TECHNISCHE UNIVERSITÄT
CHEMNITZ

Implementation eines Filme-Managers

Projektbericht

563100 Advanced Management of Data
Professur Datenverwaltungssysteme
Fakultät für Informatik

Eingereicht von:
Robin Gerstmann (649418)
Axel Obrikat (676010)

Einreichungsdatum:
09.02.2022

Betreuer:
Dr. Frank Seifert
Daniel Richter
Florian Hahn

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

DB Datenbank

DBMS Datenbank Management System

DDB Distributed Database

DDBMS Distributed Database Management System

DDL Data Definition Language

DML Data Manipulation Language

ERM Entity Relationship Model

HTML Hypertext Markup Language

RM Relationenmodell

SQL Structured Query Language

UML Unified Modelling Language

1. Einleitung

Axel Obrikat

Das heutige Angebot an verschiedenen Filmen übertrifft jegliche Ausmaße. Kein begeisterter Filme-Gucker kann genau sagen, welche Filme er bereits gesehen hat. Diese Fülle an Filmen wird nur noch übertroffen an Möglichkeiten, diese anzuschauen. Ob Streamingdienste wie Netflix, Amazon Prime oder DisneyPlus, ob per CD oder Blue-Ray-Disk oder klassisch im Kino - all das sind Möglichkeiten, die unterschiedlichsten Filme anschauen zu können. Es ist wirklich nicht schwer, die Übersicht zu verlieren.

Eine Art privater Filme-Manager kann Einzelpersonen dabei helfen, den Überblick über bereits geschaute Filme zu wahren. Dieser Bericht stellt in den folgenden Seiten die Überlegungen und Implementationen eines solchen prototypischen Systems vor. Der Manager verwaltet dabei mehrere Benutzerprofile, was den Vorteil birgt, dass durchschnittliche Filmbewertungen oder -vorschläge bereitgestellt werden können. Die spezifischen Anforderungen an den Filme-Manager sind im Anhang A Anforderungsliste vermerkt.

Alle Daten werden in einer Datenbank von PostgreSQL[1] gespeichert. Zur Anzeige und Bearbeitung der Filminformationen durch die Benutzer wird zusätzlich ein Frontend - also eine Benutzeroberfläche - angeboten, welche mit der Datenbank verbunden ist. Jegliche Programmlogik ist in die Datenbank ausgelagert, sodass das Frontend zur reinen Darstellung und Bedienung der Inhalte dient.

Das Frontend wurde in der Programmiersprache Python[2] innerhalb der Entwicklungsumgebung Visual Studio Code[3] implementiert. Für eine schnellere Entwicklung der grafischen Aspekte der Benutzeroberfläche wurde PyQt Designer[4] mit dem dazugehörigen Python Modul PyQt[5] verwendet.

Zur Kennzeichnung der Aufgabenteilung und zum Hervorheben, wer welchen Teil des Berichts geschrieben hat, sind für den Teil des Berichts die Namen unterhalb der jeweiligen Überschriften aufgezeigt. Die Vorüberlegungen und Planung, sowie die Erstellung aller Modelle und des Paper-Prototypes, wurden in gemeinsamer Arbeit erledigt. Ebenso liegt der programmierpraktische Teil hauptsächlich einer gemeinsamen Bearbeitung zu Grunde, wobei der größere Fokus von Herrn Robin Gerstmann auf dem Frontend und der von Herrn Axel Obrikat auf der Datenbank lag.

[1] siehe `https://www.postgresql.org/` [Online] Abgerufen am 02.02.2022.
[2] siehe `https://www.python.org/` [Online] Abgerufen am 02.02.2022.
[3] siehe `https://code.visualstudio.com/` [Online] Abgerufen am 02.02.2022.
[4] siehe `https://doc.qt.io/qt-5/qtdesigner-manual.html` [Online] Abgerufen am 02.02.2022.
[5] siehe `https://www.qt.io/` [Online] Abgerufen am 02.02.2022.

2. Planung

Axel Obrikat

Der Zweck des Datenbank Management System (DBMS) geht bereits aus der Einleitung hervor. Dem Benutzer soll es erleichtert werden, den Überblick über bereits geschaute Filme zu wahren. Zusätzlich sollen auf Zusatzfeatures wie der Ausgabe von durchschnittlichen Bewertungen sowie Filmvorschlägen zugegriffen werden können. Mit Hilfe einer grafischen Benutzeroberfläche, welche direkt mit der PostgreSQL-Datenbank verbunden ist, sollen den Benutzern die Bedienung und Verwaltung erheblich erleichtert werden. Diese sollen intuitiv und ohne dem Lesen eines Handbuchs die Oberfläche bedienen können.

Die Überschneidung verschiedener Nutzeraktivitäten besteht zum einen darin, dass Nutzer Filme sowie in Filmen mitwirkende Personen verwalten und löschen können, die sie selbst nicht hinzugefügt haben. Die Bewertung von Filmen kann ein Nutzer aber nur für sich vornehmen, allerdings ist für jeden die Ansicht einer gemittelten Bewertung pro Film möglich. Damit spielen indirekt also nicht nur die Filme eine Rolle, die ein Nutzer selbst hinzugefügt hat, sondern auch jene, die andere Benutzer hinzugefügt haben.

Nachdem diese Vorüberlegungen, die mehr auf den Blickwinkel der Anwender eingehen, abgeschlossen sind, wird der Fokus mehr auf die Entwicklersicht gelegt. Die verwendeten Technologien zur Programmierung des Filme-Managers sind in der Einleitung erwähnt. Die Spezifikation auf diese Technologien hat sich während der Planungsphase aufgrund zweier Aspekte ergeben.

1. Der Filme-Manager soll unabhängig vom Betriebssystems des Computers funktionieren.

2. Dieses Projekt soll innerhalb nur eines Monats fertig gestellt werden.

Unter Berücksichtigung von 1.) ist die Implementation einer nativen Applikation automatisch ausgeschlossen, da solche Anwendungen komplett vom Betriebssystem abhängig sind. Webbasierte oder hybride Anwendungen sind im Vergleich dazu plattformunabhängig und eignen sich für die Programmierung des Filme-Managers. Webapplikationen werden in der Sprache Hypertext Markup Language (HTML) programmiert und können von den Benutzern über einen Browser aufgerufen werden. Der Nachteil einer durchgängigen Internetverbindung zum Aufruf einer Webapplikation wird dabei aufgehoben, da zur Benutzung der Anwendung sowieso eine Verbindung zur Datenbank (DB) bestehen muss. [1]

Hybride Applikationen haben den Anspruch, sowohl plattformunabhängig sein zu können, als auch betriebssystembedingte (native) Vorteile zu nutzen, wie der

einfache Zugriff auf Sensordaten des Endgeräts. [2] Sowohl die webbasierte, als auch die hybride Form erfüllen die Anforderung der Plattformunabhängigkeit.

