

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/ws2324/programming-and-modelling/> zu berücksichtigen.

Abgabefrist ist der 01.02.2024 - 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 10 angelegt wurde. Dieses kann über folgenden Link erstellt werden, falls nicht bereits geschehen:

https://classroom.github.com/a/VB_hZStE

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

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

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

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.

Gradle

Füge die folgenden Zeilen in deine `build.gradle` unter `dependencies` ein, um in deinem Projekt die Bibliothek „Unirest“ zu verwenden:

```
// https://mvnrepository.com/artifact/com.konghq/unirest-java
implementation group: 'com.konghq', name: 'unirest-java', version: '3.14.1'
```

ChatApiService

Lege unter `src/main/java` im Package `de.uniks.pmws2324.nopm.service` die Klasse `ChatApiService` mit folgenden Methoden an:

- **public** `LoginDto login(LoginDto dto)`
Meldet einen Benutzer am Server an. Gibt diesen zurück, wenn die Anfrage erfolgreich war.
- **public void** `logout(LoginDto dto)`
Meldet einen Benutzer vom Server ab.
- **public** `List<Topic> getTopics()`
Gibt alle Topics vom Server zurück.
- **public** `MessageDto sendMessage(String topic, CreateMessageDto dto)`
Sendet eine Nachricht in das Topic und gibt die vollständige Nachricht zurück.
- **public** `List<MessageDto> getMessages(String topic, Instant after)`
Liefert die letzten Nachrichten aus dem Topic. Falls `after` angegeben wird (d.h. nicht `null`), werden nur die Nachrichten zurückgegeben, deren Timestamp nach diesem Zeitpunkt liegt.

Die Methoden müssen zunächst noch nicht implementiert werden, dies erfolgt in Aufgabe 3. Es genügt, zunächst den Rumpf leer zu lassen bzw. `null` oder leere Listen zurückzugeben.

Erstelle die benötigten Dtos als `records` im Package `de.uniks.pmws2324.nopm.model.dtos`.

ChatService

Als nächstes muss der `ChatService` im Package `de.uniks.pmws2324.nopm.service` angelegt werden. Der `ChatService` soll einen `ChatApiService` als Konstruktor-Parameter erhalten und in einer Klassenvariable speichern. Folgende Methoden im `ChatService` sollen **vollständig** implementiert werden:

- **public** `User login(String nickname)`
Meldet einen Benutzer am Server an.
- **public void** `logout(String nickname)`
Meldet einen Benutzer vom Server ab.

- **public** `List<Topic> getTopics()`
Gibt alle Topics vom Server zurück.
- **public** `Message sendMessage(Topic topic, User sender, String body)`
Sendet eine Nachricht und fügt sie dem Topic hinzu.
- **public void** `loadMessages(Topic topic, Instant after)`
Lädt die letzten Nachrichten des Topics. Falls `after` angegeben wird (d.h. nicht `null`), werden nur die Nachrichten hinzugefügt, deren Timestamp nach diesem Zeitpunkt liegt.

Aufgabe 2 - Oberfläche anbinden (15P)

Abschließend müssen die **App** sowie die **Controller** angepasst werden, um mit dem Server zu kommunizieren.

App

In dieser Klasse soll der **ChatService** angelegt und an Controller weitergereicht werden.

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

LoginController

Die Konsolenausgabe aus der letzten Hausaufgabe muss durch einen Aufruf des **ChatService** ersetzt werden. Im Anschluss musst du prüfen, ob dieser Aufruf erfolgreich war. Wenn ja, musst du zum App-Bildschirm wechseln. Falls nicht, soll eine Fehlermeldung in die Konsole ausgegeben werden.

AppController

In der **init**-Methode sollen die Topics geladen und als Tabs mit zugehörigen **ChatController** dargestellt werden. Lösche die beispielhaften Topics.

Außerdem muss der **Logout**-Button statt einer Konsolenausgabe die entsprechende Methode im **ChatService** aufrufen.

ChatController

Du musst in der **init**-Methode mit dem **ChatService** alle Nachrichten des Topics laden und mithilfe des **MessageController** darstellen.

Des Weiteren muss ein **Timer** verwendet werden, der alle 5 Sekunden die neuen Nachrichten lädt. Dafür muss eine Variable `lastFetchTimestamp` gespeichert werden, die als **after**-Parameter verwendet werden soll.

Bei Betätigen des Send-Buttons soll nun anstelle von `topic.withMessages(new Message()...)` mithilfe des **ChatService** eine Chatnachricht an den Server gesendet werden. Die neue Nachricht wird automatisch über den zuvor angemeldeten `PropertyChangeListener` dargestellt, der Rückgabewert von **sendMessage** muss also nicht verwendet werden.

Aufgabe 3 - Server anbinden (12P)

Unter folgendem Link findest du die Swagger-Dokumentation des NoPM-Servers:

<https://nopm.uniks.de/api/v1>

Füge diese URL als Konstante `API_URL` in einer Klasse namens `Constants` ein und nutze sie im `ChatApiService`.

Füge außerdem folgenden Code im `ChatApiService` ein, damit alle HTTP-Anfragen als `JSON` versendet werden:

```
static
{
    Unirest.config().addDefaultHeader("Content-Type", "application/json");
}
```

Verwende die beschriebenen Endpunkte mit Unirest, um die Methoden im `ChatApiService` wie in Vorlesung und Übung gezeigt zu implementieren. Schreibe zum Konvertieren von Objekten von/zu JSON eigene statische Hilfsmethoden wie `LoginDto.toJson` oder `MessageDto.fromJson`.