Übungsblatt

Komponentenbasierte Entwicklung komplexer Anwendungen

Pojektdokumentation: Vocabular-Quiz

Datenverarbeitungskonzept eines betrieblichen Informationssystems

Bearbeitungszeitraum: 08. April 2021 - 25. Juli 2021

Timur Burkholz	575306
Seweryn Kozlowski	575310
Maximilian Alexander Reech	575311

Betreuer: Prof. Dr. Martin Kempa

Inhaltsverzeichnis

Inhaltsverzeichnis I

Abbildungsverzeichnis II

1 Komponentenschnitt 1

2 Schnittstellenbeschreibung 3

3 Konzeptionelles Datenmodell 9

4 Präsentationsschicht 10

5 Frameworks 15

6 Ablaufumgebung 16

Abbildungsverzeichnis

1.1 Komponentendiagramm 1

3.1 Datenmodell 9

4.1 Startbildschirm 10
4.2 Registrierung 10
4.3 Einloggen 11
4.4 Match-Lobby 11
4.5 Match erstellen 12
4.6 Match starten 12
4.7 Kategorie auswählen 13
4.8 Antwort abgegeben 13
4.9 Auf zweiten Spieler warten 14
4.10 Match fertigstellen 14

1 Komponentenschnitt

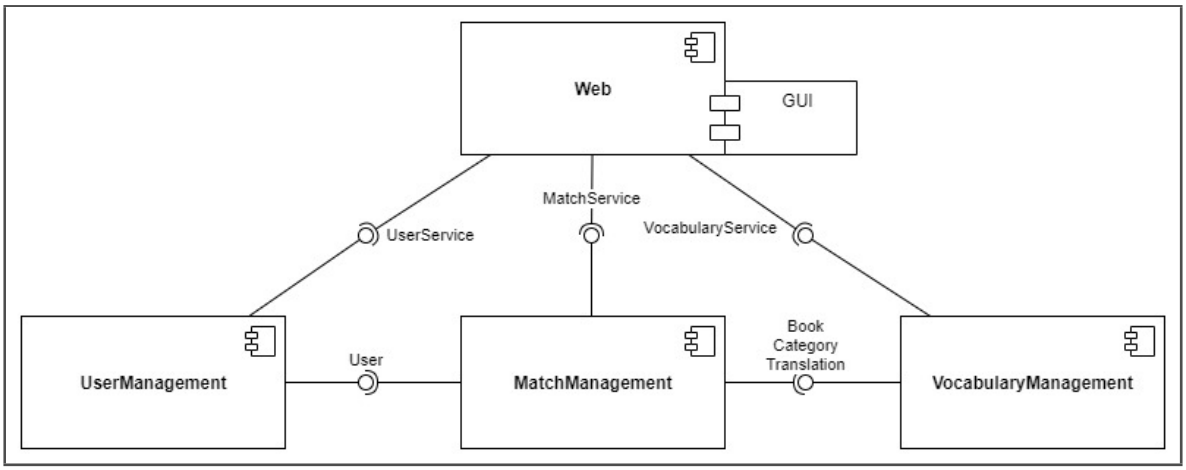

Abb. 1.1: Komponentendiagramm

Wie in Abbildung 1.1 zu sehen ist, besteht die Software aus vier Komponenten. Im Folgenden werden die Aufgaben dieser Komponenten näher beschrieben.

Web

Die „Web“ Komponente stellt die eigentliche Serveranwendung dar. Diese verwaltet somit den Zugriff über den Client bzw. der GUI auf die Anwendungslogik. Dementsprechend ist sie über Interfaces mit den anderen Komponenten verbunden.

UserManagement

Diese Komponente ist für die Benutzerverwaltung zuständig. Dabei müssen Benutzer angelegt werden und sich amelden können. Neben der Verbindung zur „Web“ Komponente ist auch eine Verbindung über die Klasse „User“ zur „MatchManagement“ Komponente notwendig, damit diese ihre Aufgaben erfüllen kann.

MatchManagement

Die Komponente „MatchManagement“ stellt die eigentliche Match- bzw. Spiele- oder auch Anwendungslogik zur Verfügung. Damit diese umfassend funktionieren kann benötigt sie die Klassen „Book“, „Category“ und „Translation“ der „VocabularyManagement“ Komponente. Zudem muss die „UserManagement“ Komponente die Klasse „User“ zu Verfügung stellen.

