Recommender-System für Events

Oliver Lindemann
Hochschule Harz
University of Applied Science
u33873, m26264
u33873@hs-harz.de

ABSTRACT
This paper describes the functionality of recommender systems and the different algorithms and techniques that are used by these systems to calculate recommendations. It also explains how such a recommender system is set up.

1. EINLEITUNG
Recommender- bzw. Empfehlungssysteme werden auf immer mehr Webseiten eingesetzt. Für moderne Unternehmen, die mit ihrer Onlinepräsenz ihren Unternehmensprofit erwirtschaften, sind diese Art von Systemen von großer Relevanz um wettbewerbsfähig zu bleiben. In dieser Facharbeit wird erläutert, was Recommender-Systeme sind, welche verschiedenen Techniken bzw. Algorithmen es zur Berechnung von Empfehlungen gibt und wie man ein solches System aufbaut. Dies wird anhand eines Event-Shops veranschaulicht, welches dem Benutzer mittels eines Recommender-Systems Empfehlungen für Events anzeigt.

2. GRUNDLAGEN
Recommender-Systeme haben die grundlegende Aufgabe etwas zu empfehlen (engl.: *to recommend*). Im Folgenden wird deswegen auch von Empfehlungssystemen gesprochen. Diese Empfehlungen haben in einem Online-Shop z.B. den Sinn, dem Kunden relevante Produkte zu präsentieren, die diesen mit hoher Wahrscheinlichkeit interessieren. Bestenfalls kauft der Kunde die empfohlenen Produkte und steigert somit den Umsatz des Unternehmens. Diese Art der personalisierten Empfehlungen für Kunden wird durch Recommender-Systeme realisiert.

Hinter Empfehlungssystemen verbergen sich verschiedenste Berechnungen aufgrund von Algorithmen und mathematischen Formeln, um einem Nutzer Produkte zu empfehlen. Diese Empfehlungen werden durch verschiedene Faktoren beeinflusst, wie z.B. durch bereits gekaufte oder bewertete Produkte. Die Techniken zur Berechnung von Empfehlungen lassen sich in fünf Arten [1] unterteilen: *Content-based*, *Knowledge-based*, *Demographic*, *Collaborative* und *Utility-based Filtering*. Dazu kann ebenfalls noch die *Hybrid Filtering* [2] Technik gezählt werden, welche eine Kombination aus den vorhergehenden Filtertechniken darstellt. In dieser Facharbeit werden allerdings nur die ersten vier Arten inkl. Hybrid Filtering verwendet und näher erläutert.

Modul:	Anpassungsfähige Systeme
Prüfer:	Prof. Dr. Kerstin Schneider
Abgabedatum:	28.02.2020

3. EMPFEHLUNGSTECHNIKEN
Alle Filtertechniken ermitteln auf unterschiedliche Weise Produkte, die dem Benutzer empfohlen werden können. Allerdings werden nicht alle errechneten Produktempfehlungen angezeigt, sondern nur solche, die der Benutzer noch nicht selbst gekauft oder bewertet hat.

3.1 Content-based Filtering
Diese Filtertechnik ermittelt basierend auf der vom Nutzer abgegebenen Bewertungen Produkte, die zu dem Benutzer passend erscheinen. Je mehr der Benutzer bewertet, desto mehr und bessere Empfehlungen können berechnet werden.

3.2 Knowledge-based Filtering
Knowledge-based Filtering funktioniert über die vom Benutzer zuvor angegebenen Präferenzen. Diese Präferenzen können z.B. bei Events gewisse Genres sein, die der Benutzer bevorzugt. Die Knowledge-based Filtertechnik ermittelt daraufhin Produkte bzw. Events, die zu den von dem Benutzer angegebenen Präferenzen passen, also dem gleichen Genre angehören.

3.3 Demographic Filtering
Demographic Filtering basiert auf den persönlichen Daten und Eigenschaften des Benutzers, wie z.B. dem Alter, Geschlecht oder Wohnort. Diese Eigenschaften werden auch als Kategorien bezeichnet. Mittels dieser Kategorien werden andere User ermittelt, die das gleiche Alter, Geschlecht oder Wohnort haben.

3.4 Collaborative Filtering
Das Funktionsprinzip von Collaborative Filtering beruht darauf, dem Benutzer ähnliche User zu finden, deren gekauften oder bewerteten Produkte einzustufen und dem Benutzer zu empfehlen. Ähnliche User werden gefunden, indem Bewertungen aggregiert werden und so eine Ähnlichkeit zwischen zwei Benutzern ermittelt werden kann.

