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/ws2122/programming-and-modelling/` zu berücksichtigen.

Abgabefrist ist der 17.02.2022 - 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 benötigst du **kein neues** Repository. Es wird das gleiche Repository benutzt, das bereits in Hausaufgabe 9 angelegt wurde. Dieses kann über folgenden Link erstellt werden, falls nicht bereits geschehen:

`https://classroom.github.com/a/xVKNlCc6`

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

Abgaben, die nicht lauffähig sind, werden mit 0 Punkten bewertet!

Das Ignorieren der Commit-Message-Vorgaben wird mit 0 Punkten bewertet!

Projekte, deren GUI nicht mit FXML-Dateien umgesetzt sind, werden mit 0 Punkten bewertet!

Vorbereitungen

Der Server gegen den du in dieser Hausaufgabe entwickelst, loggt dich nach 15 Minuten automatisch aus, solltest du es nicht vorher selber tun oder mit dem Server interagieren. In dieser Hausaufgabe wird neben REST auch die Technologie WebSocket eingesetzt. Daher wird teilweise veralteter Code gelöscht. Bitte prüfe daher sehr gründlich die Musterlösung/Vorgabe.

Musterlösung der vorherigen Aufgabe

Auch bei dieser Aufgabe kannst du auf deinem bestehenden Code aufbauen oder unsere Musterlösung[1] verwenden.

[1] `https://github.com/sekassel/pmws2122-files/tree/main/HA11`

Aufgabe 1 - Projekt erweitern (14P)

Als Vorbereitung auf die nachfolgenden Aufgaben muss das Projekt um ein paar Klassen erweitert werden. Darüber hinaus muss die `build.gradle` angepasst werden. Wie bereits bekannt enthält die Musterlösung an allen Stellen die du erweitern musst `TODO:` Kommentare.

Gradle

Füge die folgenden Zeilen in die `build.gradle`[2] in die `dependencies` ein:

```
implementation group: 'javax.websocket', name: 'javax.websocket-api', version: '1.1'
implementation group: 'org.glassfish.tyrus', name: 'tyrus-client', version: '1.15'
implementation group: 'org.glassfish.tyrus', name: 'tyrus-container-grizzly-client',
    version: '1.15'
```

Konstanten

Kopiere aus der Musterlösung[3] alle Konstanten in deine `Constants`-Klasse. Dort findest du nun Konstanten, die dabei helfen WebSockets einzubinden.

Klassen

Kopiere bzw. ergänze folgende Klassen aus der Musterlösung. Beachte dabei alte Methoden aus deinem Code zu löschen, sollten diese nicht mehr in der Musterlösung vorkommen:

- de.uniks.pmws2122.icp.StageManager[4]
- de.uniks.pmws2122.icp.util.JsonUtil[5]
- de.uniks.pmws2122.icp.net.RestService[6]
- de.uniks.pmws2122.icp.net.WebsocketClient[7]

In der Klasse `RestService` musst du folgende Methode, simultan zur letzten Hausaufgabe, implementieren:

- `getAllLoggedInUsers()`
  Liefert alle aktuell angemeldeten Nutzer zurück.

[2] `https://github.com/sekassel/pmws2122-files/blob/main/HA11/build.gradle`
[3] `https://github.com/sekassel/pmws2122-files/blob/main/HA11/Constants.java`
[4] `https://github.com/sekassel/pmws2122-files/blob/main/HA11/StageManager.java`
[5] `https://github.com/sekassel/pmws2122-files/blob/main/HA11/JsonUtil.java`
[6] `https://github.com/sekassel/pmws2122-files/blob/main/HA11/RestService.java`
[7] `https://github.com/sekassel/pmws2122-files/blob/main/HA11/WebsocketClient.java`

In der Klasse `WebSocketClient` musst du folgende Methoden implementieren:

- `WebsocketClient`-Konstruktor
  Erstellt und verbindet den WebSocket.
- `onOpen(Session session)`
  Händelt eine neu geöffnete Verbindung.
- `onClose(Session session, CloseReason reason)`
  Händelt eine geschlossene Verbindung.
- `onMessage(String message)`
  Händelt alle eingehenden Nachrichten.
- `sendMessage(String message)`
  Sendet eine Nachricht an den Server.
- `stop()`
  Schließt alle offenen Verbindungen zum Server.

Nicht entfernte `TODO:` führen zu Minuspunkten.

Aufgabe 2 - ModelService (3P)

Wie bei der letzten Hausaufgabe auch, musst du nun den `ModelService` ergänzen. Kopiere dafür den Inhalt aus der Musterlösung[8]. Beachte bitte auch hier, dass alte Methoden entfernt werden müssen. Jetzt musst du folgende Methoden implementieren:

- `buildUsers(JSONArray users)`
  Baut alle Nutzer aus einem Array.
- `buildUser(String username))`
  Baut einen Nutzer und fügt diesen dem Chat hinzu.
- `removeUser(String username)`
  Entfernt einen Nutzer aus dem Chat mit dem gegeben Namen.

