!

Aufgabe 1 - Projekt anlegen (25P)

Ziel dieser Hausaufgabe ist es, das bereits in den vergangenen Übungen erlernte Wissen anzuwenden, um die Grundlage einer Server/Client-Applikation zu erstellen. Hierfür wird wie zuvor bereits beschrieben ein neues Repository und somit auch ein neues Projekt angelegt (vgl. Seite 1 **Abgabe**).

Es folgen Vorgaben zu den zu verwendenden Technologien, sowie der Projektstruktur:

- Gradle Projekt muss den Namen PMWS2021_MiniChat_<Githubname> haben.
- .gitignore muss vor dem ersten Commit eingepflegt werden (Es kann die Vorlage aus HA04 genutzt werden).
- Für die Oberflächen muss JavaFx verwendet werden.
- Die Anwendung muss mittels MVC Pattern implementiert werden.
  - Das Modell wird in das Package de.uniks.pmws2021.chat.model generiert.
  - FXML-Dateien werden in src/main/resources im Package de.uniks.pmws2021.chat.view abgelegt.
  - Controller werden im Package de.uniks.pmws2021.chat.controller abgelegt.
- Analog zu HA07 müssen alle schreibenden Zugriffe auf das Datenmodell über einen Editor geschehen. Dieser heißt nun ChatEditor und ist im Package de.uniks.pmws2021.chat abzulegen.
- Analog zu HA06 müssen alle Szenenwechsel über einen StageManager geschehen. Dieser ist im Package de.uniks.pmws2021.chat abzulegen.

Datenmodell

Das folgende Datenmodell ist eine bindende Vorgabe und darf nicht verändert werden. Es muss entweder mit Fulib oder mit Fulib Scenarios generiert werden.

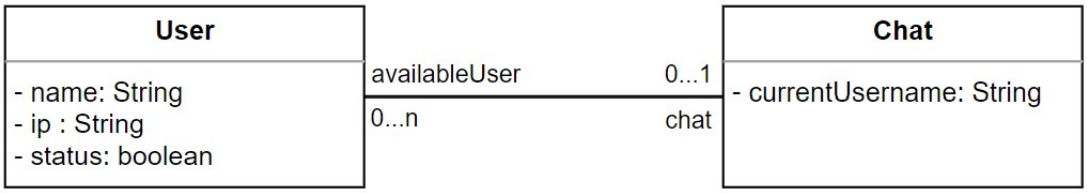

Abbildung 1: Datenmodell Chat

Aufgabe 2 - Oberfläche (18P)

Am Ende dieser Aufgabe sollen mindestens drei FXML-Dateien zu den folgenden Mockups erstellt werden. Es können weitere FXML-Dateien erstellt werden, um eine modulare View zu verwirklichen (analog zu SubViews aus HA07). Die folgenden Mockups sind lediglich Vorlagen, aber keine Vorgaben. Es ist euch somit freigestellt, das Design zu verändern. Es ist notwendig, dass die beschriebene Funktionalität der Screens in eurer Umsetzung enthalten ist. Bis auf den Szenenwechsel wird diese Funktionalität erst in den kommenden Hausaufgaben implementiert.

Aufgabe 2.1 - StartScreen

Der StartScreen bildet den Eintrittspunkt der MiniChat-Anwendung und dient der Auswahl von Server- oder Client-Oberfläche.