In Hinblick auf 2.) und aufgrund persönlicher Präferenzen der Entwickler ist die Anwendung nicht in HTML programmiert und damit keine Webanwendung. Der Filme-Manager ist in einer hybriden Applikation umgesetzt und mit Hilfe des Moduls PyQt sowie der Umgebung PyQt-Designer in Python implementiert. Die Entwicklungsumgebung bietet dabei einen Vorteil. Zur Beschleunigung des Entwicklungsverfahrens wird die Methode des Rapid Application Developments angewendet, sodass mit Hilfe von mächtigen Programmierumgebungen und vordefinierten Frameworks sowie Werkzeugen der Implementationsprozess verschnellert wird. [2]

Unter Anwendung des Programms PyQt-Designer ist es dem Entwickler möglich, die grafische Benutzeroberfläche via Drag & Drop vordefinierter Schaltflächen zu gestalten. Der notwendige Code zur Erstellung der Oberfläche wird im Hintergrund automatisiert erzeugt, welcher im weiteren Verlauf durch den Entwickler angepasst werden kann. Diese spezielle Methode der rapiden App-Entwicklung wird Low-Code-Development genannt. [3]

3. Konzeptuelle Datenmodellierung

Robin Gerstmann

Bei der Konzeptionierung des Entity Relationship Model (ERM) sind vor allem die genannten Punkte aus der Anforderungsanalyse wichtig. Davon ausgehend muss ein Datenbankdesign geschaffen werden, dass jede Anforderung an den Film-Managers erfüllt.

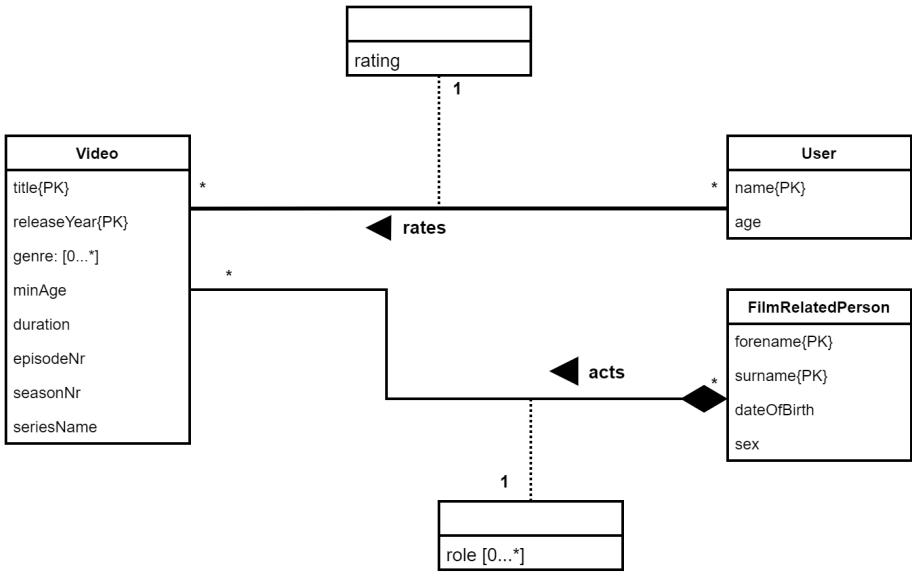

Abb. 3.1.: Aufbau des Datenbankdesigns als UML-Diagramm

Zuerst ist es wichtig, die zentralen Entitäten zu definieren und aufzuschreiben. In Abbildung 3.1 sind die folgenden Entitäten zusehen: Video, User und FilmRelated-Person. Alle drei Basis-Entitäten bilden die Grundlage für die kommende Datenbank und deren logischen Aufbau. Bei der User-Entität war es wichtig, dessen Attribut »Namen« als Primär-Schüssel zu festzulegen. Somit darf später nur jeder Name einmal vorkommen. Des Weiteren besitzt der User noch das Attribut »age«, welches bei Erweiterung des Projektes genutzt werden kann z.B. Altersfreigabe.

Eine weitere wichtige Entität ist Video. Diese sollte alle elementaren Attribute für einen Film oder Serie beinhalten. In diesem Projekt wird jedoch nicht zwischen Serie und Film direkt unterschieden. Eine Vererbung der Klasse Video mit eignen Attributen wäre durchaus möglich, jedoch aufgrund von Umfang und Zeit nicht genutzt. Deshalb wird in diesem Projekt durch das Attribut »seasonNr« zwischen Film und Serie unterschieden. Dieses Attribut darf »NULL« sein, da es sich demnach um

einen Film handelt. Der Primär-Schlüssel dieser Entität besteht aus einer Kombination von »Titel« und »Jahr der Veröffentlichung«. Das bringt den Vorteil mehrere Filme und Serien hinzuzufügen, ohne auf Kollision zu stoßen.

Um die Filme sowie Serien besser kategorisieren zu können existiert das Attribut »genre«. Es ist eine Liste, die keine bis unendlich vielen Einträge besitzen kann. Beispiele können hier Horror, Komödie oder Thriller sein. Dieses Attribut besitzt eine wichtige Aufgabe, da dieses zur Grundlage für den späteren Empfehlungs-Algorithmus genutzt wird. Um den Videos eine Altersempfehlung zu geben, besitzt es die Eigenschaft »minAge«, die sich auf die entsprechende Referenz für FSK bezieht. Eine weitere Eigenschaft um die Länge bzw. Dauer eines Film zu speichern ist das Attribut »duration«. Hier kann der Nutzer später die Dauer von Serien oder Filmen speichern.