3.5 Hybrid Filtering
Hybrid Filtering ist eine Kombination aus den oben beschriebenen Filtertechniken, wobei nicht zwingend alle Techniken verwendet werden müssen. Die unterschiedlichen Filtertechniken benötigen unterschiedliche Daten, und nicht immer stehen dem System alle notwendigen Daten zur Verfügung. Beispielsweise hat ein Nutzer bei einer Neuanmeldung noch keine Produkte bewertet. Dies schließt das Content-based Filtering aus. Stattdessen wird z.B. Demographic-Filtering herangezogen, um dem Benutzer passende Produkte zu empfehlen. Die Auswahl der Filtertechnik wird daher immer den aktuell dem System verfügbaren Daten angepasst.

4. EMPFEHLUNGSSYSTEM FÜR EVENTS

Das hier vorgestellte Empfehlungssystem handelt von einem Event-Shop, der dem Benutzer mittels verschiedener Filtertechniken bestimmte Events empfiehlt.

4.1 Aufbau

Das Empfehlungssystem für Events besteht aus einem Webserver, auf dem die Webapplikation läuft und einer Datenbank, an der die Webapplikation angebunden ist. Die Applikation ist in mehrere Komponenten aufgeteilt.

4.1.1 JSP

JSPs stellen die grobe Struktur einer Webseite dar. Sie beinhalten die Struktur und das Aussehen der angezeigten Webseite. Mittels Platzhaltern kann die JSP beim Aufruf nachträglich Informationen von der Webapplikation erfragen und diese in die Seite einbinden.

4.1.2 Servlets

Servlets sind die ‚Verwalter' der einzelnen Webseiten. Sie registrieren sich auf eine bestimmte URL, wie z.B. */Login* und erhalten damit alle Anfragen von Benutzern, die diese URL aufrufen. Die Servlets verarbeiten diese Anfragen und leiten die Benutzer auf eine entsprechende JSP weiter, die dem Anwender unter der aufgerufenen URL dargestellt wird

4.1.3 Beans

Beans sind Abbildungen von Objekten [3]. Sie stellen z.B. ein Objekt aus der Datenbank dar und beinhalten alle dessen Eigenschaften. Beans beinhalten keine komplexe Logik, sondern werden allein zur Speicherung von Daten genutzt.

4.2 Benutzerinteraktion

Der Benutzer greift über einen Webbrowser auf dieses System zu. Die Interaktion zwischen Benutzer und Empfehlungssystem läuft grundsätzlich nach folgendem Schema ab:

1. Der Benutzer ruft eine URL auf: *localhost/.../**Webseite***
2. Auf diese URL registriertes Servlet bearbeitet Anfrage und leitet den Benutzer an die ***Webseite**.jsp* weiter
3. **Webseite**.jsp wird Benutzer unter aufgerufener URL angezeigt (*/.../**Webseite***), Pfad zu *jsp* bleibt versteckt.

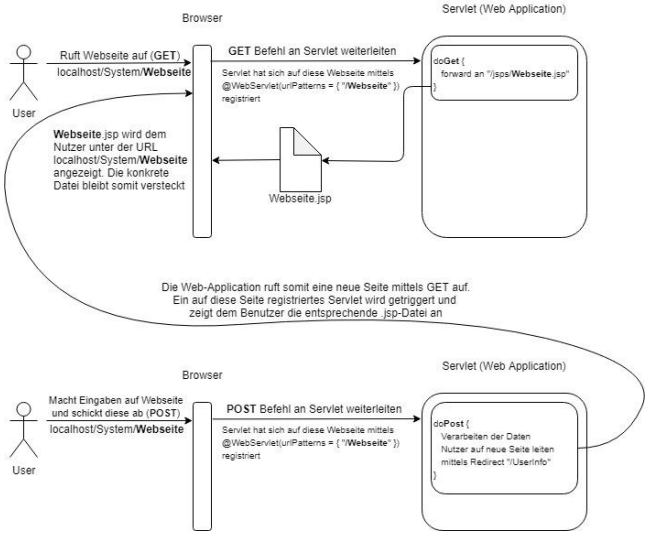

Abbildung 1 Zusammenhang Website - Servlet

4.3 Verwendete Filtertechnik

Für die Empfehlungen von Events wird in diesem System ein Hybrid-Filtering Ansatz eingesetzt. Dieser vereint die Filtertechniken Collaborative / Content-based, Knowledge-based und Demographic Filtering.