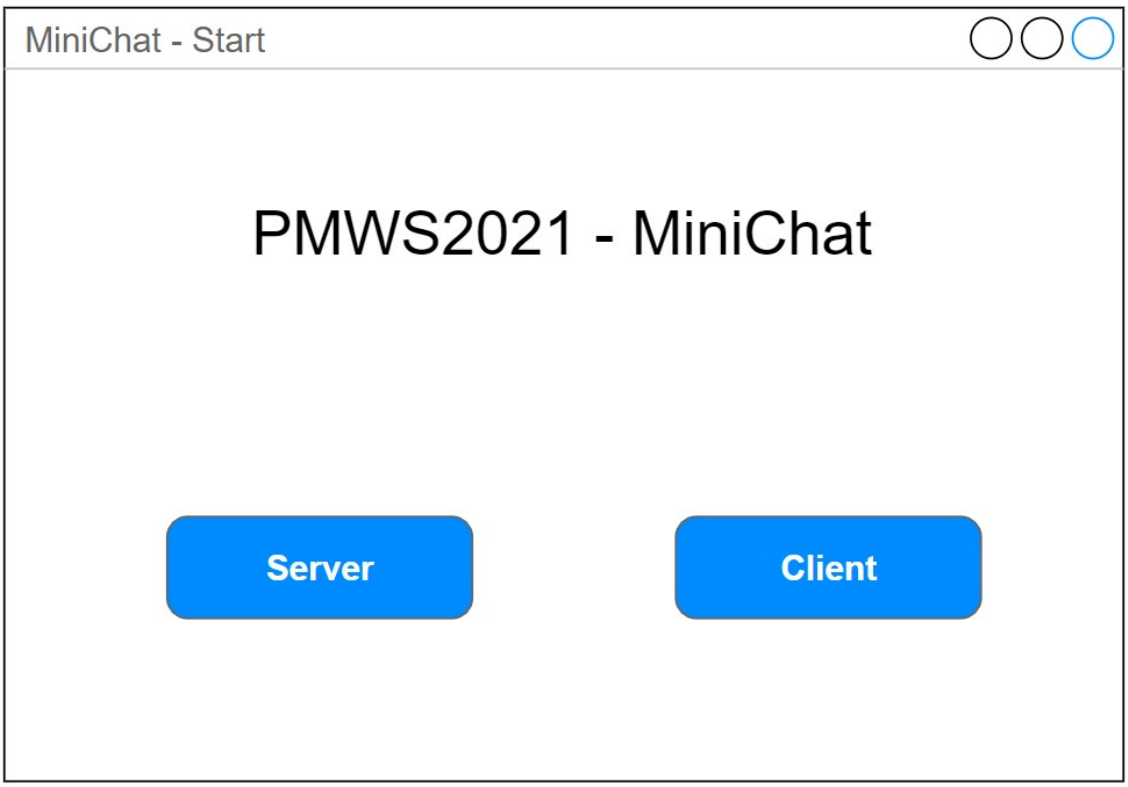

Abbildung 2: StartScreen

Funktionalität:

- Der Server-Button öffnet den ServerScreen.
- Der Client-Button öffnet einen Login-Dialog, in dem ein Name eingegeben werden muss. Im Anschluss wird der ClientScreen aufgerufen.

Aufgabe 2.2 - ServerScreen

Der ServerScreen ist zur Verwaltung aller Benutzer gedacht und bietet die Möglichkeit, einzelne oder alle Benutzer vom Server abzumelden.