Auf den ersten Blick suggeriert das Attribut »episodeNr« eine Eigenschaft zu sein, die nur für Serien bestimmt ist. Wie bereits erwähnt wurde, findet die Unterscheidung nur durch »seasonNr« statt. Jedoch kann ein Film aus mehreren Teilen bestehen und wird letztlich durch die Episode bzw. Filmteil katalogisiert. Die letzte Eigenschaft die Video besitzt ist »seriesName«. Bei diesen Attribut handelt es sich um den Namen einer Filmreihe oder der Serie. Beispiele können hierfür: Harry Potter, Dark oder Herr der Ringe sein.

Die letzte Basis-Entität ist FilmRelatedPerson. Diese Entität ist nötig, um in Filmen mitwirkende Personen abzubilden. Dies können beispielsweise Schauspieler*innen, Autor*innen oder Regisseur*innen sein. Der Vor- und Nachname bildet für diese Entität den Primär-Schlüssel als Kombination. Außerdem besitzt diese Entität zwei weitere Eigenschaften wie: »dateOfBirth« sowie »sex«. In »dateOfBirth« wird das Geburtsdatum der Person gespeichert wie auch das Geschlecht in »sex«.

Nachdem der Aufbau der Entitäten im Allgemeinen beschrieben wurde, werden nun die Beziehungen untereinander genauer erläutert. Wie aus der Abbildung 3.1 bereits zu sehen ist, gehen zwei Beziehungen hervor. Die erste, welche zwischen der Entität User und Video zu sehen ist, ist eine viele-zu-viele Beziehung. Dabei entsteht ein Beziehungsattribut namens »rating«, welches für die bewerteten Videos von User*innen steht. Es können somit unendlich viele User, unendlich viele Videos bewerten, jedoch jedes Video nur einmal. Das besagt auch die Kardinalität mit dem Stern in der Abbildung.

Die zweite Beziehung besteht zwischen FilmRelatedPerson und Video. Da eine mitwirkende Person im Film mit der Entität Video verlinkt werden kann, ist dies mittels einer „Komposition“ gekennzeichnet. Diese geht aus der Anforderung des Projektes hervor. Wenn eine Person gelöscht wird, welche mit einen Film verlinkt ist, so soll die Person, als auch die zugehörigen Filme gelöscht werden. Beide Entitäten besitzen die Kardinalität mit einem Stern. Auch aus dieser Beziehung geht ein weiteres Beziehungsattribut hervor: »role«. In role werden die Rollen als Liste gespeichert, wobei Beispiele bereits angeführt wurden. Auch hier kann es nur einen Eintrag pro mitwirkende Person geben, dabei können Personen auch ohne jegliche Rolle mit dem Video verlinkt werden, wenn sich der Benutzer zum Beispiel nicht um die genaue Rolle einer Person im Film sicher ist.

4. Logische Datenmodellierung

Robin Gerstmann

Nachdem das Konzept und Design bereits erörtert wurden, wird in diesem Kapitel das konzeptuelle Datenmodell ins logische überführt. Das folgende Relationenmodell besitzt somit den Aufbau:

User(name, age)
Rating(**name, title, releaseYear**, rating)
Video(**title, releaseYear**, minAge, duration, episodeNr, seasonNr, seriesName)
Genre(**title, releaseYear,** genre)
Acts(**title, releaseYear, surname, forname**)
Role(**title, releaseYear, surname, forname,** role)
FilmRelatedPerson(surname, forname, dateOfBirth, sex)

Abb. 4.1.: Relationenmodeel des Film-Managers

Die Abbildung 4.1 verdeutlicht die benötigten Tabellen sowie die Primär-Schlüssel sowie Fremd-Schlüssel. Die Relation »Rating« definiert ihren Primär-Schlüssel durch Fremd-Schlüssel anderer Relationen. Bei der Relation »Acts« werden sogar alle Fremd-Schlüssel genutzt, um den Primär-Schlüssel darzustellen. Die Entitäten wurde aus dem ERM ins Relationenmodell (RM) überführt. »Role« und »Genre« sind hierbei als Relation aufgeführt, da diese eine Liste darstellen. In Kapitel 5 wird darauf genauer eingegangen.

Die Relationen besitzen später in der Datenbank auch Constrains, um die Daten valide zu speichern. Somit muss bei der Tabelle »Rating« neben den Schlüsseln, ein Rating-Wert zwischen 1 - 5 abgegeben werden. Ein leeres Rating wäre nicht von Nutzen und bläht die Datenbank auf. In der Relation »Video« ist es nur die Staffel-Nummer, welche ein `NULL` enthalten darf. Da dies wie bereits erwähnt zur Unterscheidung von Film und Serie genutzt wird. Bei der Relation Video kann das »minAge« nur aus einer vordefinierten Menge: {0, 6, 12, 16, 18} annehmen. Neben den gegebenen Constrains, sind auch Trigger notwendig zu definieren. Einer davon ist notwendig, um den Film und vorhandene Rollen einer mitwirkenden Person zu löschen. Die Bedingung ist, sobald eine Person gelöscht wird, auch dessen mitwirkenden Rollen und Filme zu löschen. Um das realisieren zu können, wurde hierfür ein Trigger in der Datenbank genutzt.

5. Physische Implementation

Axel Obrikat

Der Beginn der Implementierungsphase wird durch die Erstellung eines Frameworks eingeleitet. Die Ergebnisse der Entwurfsphase werden in Code umgesetzt sowie in einem Feedback-Prozess detaillierter ausgeführt oder gänzlich abgeändert. [1]

Ein solches Gerüst soll alle Tabellen, Funktionen etc. in der Datenbank erstellen. Dies wird mit Hilfe einer einzigen Initialisierungsdatei (`ini_script.sql`) umgesetzt, sodass die gesamte Datenbank aufgesetzt und anschließend mit Testdaten gespeist wird. Zur direkten Überprüfung der korrekten Ausführung von Funktionen und Abhängigkeiten enthält das Skript weiterhin einige Testabfragen. So können die erstellten Funktionen und Tabellen bei unerwartetem Output oder beim Auftreten von Fehlern direkt angepasst und verbessert werden. Das Skript sieht also eine klare Dreiteilung vor:

1. Erstellung der Tabellen, Funktionen und Abhängigkeiten
2. Einfügen von Testdaten
3. Testen der Funktionalitäten

Weiterhin werden alle Funktionen und Tabellen in einem initialen Schritt gelöscht, sodass es keine Probleme mit identischen Tabellennamen in der Datenbank oder beim Ändern von Eingabe- sowie Ausgabeparametern von Funktionen gibt.