Die Berechnung der Empfehlungen geschieht grundsätzlich über Collaborative Filtering. Allerdings kann es vorkommen, dass der Benutzer ein sog. *Grey Sheep* ist. Ein *Grey Sheep* ist ein Benutzer, zu dem keine ähnlichen Benutzer gefunden werden konnten. In diesem Fall wird auf Content-/Knowledge-based und Demographic Filtering zugrückgegriffen. Dies geschieht ebenso, falls nicht ausreichend ähnliche Benutzer bzw. Empfehlungen über Collaborative Filtering ermittelt werden konnten.

Eine weitere Ausnahme besteht bei einem sog. *Cold Start*. Ein *Cold Start* bezeichnet den Zustand, wenn über ein Objekt noch keine Informationen vorliegen. Dies kann z.B. die Neuregistrierung eines Benutzers im Event-Shop sein. Über diesen neuen Benutzer existieren noch keine Informationen wie Käufe oder Bewertungen. Falls der Benutzer allerdings bei der Registrierung seine Präferenzen angegeben hat, wird nun mittels Knowledge-based Filtering nach Empfehlungen gesucht. Sollte der Benutzer keine Präferenzen ausgewählt haben, wird stattdessen Demographic Filtering angewandt. Tabelle 1 Eingesetzte Filtertechniken stellt die verwendeten Filtertechniken noch einmal in Relation zu den vorliegenden Daten und zeigt auf, wann welche Filtertechnik zur Berechnung von Empfehlungen eingesetzt wird.

Tabelle 1 Eingesetzte Filtertechniken

Ausreichend ähnlicher Benutzer	Präferenzen angegeben	Filtertechniken		
		CF/CN	KB	DM
Ja	Ja	**X**	**~X**	**~X**
Ja	Nein	**X**	**–**	**~X**
Nein	Ja	**–**	**X**	**X**
Nein	Nein	**–**	**–**	**X**

(CF = collaborative, CN = content-based, KB = Knowledge-based, DM = demographic)

X	Wird angewandt
~X	wird bei zu wenigen Ergebnissen zusätzlich angewandt
–	Wird nicht angewandt

4.4 Bestimmung von ähnlichen Benutzern

Beim Collaborative Filtering existieren verschiedene Algorithmen zur Bestimmung von ähnlichen Benutzern. Einer hiervon ist der Pearson-Spearman-Algorithmus. Dieser betrachtet die Bewertungen der Benutzer als Vektoren und ermittelt anhand der Formel 1 Ähnlichkeitsbestimmung nach Pearson/Spearman die Ähnlichkeit zwischen dem Benutzer $\vec{b}$ und den anderen Usern $\vec{u}$.

$$\text{Ähnlichkeit}\,(\mathbf{B},\mathbf{U}) = \frac{\left(\vec{b}-\bar{\vec{b}}\right) * \left(\vec{u}-\bar{\vec{u}}\right)}{\left|\left(\vec{b}-\bar{\vec{b}}\right)\right| * \left|\left(\vec{u}-\bar{\vec{u}}\right)\right|}$$

Formel 1 Ähnlichkeitsbestimmung nach Pearson/Spearman

Als Ergebnis dieser Formel erhält man einen Zahlenwert, der kleiner oder gleich *1* ist. Dieser Wert gibt die Ähnlichkeit zwischen dem Benutzer und einem anderen User an. Da jeder User, der das gleiche Produkt wie der Benutzer bewertet hat, eine gewisse

Ähnlichkeit zu dem Nutzer hat, wird diese berechnete Ähnlichkeit im nächsten Schritt noch weiter verwertet. Mittels eines Schwellenwertes werden nun die Ähnlichkeiten gefiltert. Ein sog. *Similarity Threshold* gibt an, ab welchem errechneten Ergebnis der User dem Benutzer tatsächlich ausreichend ähnlich ist. In großen Recommender-Systemen wird ein relativ hoher Threshold von >95%, also 0,95 gewählt. In kleineren Systemen mit weniger Daten, wie es in dem Event-Shop-Empfehlungssystem der Fall ist, wird ein niedriger Wert von >20% (0,2) verwendet, damit noch ausreichend viele Ergebnisse und somit ähnliche Nutzer berechnet werden können. Andernfalls würde es in kleinen Systemen keine zueinander ähnlichen Benutzer geben.

