



**Hausaufgabe 10 (Informatik)**

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 25.01.2024 - 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 **ein neues** Repository.

Dieses kann über folgenden Link erstellt werden, falls nicht bereits geschehen:

[https://classroom.github.com/a/VB\\_hZStE](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 anlegen (5P)

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**).

### Grundstruktur

Als Grundlage ist das vorhandene Projekt in deinem neuen Repository gegeben. Es gelten folgende Vorgaben zu den zu verwendenden Technologien, sowie der Projektstruktur:

- Für die Oberflächen muss JavaFx verwendet werden.
- Die Anwendung muss mittels MVC-Pattern implementiert werden.
  - Das Modell wird in das Package `de.uniks.pmws2324.nopm.model` mithilfe von `GenModel` generiert.
  - FXML-Dateien werden unter `src/main/resources/de/uniks/pmws2324/nopm/view` abgelegt.
  - Controller werden im Package `de.uniks.pmws2324.nopm.controller` abgelegt.
  - Analog zur Übung müssen alle Szenenwechsel über die `App` geschehen. Diese ist im Package `de.uniks.pmws2324.nopm` abzulegen.
  - Die Klasse `Main` startet die App.
- Analog zu den vorherigen Hausaufgaben müssen alle schreibenden Zugriffe auf das Datenmodell über `Services` im Package `de.uniks.pmws2324.nopm.service` geschehen.

### Datenmodell

Das folgende Datenmodell ist eine bindende Vorgabe und darf nicht verändert werden. Es muss mithilfe von Fulib und der GenModel-Klasse generiert werden.

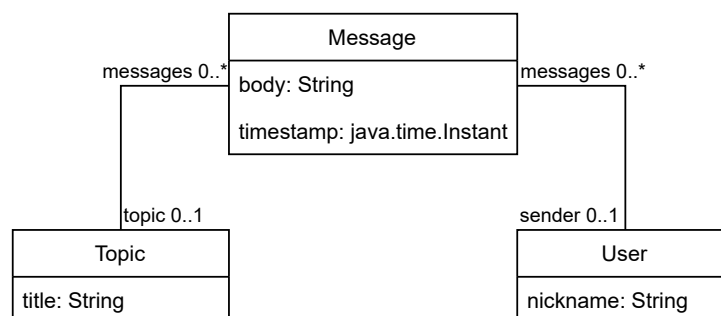


Abbildung 1: Datenmodell NoPM

## Aufgabe 2 - View erstellen (6P)

Am Ende dieser Aufgabe sollen FXML-Dateien zu den folgenden Wireframes erstellt werden. Es ist notwendig, dass die beschriebenen GUI-Elemente mit den vorgegebenen `fx:ids` enthalten sind.

### Login

Der Login-Bildschirm bildet den Eintrittspunkt der NoPM-Anwendung und dient als Anmeldemaske.

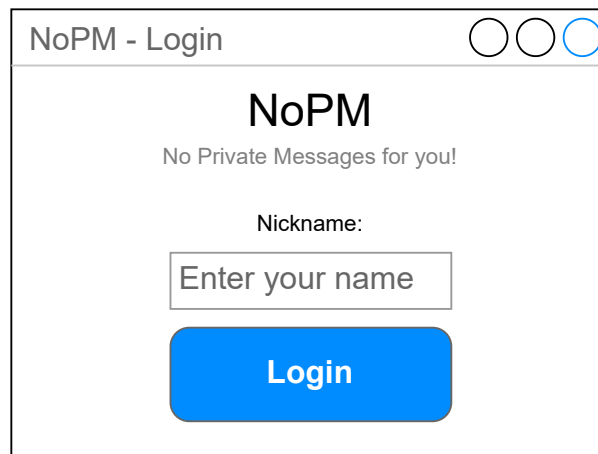


Abbildung 2: Login

### Login.fxml

- In das `Nickname-TextField` kann ein beliebiger Nickname eingegeben werden (`fx:id nicknameField`).
- Der `Login-Button` führt ein Login durch und wechselt zum Chat-Bildschirm (`fx:id loginButton`).

## Chat

Im Chat-Bildschirm soll es möglich sein, in verschiedenen Topics zu chatten. Wird eine Nachricht gesendet, so erreicht diese Nachricht alle Benutzer, die das Topic ausgewählt haben. Aufgrund der Komplexität dieser Ansicht wird sie in drei fxmles und Controller aufgeteilt.

