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 10.02.2022 - 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

Bitte entferne in der Hausaufgabe alle Testklassen. Ab dieser Hausaufgabe würde das Testen deutlich komplizierter werden und den eigentlichen Lerneffekt überschatten.

Der Server gegen den du ab dieser Hausaufgabe entwickelst, loggt dich nach 15 Minuten automatisch aus, solltest du es nicht vorher selber tun. Du wirst in dieser Hausaufgabe bemerken, dass sich die Implementierung eines Chat-Clients nur mit REST schwierig gestaltet. Dies ist gewollt und wichtig für die nächste Hausaufgabe.

Musterlösung der vorherigen Aufgabe

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

Vergleiche deine `ChatScreen.fxml`[2] mit der Musterlösung. Der „All"-Tab sollte eine `ScrollPane` mit einer `VBox` als Inhalt aufweisen.

[1] `https://github.com/sekassel/pmws2122-files/tree/main/HA10`
[2] `https://github.com/sekassel/pmws2122-files/blob/main/HA10/ChatScreen.fxml`

Aufgabe 1 - Projekt erweitern (11P)

Als Vorbereitung auf die nachfolgenden Aufgaben muss das Projekt um ein paar Klassen erweitert werden. Darüber hinaus muss die `build.gradle` angepasst werden.

Gradle

Füge die folgende Zeile in die `build.gradle`[3] in die `dependencies` ein:

```
implementation group: 'com.konghq', name: 'unirest-java', version: '3.11.09'
```

Konstanten

Kopiere aus der Musterlösung[4] alle Konstanten in deine `Constants`-Klasse. Dort findest du nun Konstanten, die dabei helfen die verschiedenen Anfrage-URLs zu verwenden.

Klassen

Lege dafür unter src/main/java in den entsprechenden Packages folgende Klassen an und kopiere die `TODO:` Kommentare bzw. den Inhalt der Klassen aus der Musterlösung:

- de.uniks.pmws2122.icp.util.JsonUtil[5]
- de.uniks.pmws2122.icp.net.RestService[6]

Darüber hinaus müssen folgende Methoden im `RestService` implementiert werden:

- `login(String nickname)`
  Meldet einen Benutzer am Server an.
- `logout(String nickname)`
  Meldet einen Benutzer vom Server ab.
- `sendChatMessage(String from, String message)`
  Sendet eine Nachricht in den globalen Chat.
- `getAllMessages()`
  Liefert die letzten 100 Nachrichten aus dem globalen Chat.

Nicht entfernte `TODO:` führen zu Minuspunkten.

[3] https://github.com/sekassel/pmws2122-files/blob/main/HA10/build.gradle
[4] https://github.com/sekassel/pmws2122-files/blob/main/HA10/Constants.java
[5] https://github.com/sekassel/pmws2122-files/blob/main/HA10/JsonUtil.java
[6] https://github.com/sekassel/pmws2122-files/blob/main/HA10/RestService.java

Aufgabe 2 - ModelService (4P)

Als nächstes muss der `ModelService` angepasst werden, kopiere den Inhalt aus der Musterlösung[7] in deine Implementierung.

Du musst die folgenden Methoden implementieren:

- `buildChat(String nickname)`
  Erzeugt ein neues Chat-Objekt mit dem aktuellen Nicknamen.
- `buildAllChatMessages(JSONArray messages)`
  Wandelt eine Liste von `JSONObject` in `ChatMessage` um und hängt diese an das aktuelle Chat-Objekt.
- `clearAllChatMessages(String nickname)`
  Löscht alle Nachrichten des aktuellen Chat-Objekts.
  Diese Methode ist nur temporär und wird später entfernt!

Nicht entfernte `TODO`: führen zu Minuspunkten.

[7] `https://github.com/sekassel/pmws2122-files/blob/main/HA10/ModelService.java`

Aufgabe 3 - Server anbinden (10P)

Abschließend müssen der `StageManager` sowie die `Controller` angepasst werden, um mit dem Server zu kommunizieren.

Vergleiche dafür die Klassen `StageManager`[8], `LoginScreenController`[9] und `ChatScreenController`[10] aus der Musterlösung mit deiner Implementierung und ergänze nötigenfalls fehlende Methoden und Kommentare. Bedenke, dass die `TODO:` Kommentare lediglich als Gedankenstütze gedacht sind.

StageManager

In der `stop()`-Methode muss zum einem ein Logout ausgeführt werden, sofern du dich vorher eingeloggt hast. Zum anderen muss der Rest-Client gestoppt werden: `Unirest.shutDown()`. Dies ist notwendig, um die Anwendung ordentlich zu beenden.

LoginScreenController

Der `print`-Befehl aus der letzten Hausaufgabe muss durch einen Aufruf des `RestService` ersetzt werden. Im Anschluss musst du prüfen, ob dieser Aufruf erfolgreich war. Wenn ja, musst du mit dem `ModelService` das `Chat`-Objekt bauen lassen und zum `ChatScreen` wechseln. Falls nicht, soll eine Fehlermeldung ausgegeben werden.

ChatScreenController

Wie aus den letzten Hausaufgaben bekannt, musst du hier einen `PropertyChangeListener` korrekt an-/ bzw. abmelden. Dieser soll an das Attribut `Chat.PROPERTY_ALL_MESSAGES` angehängt werden.

Zudem musst du in der `init`-Methode als letzte Operation mit dem `RestService` alle Nachrichten des globalen Chats laden und diese mithilfe des `ModelService`s bauen lassen.

In der Methode `onAllMessagesChanged` muss jetzt für jede neue Nachricht ein `Label` in dem `All Tab` erzeugt werden. Der Text im `Label` sollte wie folgt formatiert sein:
`Sebastian: Jetzt wirds ernst!`

Zu guter Letzt müssen die beiden `ActionListener` für den `leaveButton` und den `sendButton` implementiert werden.

In der Methode `onSendButtonPressed` soll anstelle der Konsolenausgabe jetzt mithilfe des `RestService`s eine Chatnachricht an den Server gesendet werden. Sollte dies fehlschlagen, muss eine Fehlermeldung angezeigt werden. Andernfalls sollen alle alten Chat-Nachrichten aus dem aktuellen Chat und dem All-Tab entfernt werden. Anschließend müssen die Nachrichten wieder vom Server geladen werden, wie dies bereits in der `init`-Methode getan wurde.

[8] https://github.com/sekassel/pmws2122-files/blob/main/HA10/StageManager.java
[9] https://github.com/sekassel/pmws2122-files/blob/main/HA10/LoginScreenController.java
[10] https://github.com/sekassel/pmws2122-files/blob/main/HA10/ChatScreenController.java

In der Methode `onLeaveButtonPressed` sollst du dich mit dem `RestService` ausloggen und im positiven Falle zurück auf den `LoginScreen` wechseln, andernfalls zeige eine Fehlermeldung an.

Beispiele

Diese Beispiele sollen dabei helfen das Nachrichtenprotokoll des Servers zu verstehen.

Login Anfrage

```
{"name": "Albert"}
```

Login Antwort (erfolgreich)

```
"OK"
```

Logout Anfrage

```
{"name": "Albert"}
```

Logout Antwort (erfolgreich)

```
"OK"
```

Chat-Nachricht senden

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

Chat-Nachricht senden (erfolgreich)

```
"OK"
```

Liste der Chat-Nachrichten im globalen Chat

```
[
    { "from": "Albert", "msg": "Hallo" }
    { "from": "Albert", "msg": "Welt!" }
]
```