4.4.1 Umsetzung in SQL

4.4.1.1 Collaborative / Content-based Filtering

Der oben vorgestellte Algorithmus zur Bestimmung von ähnlichen Usern muss im Empfehlungssystem noch realisiert werden. Dies geschieht im Event-Shop über eine in Abbildung 2 Pearson/Spearman-Algorithmus in SQL[1] dargestellte SQL-Anfrage an die Datenbank. Diese Anfrage bildet die Pearson-Spearman-Formel ab. Die ersten vier ? werden durch die ID des aktuellen Benutzers ersetzt; das letzte ? wird durch einen ausgewählten Threshold ersetzt, um nur Ergebnisse größer oder gleich dem Schwellwert zu erhalten.

In den meisten Fällen liefert diese Berechnung allerdings nicht genügend ähnliche User. Aus diesem Grund wird die SQL-Anfrage erneut ausgeführt. Dieses Mal allerdings nicht mit dem Benutzer, sondern mit den gefundenen ähnlichen Usern des Nutzers. Daraufhin erhält man zu den ursprünglich gefundenen Usern wiederum ähnliche User. Dies wird wiederholt, bis eine festgesetzte Anzahl an ähnlichen Usern gefunden wurde oder es keine weiteren ähnlichen User mehr gibt.

Von all diesen erhaltenen Usern werden nun die bewerteten Produkte bzw. Events ermittelt. Diese werden nach einem Mindestrating gefiltert, damit keine schlecht bewerteten Produkte dem Benutzer empfohlen werden. Abschließend werden noch alle Events herausgenommen, die der Benutzer bereits bewertet hat.

```
SELECT DISTINCT title, typename
FROM (
    SELECT e.title, e.description, e.typename,
           round(avg(r.rating), 2) schnitt
    FROM RATINGS r, EVENTS e
    WHERE
        r.eventid = e.eventid
        AND userid IN (%s)
        AND rating > %.2f
    GROUP BY e.title, e.description, e.typename
    ORDER BY schnitt DESC
    )
WHERE description NOT IN (
    SELECT e.description
    FROM EVENTS e, RATINGS r
    WHERE r.userid = %s
    AND e.eventid = r.eventid
    )
FETCH NEXT %d ROWS ONLY;
```

Abbildung 3 Events ähnlicher User für Empfehlung filtern

```
SELECT userid, Aehnlichkeit
FROM (
    SELECT zaehler.userid,
    (zaehler.skalarProdukt / nenner.prodRating) Aehnlichkeit
    FROM
        (SELECT userid, SUM(prodRating) skalarProdukt
        FROM (
            SELECT andere.userid, nutzer.eventid,
            (nutzer.ratingPS * andere.ratingPS) prodRating
            FROM
                (
                SELECT userid, eventid,
                (rating - AVGRating) ratingPS
                FROM (
                    SELECT userid, eventid, rating,
                    AVG(rating) over
                    (PARTITION BY userid) AVGRating
                    FROM RATINGS
                    )
                WHERE userid = ?
                ) nutzer
            ,
                (
                SELECT userid, eventid,
                (rating - AVGRating) ratingPS
                FROM (
                    SELECT userid, eventid, rating,
                    AVG(rating) over
                    (PARTITION BY userid) AVGRating
                    FROM RATINGS
                    )
                WHERE userid <> ?
                ) andere
            WHERE nutzer.eventid = andere.eventid
            )
        GROUP BY userid
        ) zaehler
    ,
        (
        SELECT andere.userid,
        nutzer.ratingPS * andere.ratingPS prodRating
        FROM
            (
            SELECT userid, SQRT(SUM(ratingPS)) ratingPS
            FROM (
                SELECT userid, eventid,
                POWER(rating - AVGRating, 2) ratingPS
                FROM (
                    SELECT userid, eventid, rating,
                    AVG(rating) over
                    (PARTITION BY userid) AVGRating
                    FROM RATINGS
                    )
                WHERE userid = ?
                )
            GROUP BY userid
            ) nutzer
        ,
            (
            SELECT userid, SQRT(SUM(ratingPS)) ratingPS
            FROM (
                SELECT userid, eventid,
                POWER(rating - AVGRating, 2) ratingPS
                FROM (
                    SELECT userid, eventid, rating,
                    AVG(rating) over
                    (PARTITION BY userid) AVGRating
                    FROM RATINGS
                    )
                WHERE userid <> ?
                )
            GROUP BY userid
            ) andere
        ) nenner
    WHERE zaehler.userid = nenner.userid
)
WHERE aehnlichkeit > ?;
```

Abbildung 2 Pearson/Spearman-Algorithmus in SQL

[1] Optimierungsmöglichkeiten im Abschnitt AUSBLICK erläutert.