Der erste Schritt, sowie auch der initiale, können mit Hilfe einer Data Definition Language (DDL) implementiert werden. Die Impelementation der Logik der Funktionen, das Einfügen von Testdaten sowie der letzte Schritt zum Programmieren von Tests wird durch eine Data Manipulation Language (DML) realisiert. Structured Query Language (SQL) ist eine Programmiersprache von relationalen Datenbanken, die beide Eigenschaften vereinigt.

Erstellen der Tabellen

Zum Erstellen der Tabellen werden die Relationen aus dem logischen Datenbankdesign umgesetzt. PostgreSQL erlaubt den Einbezug objektorientierter Aspekte. Die im Relationenmodell (RM) als separate Tabellen dargestellten `Role` und `Genre` werden in der Datenbank als Arrays und nicht als separate Tabellen implementiert. Diese Handlung verletzt die Bedingung der 1. Normalform bewusst, da nun mehrere Werte pro Zelle gespeichert werden. Die Implementation von Arrays und damit bewussten Verletzung der Normalform hat eindeutig prommaiertechnische Vorteile. Statt dem Erstellen von weiteren Abhängigkeiten und dem späteren Joinen von

Tabellen zum Zugriff auf die Daten können die Informationen so bequem als Textliste `TEXT[]` gespeichert werden. Ebenso werden den anderen Attributen bestimmte Datentypen zugewiesen.

Die Beziehungen zwischen den Tabellen sind im RM durch Primär- und Fremdschlüssel gegeben. Diese Abhängigkeiten werden in SQL durch Constraints implementiert. Mit Hilfe von Kaskadierung wird während der Erstellung von Tabellen ebenso angegeben, dass bestimmte Einträge gelöscht werden sollen, wenn Einträge aus anderen Tabellen, deren Fremdschlüssel darauf verweist, auch gelöscht werden.

Für die Umsetzung der Anforderung, dass Filme gelöscht werden sollen, wenn darin mitwirkende Personen gelöscht werden, kommt es zur Implementation eines Triggers samt zugehöriger Trigger-Funktion. Alle weiteren Funktionalitäten, die in A Anforderungsliste beschrieben sind, werden in SQL Funktionen umgesetzt, welche beim Klick im Frontend aufgerufen werden, die jeweilige Handlung ausführen und als Validierung der Nutzeraktion für den User einen String mit entsprechender Antwort zurückgeben. Dabei unterscheidet die Datenbank bei kritischen Stellen je nach Eingabe durch den Benutzer. Beispielsweise wird der Text «Der Film existiert bereits» zurückgegeben, wenn versehentlich ein existierender Film erneut eingefügt werden soll.

Zum Verständnis des Algorithmus zur Ausgabe von Filmvorschlägen wird dieser im Folgenden separat erklärt.

Algorithmus zur Ausgabe von Filmvorschlägen

Filmvorschläge für Nutzer basieren auf den Genren von bereits bewerteten Filmen. Die Bewertung eines Films mit eins bis fünf Sternen überträgt sich demnach eins zu eins auf dessen zugeordnete Genre. Für die Bewertung noch nicht geschauter Filme und dem Erstellen eines Rankings derer, werden die zugehörigen Genre-Bewertungen dann gemittelt, sodass Filmvorschläge ebenso mit eins bis fünf Sternen geordnet werden. Hat der Nutzer noch keine Filme mit Genren bewertet, die anderen noch nicht geschauten Filmen zugeordnet sind, so können diese Filme für die Ausgabe von Vorschlägen auch nicht berücksichtigt werden. Ebenso erhält ein Nutzer keine Vorschläge, wenn er noch keine oder alle Filme der Datenbank bewertet hat. Ausgegeben werden nur Vorschläge von noch nicht geschauten Filmen. Im Einzelnen wird der Algorithmus dreigeteilt implementiert.

Im ersten Teil wird für den aktuellen Nutzer ermittelt, welche Filme dieser bereits bewertet hat. Je nach zugeordnetem Genre werden damit Rückschlüsse auf die Genre-Präferenz des Nutzers angestellt und über alle Filme mit dem jeweiligen Genre gemittelt.

$$\text{Genre-Bewertung} = \frac{\text{Film-Bewertungen mit jeweiligen Genre}}{\text{Anzahl bewerteter Filme mit jeweiligen Genre}} \tag{5.1}$$

Als zweiter Schritt werden die Genre-Bewertungen in Beziehung mit den Filmen gesetzt. Ergebnis diesen Schritts ist eine Liste aller Filme samt Einschätzung, die

anhand von Filmen mit dem selben Genre generiert wird. Je höher die Einschätzung, desto höher die Beurteilung des Algorithmus, dass der Nutzer diesen Film als nächsten schauen sollte.

$$\text{Einschätzung} = \frac{\text{Genre-Bewertungen der zum Film zugeordneten Genre}}{\text{Anzahl zugeordneter, gemittelter Genre-Bewertungen}} \tag{5.2}$$

Im letzten Schritt muss diese Einschätzung noch um diejenigen Filme gefiltert werden, die der Nutzer bereits gesehen hat, sodass diese in den Filmvorschlägen nicht ausgegeben werden. Die drei Schritte sind im Initialisierungsskript mit Hilfe von drei Funktionen umgesetzt, die die angesprochenen Schritte beinhalten.

Testdaten und -funktionen

Zum Erfüllen der Anforderungen werden am Ende des SQL-Skripts noch Testdaten eingefügt. Ebenso kommt es zum Testen der Funktionalitäten mit Hilfe von kurzen `SELECT` Anweisungen, die nach erfolgreichem Test auskommentiert im Initialisierungsskript beibehalten werden.

6. Entwicklung des Frontends

Robin Gerstmann

Paper-Prototype

Am Anfang des Frontends stand vor allem ein spezieller Punkt im Vordergrund: einfache Bedienbarkeit. Die Abbildungen im Anhang B Bilder zum Paper-Prototype zeigen die ersten Ideen und Wünsche wie das Frontend aussehen soll. Das Anmelde-Fenster sollte eigentlich die vorhandenen User anzeigen und mittels Button den Manager öffnen. Dies wurde aber nach Beginn der Implementierung verworfen, da es einige Probleme gab. Auch wurde die Idee eingebracht eine Registrierung neuer Benutzer mit einzubinden. Diese Feature sind allerdings keine direkten Anforderungen, wodurch das User-Management vorerst sehr einfach gehalten wurde. Die Ideen können zu einem späteren Zeitpunkt aber noch implementiert werden.