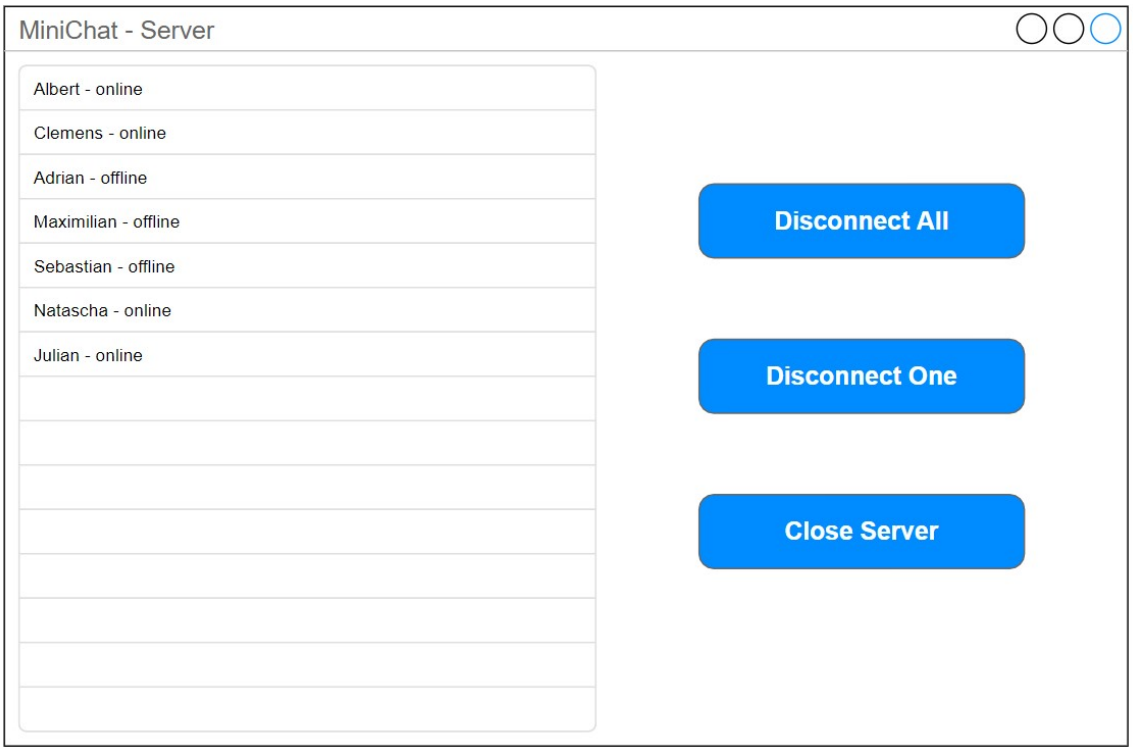

Abbildung 3: ServerScreen

Funktionalität:

- Es existiert eine Liste aller Benutzer. Dort werden Name und Online-Status angezeigt. Jeder der Einträge muss einzeln auswählbar sein.
- Der Disconnect All-Button meldet alle Benutzer ab, welche momentan online sind.
- Der Disconnect One-Button meldet den aktuell in der Liste ausgewählten Benutzer ab.
- Der Close Server-Button beendet den Server und öffnet den StartScreen.

Aufgabe 2.3 - ClientScreen

Im ClientScreen soll es möglich sein, mit anderen Benutzern zu chatten. Wird eine Nachricht mit selektiertem All-Tab gesendet, so erreicht diese Nachricht alle Benutzer, die online sind. Alternativ kann ein neuer Tab mit einem privaten Chat geöffnet werden, zum Beispiel durch das Doppelklicken eines Benutzers.