4.4.1.2 *Knowledge-based / Demographic Filtering*

Im Falle eines Grey Sheeps, Cold Starts oder bei nicht ausreichend ähnlichen Benutzern bzw. Empfehlungen durch Collaborative Filtering werden Knowledge-based und Demographic Filtering herangezogen. Dies erfolgt im Event-Shop-Empfehlungssystem in drei Abstufungen.

Die erste Abstufung und somit auch die strengste filtert nach den vom Benutzer angegebenen Präferenzen (Genres), dem gleichen Geschlecht und einem ähnlichen Alter[2]:

```
SELECT eventID, title
FROM events
WHERE title IN
    (SELECT DISTINCT e.title
    FROM RATINGS r, EVENTS e,
        (SELECT c.userid userlist
        FROM customer c,
            (SELECT GENDER_SHORT gender, BIRTHDAY_YEAR birthday
            FROM CUSTOMER
            WHERE userid = ?) info
        WHERE c.userid <> ?
        AND c.GENDER_SHORT = info.gender
        AND c.BIRTHDAY_YEAR < (info.birthday + 5)
        AND c.BIRTHDAY_YEAR > (info.birthday - 5)
        ) ageGender
    WHERE r.RATING >= ?
    AND r.eventid = e.eventid
    AND r.userid IN ageGender.userlist
    AND e.genre IN
        (SELECT DISTINCT genre
        FROM CUSTOMER, EVENTS
        WHERE userid = ?
        AND(preference1 = genre
            OR preference2 = genre
            OR preference3 = genre)
        )
    );
```

Abbildung 4 KB & DF - Präferenzen, Alter, Geschlecht

Liefert die erste und strengste Stufe des Knowledge-based und Demographic Filtering keine ausreichende Menge an Empfehlungen, wird die zweite und mittlere Stufe angewandt. Diese filtert nur noch nach den vom Benutzer angegebenen Präferenzen und ähnlichem Alter:

```
SELECT DISTINCT e.title
FROM RATINGS r, EVENTS e,
    (SELECT c.userid userlist
    FROM customer c,
        (SELECT BIRTHDAY_YEAR birthday
        FROM CUSTOMER
        WHERE userid = ?) info
    WHERE c.userid <> ?
    AND c.BIRTHDAY_YEAR < (info.birthday + 5)
    AND c.BIRTHDAY_YEAR > (info.birthday - 5)
    ) age
WHERE r.RATING >= ?
AND r.userid IN age.userlist
AND r.eventid = e.eventid
and e.genre IN
        (SELECT DISTINCT genre
        FROM CUSTOMER, EVENTS
        WHERE userid = ?
        AND(
            preference1 = genre
            OR preference2 = genre
            OR preference3 = genre
            )
        );
```

Abbildung 5 KB & DF - Präferenzen, Alter

Sollte die mittlere Stufe auch keine ausreichende Anzahl an Empfehlungen liefern, wird die letzte und am wenigsten strenge Filtermethode eingesetzt, die lediglich nach einem demographischen Aspekt filtert, dem ähnlichen Alter. Dies geschieht auch, falls der Benutzer neu registriert ist und keine Präferenzen angegeben hat.

```
SELECT DISTINCT e.title
FROM RATINGS r, EVENTS e,
    (SELECT c.userid userlist
    FROM customer c,
        (SELECT BIRTHDAY_YEAR birthday
        FROM CUSTOMER
        WHERE userid = ?) info
    WHERE c.userid <> ?
    AND c.BIRTHDAY_YEAR < (info.birthday + 5)
    AND c.BIRTHDAY_YEAR > (info.birthday - 5)
    ) age
WHERE r.RATING >= ?
AND r.userid IN age.userlist
AND r.eventid = e.eventid;
```

Abbildung 6 DF - Alter

Für den unwahrscheinlichen Fall, dass keine Filtertechnik erfolgreich ist und keine Empfehlungen berechnet werden konnten, werden dem Benutzer alle verfügbaren Events angezeigt.

4.5 Servlets & JSPs

Das Event-Shop-Recommender-System bietet verschiedene Seiten, auf der der Benutzer z.B. sich einloggen und seine Event-Empfehlungen, Käufe oder Statistiken einsehen kann. Diese Seiten werden jeweils durch ein eigenes Servlet kontrolliert, welches im Nachfolgenden kurz erläutert wird.

4.5.1 *LoginServlet*

Das LoginServlet ist bei jedem Besuch des Empfehlungssystems der Einstiegspunkt für den Anwender. Es stellt dem Benutzer das Login.jsp dar, über das er sich mittels Benutzername und Passwort einloggen kann. Falls der Anwender bereits eingeloggt ist und die Login-Seite aufruft, wird dieser automatisch zu seinen Empfehlungen weitergeleitet und muss sich kein zweites Mal anmelden.