Nachdem der User sich angemeldet hat, sollen folgende Funktionen bereitgestellt werden: Film-Manager, Crew-Manager, Film-Bewertungen und Film-Vorschläge. So wurde die Idee eines kleines Button eingebracht, um ein weiteres Fenster zu öffnen und dort den Name der Nutzer*innen zu ändern. Wird einer der 4 Optionen im Manager gedrückt, so öffnet sich ein weiteres Fenster. Je nach Option können Filme angezeigt, Personen angezeigt, durchschnittliche Film-Bewertungen (aller User) oder die Film-Vorschläge betrachtet werden. Das ist die erste Ebene.

Die Fenster Crew-Management und Film-Management besitzen demnach ein Plus-Symbol, um jeweils einen neuen Eintrag tätigen zu können. Nach einem neuen Film-Eintrag sollte sich ein weiteres Fenster öffnen, umso die Personen zu verlinken. Dies wurde im weiteren Verlauf jedoch anderweitig umgesetzt. Des Weiteren kann durch das Klicken auf einen vorhanden Eintrag ein Film oder eine Person bearbeitet und gelöscht werden. Im Film-Bearbeitungsfenster ist es auch möglich, den Film zu bewerten oder die vorhandene Bewertung wieder zu löschen. Ein geplantes (nicht umgesetzt) Feature ist es auch, die Filmbewertungen nach eignen und Bewertungen anderer zu filtern. Dieser Protoype ist somit eine erste Form der Darstellung und Zusammenkunft vieler Ideen. Im Laufe der Zeit kamen somit Ideen und Umsetzung zusammen, die das eigentliche Frontend formen.

Verknüpfung zum Backend herstellen

Damit das Frontend mit der vorhanden Datenbank kommunizieren konnte, mussten einige Voreinstellungen getroffen werden. Zum einem war es wichtig eine VPN-Verbindung mit dem Universitätsnetz der Technischen Universität Chemnitz herzustellen. Des Weiteren wird, wie bereits angesprochen, eine Bibliothek zur Kommunikation (`psycopg2`) benötigt. Eines der größten Herausforderungen entstand aus der Anforderung keine Logik in Frontend zu bringen. Diese Anforderung wurde

auch umgesetzt. Ein Problem war aber, dass Python kein `NULL-Type` kennt und es somit unmittelbar kollidieren würde. Es musste ein mapping zur Transferierung von `NULL` zu `None` erstellt werden. Da die Eingabefelder in PyQT bei einem leeren Feld ein leeren String zurückgeben, führt das zu einem weiteren Problem. Somit musste abgefangen werden, ob zum Beispiel die Staffel-Nummer eingetragen wurde. Wenn es keine gab, gibt PyQt den leeren String zurück. Bevor die Anfrage gesendet wird, wird der Parameter auf `None` gesetzt. `None` ist in Python der gewünschte `NULL-Type` bei Datenbanken. Wird ein leerer String übergeben so sind mehrere Constraints von Datentypen bis `NULL` verletzt.

Erstellung der grafischen Benutzeroberfläche

Bei der Entwicklung der grafischen Oberfläche konnten einige Designelemente aus einem privaten Projekt übernommen werden. Dadurch das PyQt5 einige CSS-Befehle interpretieren kann, sind somit schöne Designs möglich gewesen. Mit den QT-Designer konnten einfache Elemente wie Label, Inputs, Buttons, etc. eingebunden und genutzt werden. Wichtig war es, dass Design aus dem Paper-Prototype zu übernehmen und einfach und intuitiv zu gestalten. Für jedes einzelne Fenster musste somit ein Designelement geschaffen werden. Nach dem die Designelemente erstellt worden, muss ein wenig Interaktion eingebaut werden. Die Designelemente sind hierbei von der eigentlichen Logik wie: Drücken eines Button unberührt. Somit konnten Designs überarbeitet werden ohne die Logik wieder neu implementieren zu müssen: Model-View-Controller. Neben den Design-Klassen existieren die eigentlich logischen Klassen, die mit `*Class.py` enden. Hier sind die Anzeigen des Fensters verändert, auch Elemente wie das Ausführen von Anfragen an die Datenbank. In manchen Umständen, wie das Anklicken vorhandener Personenrollen, ließ sich Logik nicht ganz vermeiden. PyQt handelt event-basierenden. Wird ein Eintrag in der Liste geklickt, kann es sich um eine Person oder dessen Rolle handeln. Zum Bearbeiten von Rollen einer Person, benötigt die Datenbank aber alle änderbaren Rollen, sodass nur der Personenname klickbar sein darf und nicht einzelne Rollen. Um das fehlerfrei zu garantieren wurde hier eine Liste genutzt, da sonst das Programm eine Exception werfen würde und unbrauchbar wird.

Auch das übergeben von Parametern von bereits geschlossenen Fenstern war eine Herausforderung. Wurde beispielsweise eine Rolle in einem Film hinzugefügt, so zeigt ein weiteres Fenster alle nicht vorhanden Personen an. Mittels Doppel-Klick sollte die Person ausgewählt und im vorherigen Fenster eingetragen werden. Durch das Setzen einer Klassen-Variable und vorherigen Speichern des Wertes in dieser, konnte das Problem umgangen werden, sodass die ausgewählte Person auch im richtigen Feld des anderen Fensters angezeigt wird.

7. Mögliche Umsetzung als Verteilte Datenbank

Axel Obrikat

Dieser Abschnitt bespricht eine mögliche Umsetzung der DB als Verteilte Datenbank (DDB). Eine Distributed Database (DDB) speichert Daten an verschiedenen Standorten. Die Verarbeitung erfolgt über Datenbankknoten, welche miteinander logisch verbunden sind und von einem Distributed Database Management System (DDBMS) verwaltet werden. Dadurch werden große, unübersichtliche Aufgaben in kleinere aufgespalten, die leichter zu verarbeiten sind.

Eine solche Verteilung wirkt sich positiv auf die Datenstruktur und -autonomie aus. Daraus entstehen ebenso Vorteile in Performance, Ausfallsicherheit und Reliabilität der Daten. Auf der anderen Seite führt eine DDB zur Erhöhung von Kosten und Komplexität. Ebenso kann die Implementation einer DDB schneller zu Sicherheitslücken führen. Dabei gibt es verschiedene Typen von DDB.