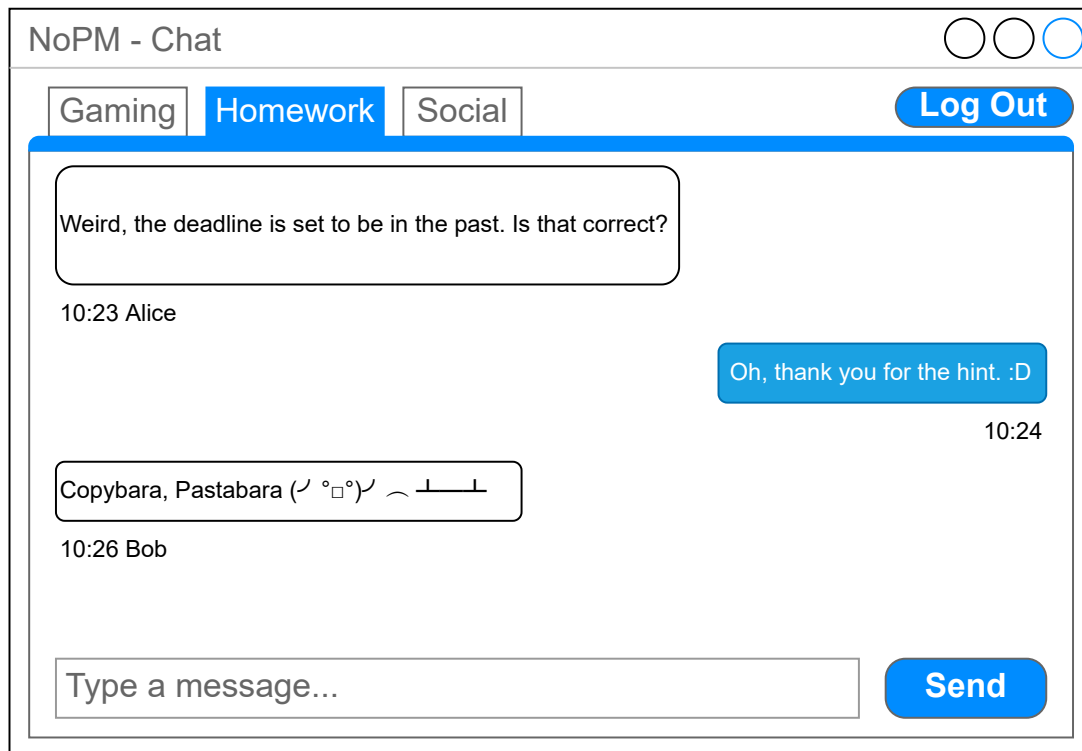


Abbildung 3: Chat

### App.fxml

Die Hauptansicht unserer Anwendung wird durch den `AppController` verwaltet.

- Zur Anzeige von verschiedenen Chats soll eine `TabPane` (fx:id chatTabPane) verwendet werden.
- Der `Logout-Button` loggt den Benutzer aus und öffnet den Login-Bildschirm (fx:id logoutButton).

### Chat.fxml

Der Inhalt der `TabPane` wird jeweils durch einen `ChatController` verwaltet.

- Ein `VBox` enthält alle Nachrichten (fx:id messageBox). Diese darf optional in eine `ScrollPane` (fx:id messageScroll) eingesetzt werden, um lange Nachrichtenketten darzustellen.
- Ein `TextField` dient zur Formulierung von Chatnachrichten (fx:id messageField).

- Ein **Send-Button** versendet eine Chatnachricht (fx:id `sendButton`). Zusätzlich soll das Versenden einer Nachricht durch Drücken der Enter-Taste im Textfeld implementiert werden (Hinweis: „Default Button“).

### Message.fxml

Nachrichten werden durch einen separaten **MessageController** realisiert. Dieser sollte folgende Elemente enthalten und verwalten:

- Die Sprechblase der Nachricht (fx:id `messageBubble`, z.B. **TextFlow**)
- Den **Text** der Nachricht (fx:id `bodyText`)
- Den Zeitstempel und Absender als **Text** (fx:id `infoText`)

## Aufgabe 3 - Funktionalität (9P)

Ziel dieser Aufgabe ist es, die Anwendung zu implementieren. Jede View wird dabei wie bekannt von einem eigenen Controller verwaltet.

Jeder Button soll mit Funktionalität versehen werden. In dieser Hausaufgabe soll noch keine Kommunikation über HTTP oder WebSockets implementiert werden. Stattdessen werden Konsolenausgaben an den Stellen platziert, an denen später die eigentliche Implementierung erfolgt.

### Login

Der Login-Button soll „**Login** <Nickname>“ in der Konsole ausgeben und danach die Szene zum **AppController** wechseln.

### App

Der Leave-Button soll „**Logout** <Nickname>“ in der Konsole ausgeben und danach die Szene zurück zum Login wechseln.

Die Tabs sowie zugehörige **ChatController** sollen anhand von einer selbst gewählten Topic-Liste (mindestens zwei verschiedene Topics) erstellt werden. Achte dabei auf die korrekte Verwaltung der Subcontroller.

### Chat

Der Send-Button soll eine neue Nachricht mit `topic.withMessages(new Message()...)` hinzufügen und das Message-Feld leeren.

Wie aus den letzten Hausaufgaben bekannt, musst du außerdem einen `PropertyChangeListener` korrekt an- bzw. abmelden. Dieser soll an das Attribut `Topic.PROPERTY_MESSAGES` angehängt werden und neue Nachrichten mit **MessageController** darstellen. Achte dabei auf die korrekte Verwaltung der Subcontroller.

### Message

Die Felder für Body und Information (Absender und Zeitstempel) sollen anhand der **Message** ausgefüllt werden. Beachte, dass Nachrichten nicht bearbeitet werden können und daher keine `PropertyChangeListener` benötigen. Der Zeitstempel kann mit folgendem Code als Uhrzeit dargestellt werden:

```
private static final DateTimeFormatter TIME_FORMAT
    = DateTimeFormatter.ofPattern("HH:mm");
```

```
timestamp.atZone(ZoneId.systemDefault()).format(TIME_FORMAT)
```

### Fenstertitel

In dieser Anwendung sollen Fenstertitel gesetzt werden. Orientiere dich dabei an den Wireframes (Abbildung 2 und 3).



## Aufgabe 4 - Tests (3P)

Wie in den vorherigen Hausaufgaben soll die implementierte Funktionalität getestet werden. **Ein** GUI-Test soll **alle** in Aufgabe 3 beschriebenen Szenenwechsel prüfen. Implementiere diesen Test in einer Klasse namens `AppTest`. Konkret soll dieser Test alle Buttons klicken, Textfelder ausfüllen und Fenstertitel überprüfen. Die Konsolenausgaben sollen **nicht** getestet werden. Jedoch soll das Anzeigen von Nachrichten überprüft werden.

**Alle Tests müssen erfolgreich ausführbar sein, andernfalls wird mit 0 Punkten bewertet!**