4.5.2 *LogoutServlet*

Das LogoutServlet wird beim Anklicken der Schaltfläche *Logout* aufgerufen, welche auf jeder Seite des Empfehlungssystems sichtbar ist. Es meldet den Benutzer von dem Empfehlungssystem ab und leitet ihn zur Login-Seite weiter. Der Anwender kann nun, ohne erneutes Anmelden, nicht mehr auf das Empfehlungssystem zugreifen.

4.5.3 *RegisterServlet*

Das RegisterServlet ist für das Registrieren eines neuen Benutzers verantwortlich. Es stellt dem Benutzer die Register.jsp und Register2.jsp dar. Auf der Seite Register.jsp trägt der Benutzer all seine Daten für die Registrierung ein. Wenn alle Daten korrekt sind, wird der Anwender auf Register2.jsp weitergeleitet. Hier kann er nun eine Auswahl seiner Präferenzen tätigen. Danach ist die Registrierung abgeschlossen. Alle Eingaben und Aktionen, die in den beiden JSPs getätigt werden, verwaltet das RegisterServlet.

4.5.4 *RecommenderServlet*

Das RecommenderServlet ist für die Darstellung der Empfehlungen eines Benutzers verantwortlich. Es ermittelt über die bereits vorgestellten Empfehlungsalgorithmen die Empfehlungen des Nutzers und stellt diese nach Genre gruppiert dar. Bei Auswahl eines

[2] *ähnlich* bedeutet in diesem Fall eine Abweichung von ±5 Jahren

Events zeigt das RecommenderServlet dem Anwender alle Termine, an denen dieses Event stattfindet. Sobald der Benutzer sich für einen Termin entschieden hat, wird er auf die Kaufseite weitergeleitet, welche vom PurchaseServlet verwaltet wird.

4.5.5 PurchaseServlet

Das PurchaseServlet bildet das zuvor ausgewählte Event mit all seinen Details ab, wie dem Veranstaltungsort, dem Datum mit Uhrzeitangabe, der genauen Adresse, dem Veranstalter und der Dauer des Events. Der Anwender hat nun noch über geeignete Schaltflächen die Möglichkeit, die Anzahl der zu kaufenden Tickets, die Bezahlart und eine Rabattmöglichkeit auszuwählen. Beim Drücken der *Kaufen*-Schaltfläche schließt der Benutzer den Kauf ab und wird zurück zu seinen Empfehlungen geleitet.

4.5.6 RatingServlet

Die Bewertungs-Seite wird von dem RatingServlet verwaltet. Auf dieser kann der Benutzer erworbene Events mit einem Zahlenwert von 1 – 5 bewerten. Diese Bewertungen werden über die Schaltfläche *Bewertungen abschicken* in das System eingetragen und zukünftig bei der Berechnung von Empfehlungen für den Benutzer berücksichtigt.

4.5.7 StatistikenServlet

Das StatistikenServlet erlaubt es dem Benutzer verschiedene Statistiken über das Empfehlungssystem einzusehen. Bei Auswahl einer entsprechenden Schaltfläche werden dem Anwender verschiedene Grafiken angezeigt, wie z.B. die Anzahl der Events pro Stadt. Diese Diagramme werden mittels CanvasJS[3] abgebildet.

Des Weiteren existieren noch folgende Servlets:

- HistoryServlet: zeigt dem Benutzer seine getätigten Käufe an
- NewReleasesServlet: zeigt dem Benutzer die neusten 5 Events an
- PageNotFoundServlet: tritt jedes Mal in Aktion, sobald der Benutzer eine nicht existierende Seite aufruft. Dieses zeigt dem Benutzer einen *404 Not Found* Fehlercode.

Alle Servlets (ausgenommen dem StatistikenServlet) prüfen bei Aufruf, ob der Benutzer eingeloggt ist. Falls dies nicht der Fall ist, wird der Nutzer auf die Login-Seite weitergeleitet.

4.6 Die Datenbankanbindung

Die Kommunikation mit der Datenbank erfolgt primär über die Klasse DatabaseAdapter. Diese benötigt allerdings zum Aufbau der Verbindung die DatabaseConnection.

4.6.1 DatabaseConnection

Diese Klasse beinhaltet die Verbindungsinformationen zu dem Datenbanksystem und liefert über die Methode *getConnetion()* eine Verbindung zur Datenbank. Um die Geschwindigkeit für Anfragen an die Datenbank zu beschleunigen, werden die Connections mittels eines ConnectionPools über die von Apache Tomcat bereitgestellte Klasse BasicDataSource verwaltet. Sobald eine Connection für eine Anfrage benötigt wird, kann diese so direkt aus dem ConnectionPool entnommen und nach Abschluss der Anfrage dorthin zurückgelegt werden.

