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 11.02.2021 - 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 das **bestehende MiniChat-Repository**, welches über folgenden Link angelegt werden kann, falls nicht bereits geschehen:

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

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

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

Vorbereitungen

Anpassungen der build.gradle

Für die Implementierung des Servers mit REST und Websockets müssen Anpassungen an der build.gradle vorgenommen werden. Um Copy-Paste-Fehlern vorzubeugen, findet ihr die hinzuzufügenden Zeilen auf unserem Blog im .zip-Archiv. Führt ein Gradle Refresh durch.

Fügt anschließend die restlichen Dateien aus dem .zip-Archiv ein. Ihr solltet vor Beginn der Aufgabe 1 folgende Dateien in euer Projekt eingefügt haben:

- de.uniks.pmws2021.chat.Constants
- de.uniks.pmws2021.chat.util.JsonUtil
- de.uniks.pmws2021.chat.util.ValidationUtil
- de.uniks.pmws2021.chat.util.ServerResponse
- de.uniks.pmws2021.chat.network.server.ChatServer
- de.uniks.pmws2021.chat.network.server.websocket.ChatSocket
- de.uniks.pmws2021.chat.network.server.controller.UserController

Aufgabe 1 - ChatServer (11P)

In dieser Aufgabe soll die ChatServer-Klasse vervollständigt werden. Hierzu dienen die Kommentare in der oben erwähnten Vorlage.

Der Konstruktor des ChatServers muss folgendes leisten:

- Den Port der Anwendung setzen
- ChatSocket und UserController initialisieren
- Folgende REST-Endpunkte definieren
  - GET `/api/users`
    Gibt alle eingeloggten Benutzer zurück
  - POST `/api/users/login`
    Führt die Anmeldung eines Benutzers aus
  - POST `/api/users/logout`
    Führt die Abmeldung eines Benutzers aus

Darüber hinaus müssen folgende Methoden implementiert werden:

- `disconnectUser(User user)`
  Meldet einen Benutzer vom Server ab
- `stopServer()`
  Meldet alle Benutzer ab und beendet den Server

Aufgabe 2 - ChatSocket (38P)

In dieser Aufgabe soll die ChatSocket-Klasse vervollständigt werden. Hierzu dienen die Kommentare in der oben erwähnten Vorlage.

Folgende Methoden müssen implementiert werden:

- `onNewConnection(Session session)`
  Registrieren einer neuen Verbindung mit einem Userclient.
- `onConnectionClose(Session session, int statusCode, String reason)`
  Eine bestehenden Verbindung mit einem Userclient wurde geschlossen.
- `onMessage(Session session, String message)`
  Verarbeitet die eingehende Nachricht und versendet sie an die richtigen Benutzer. Dabei müssen eingehende Nachrichten wie unten gezeigt umgeformt werden.
- `killConnection(User user, String reason)`
  Abmelden einer bestehenden Verbindung durch den Server.
- `sendUserJoined(User user)`
  Sendet eine Systemnachricht mit der Information, welcher Benutzer sich angemeldet hat.
- `sendUserLeft(User user)`
  Sendet eine Systemnachricht mit der Information, welcher Benutzer sich abgemeldet hat.
- `sendSystemMessage(String message)`
  Sendet die übergebene Nachricht an alle Userclients als Systemnachricht.

Hinweis

In den Vorlagen findet Ihr zwei Testklassen:

- SendWebSocketMessageTest -> Testet die Verbindung zum Websocket und das Versenden einer Nachricht im öffentlichen Chat
- SendPrivateWebSocketMessageTest -> Testet das Versenden einer privaten Nachricht

Durch diese Testklassen könnt Ihr euren Websocket testen. Bevor Ihr den Test ausführt, müssen folgende Schritte abgearbeitet werden:

1. Server Anwendung starten
2. Per Postman zwei User einloggen: Albert, Clemens
3. Tests ausführen

Beispielnachrichten

Systemnachrichten von Server zu Client

Login Action

```
{
    "channel": "system",
    "from": "server",
    "data": {
        "action": "joined",
        "user": "clemens"
    }
}
```

Logout Action

```
{
    "channel": "system",
    "from": "server",
    "data": {
        "action": "left",
        "user": "clemens"
    }
}
```

Nachrichten von Client zu Server

Public Chat

```
{
    "channel": "all",
    "msg": "Hallo ihr da"
}
```

Private Chat

```
{
    "channel": "private",
    "to": "clemens",
    "msg": "Na du"
}
```

Nachrichten von Server zu Client

Public Message

```
{
    "channel": "all",
    "from": "seb",
    "msg": "Hallo ihr da"
}
```

Private Message

```
{
    "channel": "private",
    "from": "seb",
    "msg": "Na du"
}
```

Aufgabe 3 - UserController (14P)

In dieser Aufgabe soll die UserController-Klasse vervollständigt werden. Hierzu dienen die Kommentare in der oben erwähnten Vorlage.

- `getAllLoggedInUsers(Request req, Response res)`
  Sendet eine Liste der aktuell angemeldeten Benutzer.
- `login(Request req, Response res)`
  Loggt den mitgesendeten Benutzer ein.
- `logout(Request req, Response res)`
  Loggt den mitgesendeten Benutzer aus.

Beispiele

Login Anfrage

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

Login Antwort (erfolgreich)

```
{
    "status": "success",
    "msg": "",
    "data": {
        "url": "/ws/chat"
    }
}
```

Logout Anfrage

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

Logout Antwort (erfolgreich)

```
{
    "status": "success",
    "msg": "",
    "data": {
        "msg": "bye"
    }
}
```

Liste der online User Antwort

```
{
    "status": "success",
    "msg": "",
    "data": [
        {
            "name": "Albert",
            "ip": "127.0.0.1",
            "status": true
        }
    ]
}
```

Aufgabe 4 - UI-Anbindung (17P)

Abschließend soll die Server-Applikation komplettiert werden. Hierzu muss euer Controller der ServerView (vgl. HA 09) angepasst werden.

init()

- Server starten
- Liste der gespeicherten Benutzer laden und anzeigen
- PropertyChangeListener für Benutzerliste, um neu angemeldete Benutzer in die angezeigte Liste aufzunehmen
- PropertyChangeListener für den Online-Status jedes Benutzers

stop()

- PropertyChangeListener entfernen
- Server stoppen

User speichern

Sobald ein User sich per REST das erste Mal einloggt, soll dieser in die Speicherdatei aufgenommen werden. Wählt selbstständig eine geeignete Stelle in der Serveranwendung, an der die Speicheroperation ausgeführt werden kann.

Buttons

Implementiert zuletzt die Funktionalität der Buttons des ServerScreens entsprechend der Beschreibungen aus HA 09.