Für die Umgestaltung des den Filme-Managers zu einer DDB, bietet sich in Hinblick auf die Stufe der Homogenität eine homogene DDB an. Anders als bei der heterogenen Art benutzen alle DBMS die gleiche Software, was den weiteren Verlauf der Erstellung der DDB erleichtert.

In Sachen Distribution und Autonomie wird ein Szenario angenommen, dass der Filme-Manager weltweit angeboten werden soll. Für jede Region (zum Beispiel pro Kontinent) sollte demnach ein separater Datenbankknoten vorhanden sein, der größtenteils autonom arbeiten kann. Ebenso sollte eine Netzwerk-Partitionierung mit gewisser Partitionstoleranz vorhanden sein. Dies gewährleistet beim Ausfall eines Knotens, dass andere Knoten relativ ungestört weiterlaufen können. Diese Verteilung begründet sich darin, da Nutzer aus Europa, nicht auf die Inhalte von Nutzern aus Amerika für die Ausgabe von Filmvorschlägen oder der durchschnittlichen Bewertung von Filmen zugreifen müssen. Die Vergleiche geschehen innerhalb der Region. Pro Kontinent sollten genügend Vergleichsdaten vorhanden sein

Möchte ein Nutzer umziehen, so sollte ein übergeordnetes System, welches Zugriff auf alle Knoten besitzt, den jeweiligen Benutzer samt abgegebener Bewertungen in den Knoten umlagern. Um eine fehlerfreie Umlagerung zu garantieren, müssen in einem vorangestellten Schritt alle Nutzer der gesamten DDB mit einer internen, einzigartigen ID versehen werden, sodass es zu keinen Redundanzen bei den Benutzernamen kommen kann. Weiterhin muss das übergeordnete System dafür sorgen, dass bewertete Filme sowie darin mitwirkende Personen aus DB A ebenso in DB B vorhanden sind und somit bei Umlagerung, falls nötig, in DB B kopiert werden.

8. Fazit

Robin Gerstmann

Das Projekt einen Film-Manager zu entwickeln brachte viele ungesehene Herausforderungen mit sich. Das anfangs erwähnte Design konnte nicht zu 100% umgesetzt werden. Auch waren mögliche Zusatzfeatures, die nicht in den Anforderungen aufgeführt sind, wie einen Nutzer zu registrieren oder Film-Bewertungen zu filtern, nicht ohne Weiteres realisierbar. Die Zeit für das Projekt war angemessen, jedoch nicht ausreichend um solche zusätzlich Features einzubinden und umfangreich zu testen. Durch mehrere Meetings in der Woche, konnten jedoch alle auffindbaren Fehler schnell und einfach gelöst werden. Wurde ein Fehler gefunden, wurde sofort ein Feedback gegeben, umso Front- und Backend mehr und mehr zu vereinen. Im Backend gab es in der ersten Phase die Idee, Video in Film und Serie aufzuteilen. Somit das Video die Mutterklasse ist und beide Relationen von ihr erben. Leider kam das schnell zu Problemen bei der Implementierung der Relation »rating« und «acts». Demnach werden nämlich Film und Serie in verschiedenen Tabellen gespeichert, was folglich die Relation »rating« unbrauchbar machte. Es kam wieder und wieder der Fehler, dass eine Serie und Film nicht gefunden wurde. Deshalb wurde dann entschieden, das Design der Datenbank anderweitig aufzusetzen und es mit einer Relation zusammenzufassen sowie nur ein weiteres Attribut hinzuzufügen, um von Serie oder Filmreihe zu unterscheiden. Ein weiterer interessanter Fehler war, dass die SQL-Datenbank nicht zulässt, eine Tabelle Namens »user« zu erstellen. Es wurden einige Versuche angestellt, bis herausgefunden wurde, das wohl dieser Name auf den Index für Relation-Namen steht.

Trotz dieser Herausforderungen wurde das Projekt erfolgreich realisiert und alle gegeben Anforderungen erfüllt. Das Backend und Frontend kommunizieren miteinander und sollte bei richtiger Konfiguration keine Laufzeitfehler verursachen. Dennoch kann das Projekt ausgiebig erweitert werden. Eine Erweiterung könnte das Filtern von Serien und Filmen sein, das bereits angesprochen wurde. Gleichzeitig könnten Nutzer*innen statt Namen auch Passwörter nutzen, um sich zu authentifizieren. Schließlich wurde der Aspekt Sicherheit und Datenschutz in diesen Projekt nicht betrachtet. Eine umfangreiche Aufgabe könnte sein, Filmen und Usern ein Bild zu geben oder eine Kommentarfunktion einzubinden. Somit können andere User*innen nicht nur bewerten, sondern auch kommentieren, was ihnen an den Film oder Serie gefallen hat und was nicht. Die Ideen und Grenzen für das Projekt sind somit die User und Benutzer selbst gekoppelt. Alles in allem bildet der prototypische Film-Manager den Usern aber einen guten Überblick über alle geschauten Filme, egal auf welcher Plattform diese geschaut wurden und enthält ebenso weitere Features wie dem Vorschlagen von noch nicht geschauten/bewerteten Filmen.

Literaturverzeichnis

[1] Aichele, C., Schönberger, M.: App-Entwicklung – effizient und erfolgreich: Eine kompakte Darstellung von Konzepten, Methoden und Werkzeugen. Springer Fachmedien Wiesbaden, Wiesbaden (2016), https://doi.org/10.1007/978-3-658-13685-7

[2] Barton, T., Müller, C., Seel, C.: Mobile Anwendungen in Unternehmen: Konzepte und betriebliche Einsatzszenarien. Springer Fachmedien Wiesbaden, Wiesbaden (2016), https://doi.org/10.1007/978-3-658-12010-8

[3] Nunes, I.L.: Advances in Human Factors and Systems Interaction. Springer International Publishing, Cham (2020)

A. Anforderungsliste

Zur transparenteren Übersicht ist folgend eine Liste mit den Anforderungen an das Filmverwaltungssystem aufgeführt.