[3] CanvasJS (https://canvasjs.com/) stellt in JavaScript codierte Daten als Grafiken und Diagramme dar.

4.6.2 DatabaseAdapter

Die Klasse DatabaseAdapter stellt Methoden zum Holen von verschiedenen Daten aus der Datenbank bereit. Ein Beispiel für den Ablauf einer solchen Methode ist nachfolgend dargestellt. Der abgebildete Ausschnitt zeigt die Anfrage zum Holen aller nicht bewerteten Käufe eines Benutzers:

```
PreparedStatement statement = conn.prepareStatement(
          "SELECT * " //
        + "FROM events " //
        + "WHERE eventID IN "
        + "    (" //
        + "    SELECT eventID " //
        + "    FROM purchases " //
        + "    WHERE userID = ? "
        + "    ) " //
        + "AND eventID NOT IN "
        + "    (" //
        + "    SELECT eventID " //
        + "    FROM ratings " //
        + "    WHERE userID = ? " //
        + "      AND rating > 0" //
        + "    )");
statement.setInt(1, user.getId());
statement.setInt(2, user.getId());
```

Abbildung 7 Methode getUnratedPurchasesOfUser(...)

Über die Methode *prepareStatement(…)* wird eine Abfrage vorbereitet. In dieser Abfrage sind allerdings über Fragezeichen noch Platzhalter eingebaut, welche im Anschluss mit passenden Werten ersetzt werden, z.B. wird in diesem Fall mittels *statement.setInt(1, user.getId())* die ID des Benutzers in die Abfrage eingefügt.

Anschließend wird das Ergebnis der Datenbankanfrage ausgewertet und die gelieferten Events mit der Methode *createEventFromResult(…)* in JavaBeans konvertiert und in einer Liste zurückgegeben:

```
List<Event> events = new ArrayList<>();

...

ResultSet result = statement.executeQuery();
while (result.next()) {
    events.add(createEventFromResult(result));
}

return events;
```

Abbildung 8 UnratedPurchases - Erstellen von Events

4.6.3 RunWithConnection

Damit die einzelnen Methoden für Datenbankabfragen die Verbindung zur Datenbank nicht selbst verwalten müssen, existiert hierfür eine weitere Methode: *runWithConnection(…)*. Diese Methode öffnet eine Datenbankverbindung und schließt diese nach beendeter Anfrage wieder bzw. legt diese in den Verbindungspool zurück.

```
private static <R> R runWithConnection(SqlFunction<Connection, R> function)
        throws SQLException {
    try (Connection conn = getConnection()) {
        return function.apply(conn);
    }
}
```

Abbildung 9 Methode runWithConnection

Diese Methode erwartet als einzigen Parameter eine parametrisierte *SqlFunction*. Eine SqlFunction ist eine Abwandlung des Interfaces *Function*<T,R>, welches ein Argument *T* entgegennimmt und ein Ergebnis *R* zurückliefert [4]. Die *SqlFunction* funktioniert nach dem gleichen Prinzip, kann darüber hinaus allerdings auch eine *SQLException* werfen.

```
public interface SqlFunction<T, R> {

    R apply(T t) throws SQLException;

}
```

Abbildung 10 Interface SqlFunction

Die Methode *runWithConnection(…)* erwartet nun eine SqlFunction mit einem frei wählbaren Datentypen *R*, welcher zugleich als Rückgabedatentyp dieser Methode dient und einem festgelegten Eingabedatentypen *T* vom Typ *Connection*. Die gegebene Funktion bekommt somit immer eine *Connection* als Eingabedatentyp und produziert ein Ergebnis vom gewählten Typ *T*.

Durch den *try-with-resources*-Block wird eine Verbindung zur Datenbank geholt (mittels *getConnection()*), danach auf die gegebene Funktion angewendet und nach Beendigung automatisch wieder geschlossen.

Die Methoden für Datenbankabfragen werden mit der Verwendung auf das Wesentliche reduziert und es existiert kein redundanter Code für die Connection-Verwaltung. Die Methoden rufen lediglich *runWithConnection(…)* wie in Abbildung 11 Anwenden von runWithConnection(...) gezeigt auf und die eigentliche Logik erfolgt dann innerhalb der geschweiften Klammern.

```
return runWithConnection(conn -> {

  ...

});
```

Abbildung 11 Anwenden von runWithConnection(...)