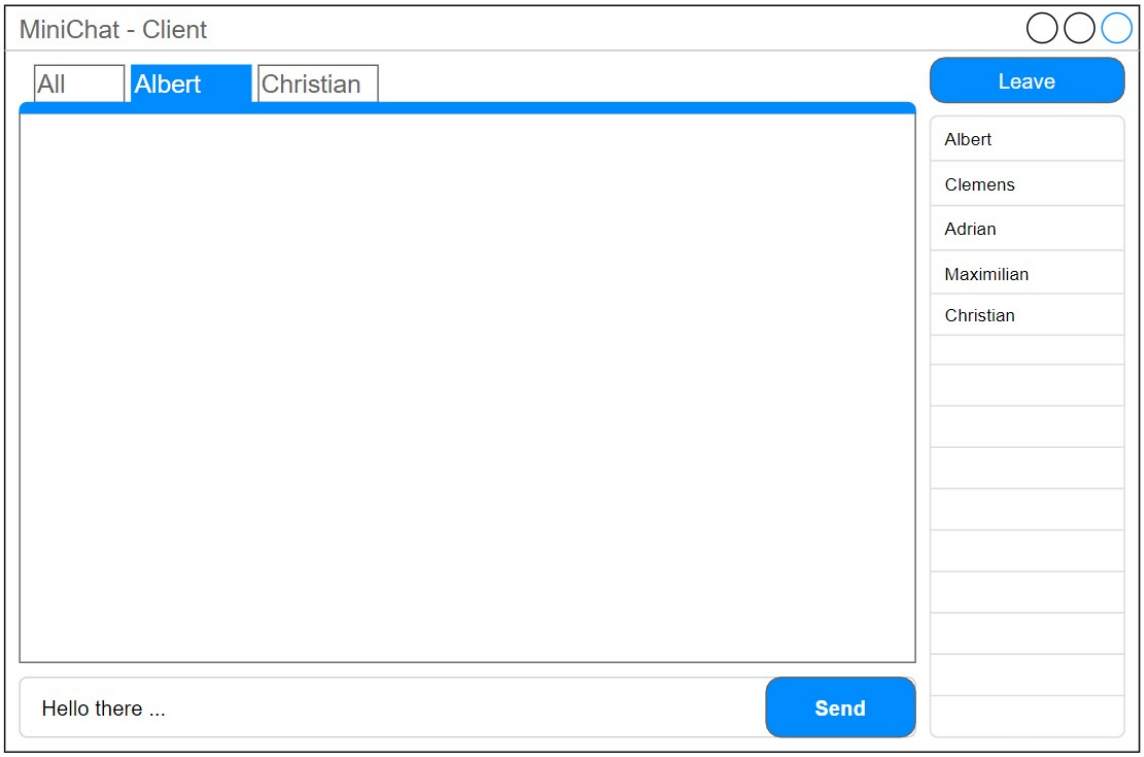

Abbildung 4: ClientScreen

Funktionalität:

- Zur Anzeige von verschiedenen Chats sollte eine TabView verwendet werden. Hierbei gibt es einen All-Tab, welcher nicht geschlossen werden kann.
- Ein TextField dient zur Formulierung von Chatnachrichten.
- Ein Send-Button versendet eine Chatnachricht. Optional kann dies zusätzlich über die Enter-Taste im Textfeld implementiert werden.
- Es existiert eine Liste aller momentan angemeldeten Benutzer, da nur mit diesen gechattet werden kann.
- Der Leave-Button beendet den Client und öffnet den StartScreen.

Aufgabe 3 - Funktionalität (25P)

Ziel dieser Aufgabe ist es, die Anwendung zu implementieren. Jede View wird dabei wie bekannt von einem eigenen Controller verwaltet. Zudem soll jeder Button mit Funktionalität versehen werden. In dieser Hausaufgabe soll noch keine Kommunikation über Websockets oder REST implementiert werden. Stattdessen verwenden wir Konsolenausgaben an den Stellen, wo später die eigentliche Implementierung erfolgt.

Beispiel:

Beim Klicken des Disconnect All-Button wird auf der Konsole "Disconnect all Users" ausgegeben.

Folgende Liste soll als Hilfestellung für die zu implementierende Funktionalität dienen.

StartScreen

Der StartScreen kann in seiner kompletten Funktionalität implementiert werden. (vgl Aufgabe 2.1)

ServerScreen

- Close Server Button -> Aktion auf Konsole ausgeben -> Scene wechseln
- Disconnect All Button -> Aktion auf Konsole ausgeben
- Disconnect One Button -> Aktion auf Konsole ausgeben

ClientScreen

- Leave Button -> Aktion auf Konsole ausgeben -> Scene wechseln
- Send Button -> Inhalt des TextFields auf Konsole ausgeben

Aufgabe 4 - Save&Load (13P)

Wie bereits in Hausaufgabe 08 umgesetzt, soll in dieser Anwendung das Speichern und Laden von Daten möglich sein. Es soll eine Liste von Usern gespeichert und geladen werden können. Diese Liste wird in der Anwendung von der Serverseite verwendet, um alle jemals eingeloggten User anzeigen zu können, auch wenn diese aktuell offline sind.

Legt eine Klasse mit dem Namen ResourceManager im Package de.uniks.pmws2021.chat.util an. Implementiert darin die Methoden saveServerUsers und loadServerUsers.

saveServerUsers bekommt ein User-Objekt übergeben. Die Methode lädt daraufhin die alten Daten aus der Speicherdatei, ersetzt gegebenenfalls das alte User-Objekt aus der geladenen Liste und speichert die aktualisierte Liste dann erneut in der Datei.

loadServerUsers lädt eine Speicherdatei und gibt deren Inhalt als Liste mit User-Objekten zurück.

Aufgabe 5 - Tests (19P)

Wie in vorherigen Hausaufgaben soll die implementierte Funktionalität getestet werden. **Ein** GUI-Test soll **alle** in Aufgabe 2 beschriebenen Szenenwechsel analog zu HA06 prüfen. Konkret soll dieser Test alle Buttons klicken, Textfelder ausfüllen und Fenstertitel überprüfen. Speichern und Laden sollen exakt wie in HA08 durch **einen** regulären Test, d.h. ohne Starten der Anwendung, abgedeckt werden. Es ist nicht notwendig, die Konsolenausgaben zu testen.