- PostgreSQL-Datenbank
  - Programmlogik direkt in der Datenbank mit Hilfe von PL/pgSQL programmiert
    - Management von Filmen
      - Filme hinzufügen (kann untergeordnet sein [Serien oder Filmreihen] - Titel und Erscheinungsjahr sollten angegeben und einzigartig sein - Filme können ein oder mehr Genres haben - in Filmen mitwirkende Personen müssen mit entpsrechender Rolle verlinkt sein)
      - Überblick über alle Filme in der Datenbank (untergeordnete Filme sollen beim Überblick nicht sofort angezeigt werden)
      - Ändern der Filmattribute
      - Entfernen von Filmen (Untergeordnete Filme sowie dazugehörige Bewertungen sollen auch entfernt werden)
    - Management von im Film mitwirkenden Personen
      - Hinzufügen von im Film mitwirkenden Personen (Name zur Identifikation und eventuell weiterer Attribute)
      - Überblick über alle in Filmen mitwirkenden Personen
      - Ändern der Attribute von im Film mitwirkenden Personen
      - Entfernen von im Film mitwirkenden Personen (wenn im Film mitwirkende Personen entfernt werden, so wird der Film auch entfernt - wenn alle Filme, in der die im Film mitwirkende Person mitgespielt hat, entfernt wurden, bleibt diese Person dennoch bestehen)
    - Management von Filmbewertungen
      - Hinzufügen von Filmbewertungen mit einem einfachen, eigens implementierten Bewertungssystem,
      - Überblick über alle Filmbewertungen
      - Ändern von Filmbewertungen

            - Entfernen von Filmbewertungen
        - Erhalt von Filmvorschlägen für die Benutzer
            - Ein Benutzer, der mindestens einen Film bewertet hat, soll eine Liste mit Filmvorschlägen erhalten, die noch nicht vom selbigen Benutzer bewertet wurden
            - Ein Benutzer, der noch keinen Film bewertet hat, kann auch keine Vorschläge erhalten
            - Ein Benutzer, der bereits alle Filme bewertet hat, kann keine weiteren Vorschläge erhalten
            - Der Algorithmus soll relativ einfach gehalten werden, basierend auf den getätigten Bewertungen
    - Benutzer werden durch einen Benutzernamen identifiziert
        - Name kann leicht geändert werden
        - Benutzer müssen nicht gelöscht oder neu angelegt werden können
    - Die gesamten Benutzer- und Filminformationen, sowie alle weiteren Informationen, werden in der Datenbank gespeichert

- Benutzeroberfläche zur Interaktion mit der Datenbank
    - Für die Erstellung der Benutzeroberfläche gibt es keine Restriktionen (Programmiersprache, Bibliotheken...)
    - Die Benutzeroberfläche sollte so einfach wie möglich gehalten werden und keinerlei Logik enthalten.
    - Die Oberfläche hilft der Visualisierung und Interaktion mit den Daten und bezieht Aspekte der Benutzerfreundlichkeit mit ein

- Testdaten
    - Zehn im Film mitwirkende Personen, welche verschiedene Rollen in den Filmen innehaben, darunter...
        - ...sind mindestens fünf Personen bei mindestens drei Filmen mitwirkend, wobei vermieden werden soll, dass diese Personen bei den gleichen Filmen mitwirken
        - ...ist mindestens eine Person bei einem einzigen Film in mindestens drei Rollen mitwirkend
    - Zehn Filme, die alle zu mindestens einem Genre zugeordnet sind, wobei...
        - ...drei Filme zu einem untergeordnet sind
        - ...fünf Filme, die mindestens drei unterschiedlichen Genren zugeordnet sind, wobei vermieden werden sollte, dass zwei Filme den exakt gleichen Genren zugeordnet worden sind

  - Fünf Benutzer, wobei...
    * ...ein Benutzer keine Bewertung abgegeben hat
    * ...drei Benutzer circa die Hälfte der verfügbaren Filme bewertet haben, die bewerteten Filme sollten unter diesen drei Benutzern aber variieren
    * ...ein Benutzer alle Filme bewertet hat

- Sonstiges
  - Überblick über alle verwendeten Technologien (ausgenommen PostgreSQL and PL/pgSQL) unter Motivationsbeschreibung
  - Visualisierung des Datenbankmodells als ERM in Unified Modelling Language (UML) und als dazugehöriges RM
  - Diskussion unter welchen Bedingungen und welcher Verteilung die Datenbank als Verteilte Datenbank genutzt werden könnte (nicht mehr als drei Möglichkeiten sollten besprochen werden)