Nachfolgend wird dargestellt, wie sich die Benutzung der Methode *runWithConnection(…)* auswirkt:

```
List<Event> allEvents = new ArrayList<>();
Connection conn = null;
try {
    conn = getConnection();

    PreparedStatement statement = conn.prepareStatement("SELECT * FROM events");

    ResultSet result = statement.executeQuery();
    while (result.next()) {
        allEvents.add(createEventFromResult(result));
    }

} finally {
    if(conn != null) {
        conn.close();
    }
}

return allEvents;
```

Abbildung 12 Methode ohne runWithConnection(...)

```
return runWithConnection(conn -> {
    List<Event> events = new ArrayList<>();

    PreparedStatement statement = conn.prepareStatement("SELECT * FROM events");

    ResultSet result = statement.executeQuery();
    while (result.next()) {
        events.add(createEventFromResult(result));
    }

    return events;
});
```

Abbildung 13 Methode mit runWithConnection(...)

4.7 Session

Abseits der Servlets, JSPs und der Datenbankverbindung gibt es noch eine wichtige Klasse, die das Verwalten von Daten über mehrere Servlets hinweg ermöglicht. Die Klasse *Session* dient als Speicher von Daten, die während der Laufzeit des Systems beständig sind. Ein Beispiel hierfür ist der angemeldete Benutzer. Dieser soll nach Beendigung des LoginServlets weiterhin bestehen bleiben und für andere Servlets verfügbar sein. Das LoginServlet speichert bei erfolgreicher Anmeldung den Benutzer mittels eines Keys in der Session ab. Über diesen Key können die anderen Servlets auf den Nutzer wieder zugreifen. Beim Abmelden wird der Benutzer wieder aus der Session entfernt.

5. ZUSAMMENFASSUNG

In dieser Facharbeit wurde erläutert, was ein Recommender-System ist, welche Möglichkeiten es gibt Empfehlungen zu berechnen und wie ein solches System mitsamt seinen Servlets, JSPs und Datenbankanbindung aufgebaut wird. Zudem wurde speziell auf den Pearson-Spearman-Algorithmus zum Collaborative-Filtering eingegangen und dieser näher erläutert.

6. AUSBLICK

In kleinen Systemen mit überschaubaren Datenbankeinträgen funktioniert der vorgestellte Collaborative Filtering Algorithmus in SQL noch schnell genug. Für größere Datenbanken mit unzähligen Datensätzen würde dieser Algorithmus in der vorliegenden Form zu langsam sein. Hier käme eine Materialized-View zum Einsatz, um die Berechnung der Durchschnittsbewertung jedes Benutzers vorwegzunehmen. Diese View würde bei jeder neu eingefügten Bewertung die kompletten Vektoren im Hintergrund neu berechnen. Bei einer Anfrage und Berechnung von ähnlichen Nutzern mittels des Pearson-Spearman-Verfahrens greift dieser dann auf diese neu berechneten Werte zu und berechnet diese nicht selbst. Dies beschleunigt die Berechnung der Empfehlungen eines Benutzers.

Zudem fällt bei diesem System auf, dass neue Events, die noch kein Benutzer bewertet hat, mit hoher Wahrscheinlichkeit dem Anwender nicht empfohlen werden. Dies liegt mitunter an den verwendeten Empfehlungsalgorithmen und sollte bei einer kommerziellen Nutzung eines Empfehlungssystems möglichst nicht auftreten.

7. QUELLENVERWEIS

[1] Burke, R. Hybrid Recommender Systems: Survey and Experiments. *User Model User-Adap Inter* 12**,** 331–370 (2002). https://doi.org/10.1023/A:1021240730564

[2] Özgöbek, Ö.; Gulla, J. and Erdur, R. (2014). A Survey on Challenges and Methods in News Recommendation.In *Proceedings of the 10th International Conference on Web Information Systems and Technologies - Volume 1: WEBIST*, ISBN 978-989-758-024-6, pages 278-285. DOI: 10.5220/0004844202780285Ding, W., and Marchionini, G. *A Study on Video Browsing Strategies.* Technical Report UMIACS-TR-97-40, University of Maryland, College Park, MD, 1997.

[3] Hamilton, G., 1997. JavaBeans™ Specification 1.01 Final Release. *Sun Microsystem, Augustus*, *8*

[4] Java™ Platform, Standard Edition 8 API Sepcification, (2020). *Interface Function<T,R>*. [online] Available at: https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html [Accessed Feb 17, 2020]