Nicht entfernte `TODO:` führen zu Minuspunkten.

[8] `https://github.com/sekassel/pmws2122-files/blob/main/HA11/ModelService.java`

Aufgabe 3 - WebSocket verwenden (11P)

In dieser Aufgabe muss der `WebsocketClient` in den `ChatController` integriert werden.

Vergleiche dafür die Klasse `ChatController` aus der Musterlösung[9] mit deiner Implementierung. Ergänze nötigenfalls fehlende Methoden und Kommentare und entferne veralteten Code, welcher nicht mehr in der Musterlösung vorhanden ist. Auch hier dienen die `TODO:` Kommentare als Gedankenstütze.

Passe als erstes die `init`-Methode an, orientiere dich dabei an der Übung bzw. an den letzten Hausaufgaben.

Danach musst du die `stop`-Methode erweitern.

Als nächstes kannst du den neuen PropertyChangeListener `onOnlineUsersChanged` implementieren. Beachte hier die beiden Fälle, dass entweder der `newValue` oder der `oldValue` gesetzt sein kann.

Jetzt kann der alte REST-Aufruf zum Senden einer Chat-Nachricht durch den neu implementieren `WebsocketClient` ersetzt werden. Passe dafür die `onSendButtonPressed`-Methode an.

Abschließend musst du die `ChainOfResponsibility` mit den Methoden `onIncomingMessage`, `handleChatAction`, `handleUserJoinedAction` und `handleUserLeftAction`, wie in der Übung gezeigt, implementieren.

Nicht entfernte `TODO:` führen zu Minuspunkten.

[9] `https://github.com/sekassel/pmws2122-files/blob/main/HA11/ChatScreenController.java`

Aufgabe 4 - Remember Me (9P)

In dieser Aufgabe musst du die Anwendung um Persistenz erweitern. Es muss eine Datei mit Namen `config.json` erstellt werden. In dieser soll gespeichert werden, ob ein Nutzer während der Anmeldung seinen zuletzt benutzten Namen wiederverwenden will.

Erstelle dafür zuallererst folgende neue Klasse im Ordner src/main/java im entsprechenden Package und kopiere den Inhalt aus der Musterlösung:

- de.uniks.pmws2122.icp.util.ResourceManager[10]

Implementiere den statischen Konstruktor wie in der Übung gezeigt. Anschließend musst du die beiden Methoden `loadConfig` und `saveConfig` implementieren. Du kannst zum Speichern des `Default Content` die Klasse `JsonUtil` verwenden. Dort findest du eine vorgefertigte Methode.

Vergleiche nun deinen `LoginScreenController` mit der Musterlösunng[11] und ergänze die `TODO:` Kommentare.

Lade als erstes die neue `CheckBox` in der `init`-Methode. Beachte, dass dafür die `LoginScreen.fxml` angepasst werden muss. Für die neue `CheckBox` soll als fx:id `rememberMeCheckBox` verwendet werden. Du kannst frei entscheiden wo du dieses Oberflächenelement platzierst.

Ergänze zu guter Letzt die `init`- und `onLoginButtonPressed`-Methoden wie in den `TODO:` Kommentaren gefordert.

Nicht entfernte `TODO:` führen zu Minuspunkten.

[10] `https://github.com/sekassel/pmws2122-files/blob/main/HA11/ResourceManager.java`
[11] `https://github.com/sekassel/pmws2122-files/blob/main/HA11/LoginScreenController.java`

Bonusaufgabe - Privater Chat

Diese Aufgabe ist lediglich als Fingerübung gedacht und wird daher nicht bewertet.

Der Server bietet die Möglichkeit nicht nur global zu chatten, sondern auch privat mit allen Benutzern die online sind. Dafür muss eine Chatnachricht wie folgt erweitert werden.

Globale Nachricht senden

```
{"from": "Albert", "msg": "Hallo"}
```

Private Nachricht senden

```
{"from": "Albert",
 "to": "Sebastian", "msg": "Hallo"}
```

Für diese Aufgabe geben wir dir keine weiteren Tipps. Alles was du brauchst wurde bereits in dieser Hausaufgabe durchgeführt. Der Server speichert den privaten Chat, anders als den globalen, nicht ab. Du musst hier also auch die Chats auf eine beliebige Art speichern. Der `ResourceManager` sollte hier deine erste Anlaufstelle sein.

Wir wünschen dir viel Erfolg und Spaß bei dieser Aufgabe. Nach dieser Hausaufgabe stellen wir den Code für den Server bereit, damit du Zuhause weiter experimentieren und üben kannst.