B. Bilder zum Paper-Prototype

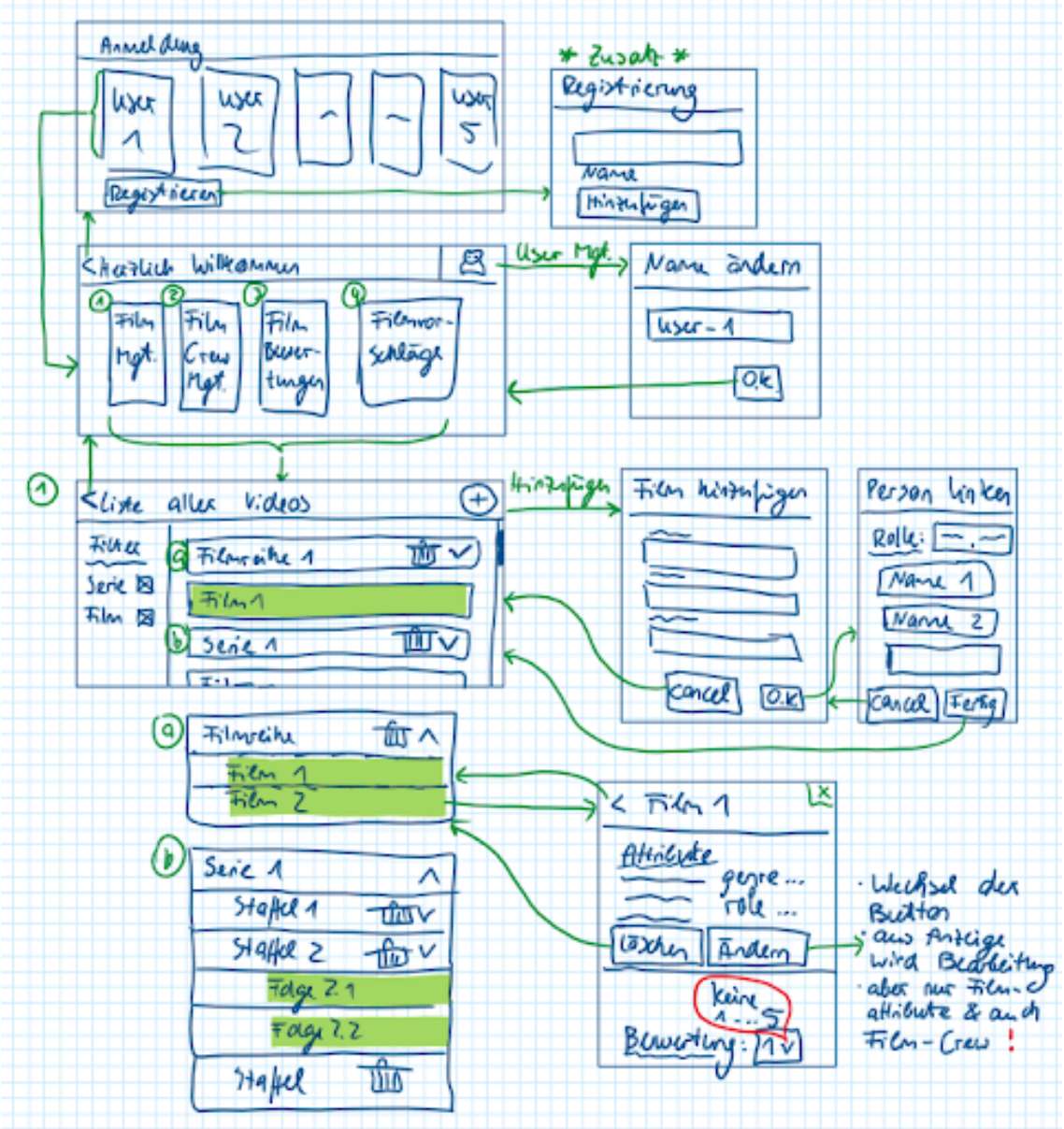

Abb. B.1.: Paper-Prototype Teil 1

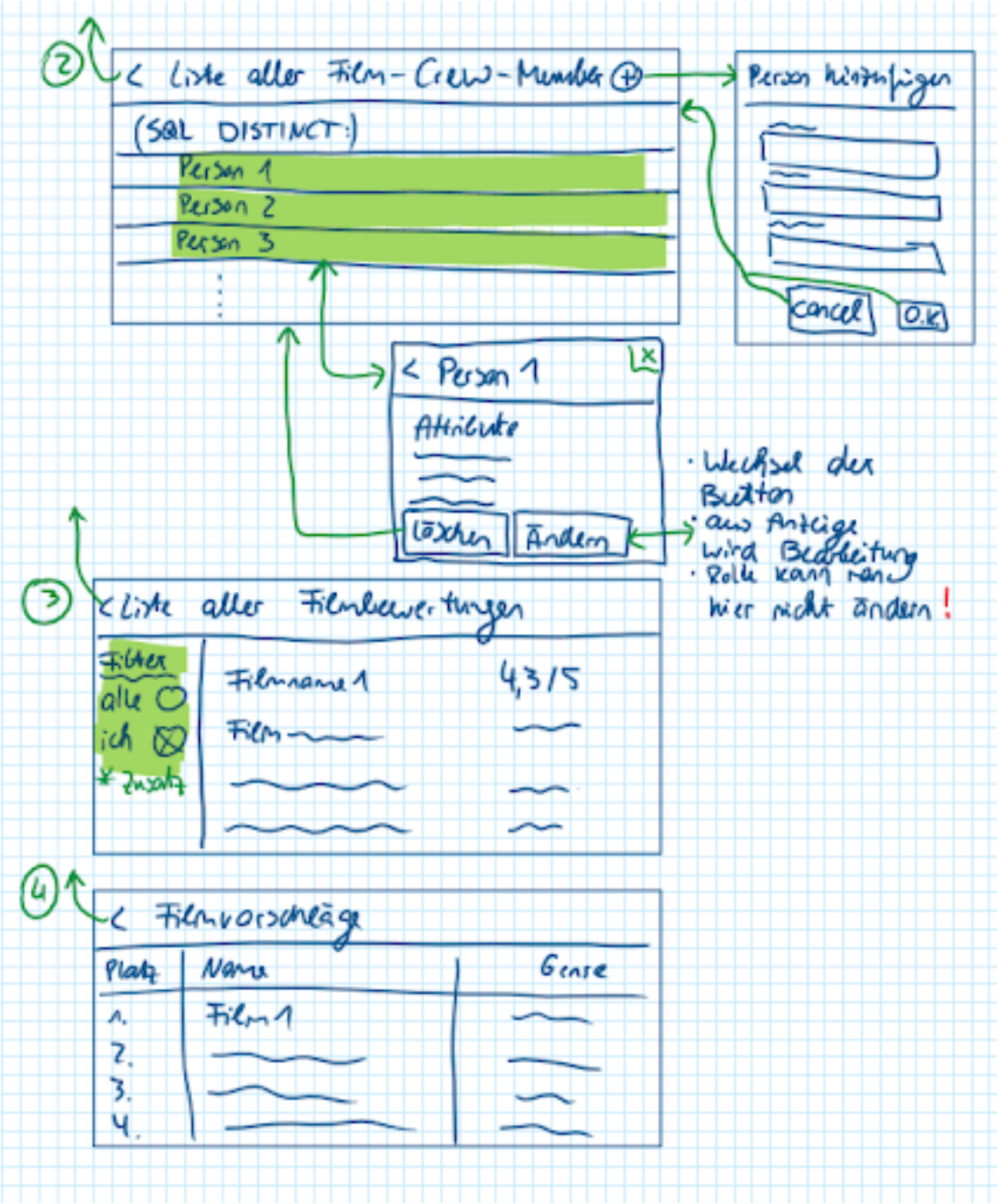

Abb. B.2.: Paper-Prototype Teil 2

Eidesstattliche Erklärung

Ich erkläre ehrenwörtlich,

1. dass ich meine Projektarbeit selbstständig verfasst und ohne andere als die angegebenen Hilfsmittel verfasst habe.
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur, sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.
3. dass ich die Projektarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

09.02.2022, Chemnitz	Robin Gerstmann, Axel Obrikat
Datum, Ort	Verfasser