VocabularyManagement

Zur Verwaltung der Vokabeldaten wird die „VocabularyManagement“ Komponente herangezogen. Diese stellt der „MatchManagement“ Komponente notwendige Klassen. Zudem erlaubt sie dem Benutzer über die „Web“ Komponente neue Daten als Datei hochzuladen.

2 Schnittstellenbeschreibung

Im Folgenden werden die Schnittstellen zwischen den Anwendungskomponenten und der „Web“ Komponente beschrieben.

UserService

Als Schnittstelle zwischen „Web“ Komponente und „UserManagement“ Komponente dient das „UserService“ Interfaces. Dieses biete folgende Funktionalitäten:

```
package vocab.services;

import vocab.domain.User;
import vocab.exceptions.ResourceNotFoundException;
import java.sql.SQLException;

/**
 * The following interface provides methods to manage users.
 * @version 0.1
 */
public interface UserService {
    /**
     * This method adds an new user.
     * @param username The name of the new user.
     * @param password The required password of the new user.
     * @return The method returns a the added user.
     * @throws SQLException The method throws a SQLException in case of the
         requested username is already in use.
     */
    User addUser(String username, String password) throws SQLException;

    /**
     * This method is for respective logging a user in.
     * @param username The name of the user.
     * @param password The required password of the user.
     * @return The method returns a User instance.
     * @throws ResourceNotFoundException The method throws an exception in
         case of the requested user could not be found.
     */
    User getUser(String username, String password) throws
        ResourceNotFoundException;
```

```
    /**
     * This method is used to get a user.
     * @param id The method requires the id representing the user.
     * @return The method return the requested user.
     * @throws ResourceNotFoundException The method throws an exception in
         case of the requested user could not be found.
     */
    User getUserById(Long id) throws ResourceNotFoundException;
}
```

MatchService

Die Schnittstelle zwischen „Web“ Komponente und der „MatchManagement“ Komponente wird durch das „MatchService“ Interfaces definiert. Dieses stellt folgende Funktionalitäten zur Verfügung:

```
package vocab.services;

import vocab.domain.*;
import vocab.exceptions.ResourceNotFoundException;
import javax.persistence.OptimisticLockException;
import java.util.List;

/**
 * This interface provides methods to manage matches.
 * @version 0.1
 */
public interface MatchService {
    /**
     * This method can be used to create a new match.
     * @param user The method requires the user who creates the match.
     * @param book The method requires a book, used to play.
     * @return The method returns a new Match instance.
     */
    Match createMatch(User user, Book book);

    /**
     * This method can be used to access a match.
     * @param match_id The method requires the match_id representing the match.
     * @return The method returns a Match instance, or null in case of not
         finding a match with the match_id.
     * @throws ResourceNotFoundException The method throws an exception case
```

```
     of the match cannot be found.
 */
Match getMatch(Long match_id) throws ResourceNotFoundException;

/**
 * This method can be used to update a match persistent.
 * @param match The method requires the match to be updated.
 * @return The method returns a boolean representing the success of the
     method.
 * @throws OptimisticLockException The method throws an exception case of
     the match was already updated.
 */
void updateMatch(Match match) throws OptimisticLockException;

/**
 * This method can be used to get all matches with only one user.
 * @return The method returns a List of matches representing all matches
     with only one user.
 */
List<Match> getAvailableMatches(User user);

/**
 * This method is used to add a second user to a match.
 * @param user The user that should be added.
 * @param match_id The id ot the match.
 * @return The method returns the match that the second user was added.
 * @throws OptimisticLockException The method throws an exception case of
     the match was updated.
 */
Match joinMatch(User user, Long match_id) throws OptimisticLockException,
    ResourceNotFoundException;

/**
 * This method provides the functionality to answer a question.
 * @param answer The method requires a String instance representing the
     answer.
 * @param question The method requires the question instance that should
     be answered.
 * @param user The method requires the user that answers the question.
 * @return The method returns a boolean representing correctness of the
     give answer.
 */
Boolean submitAnswer(String answer, Question question, User user);

/**
 * This method provides the functionality to flag a match as finished.
```

```
     * @param match_id The method requires the id of the match that should be
         finished.
     * @throws ResourceNotFoundException The method throws an exception case
         of the match cannot be found.
     */
    void finishMatch(Long match_id) throws ResourceNotFoundException,
        OptimisticLockException;

    /**
     * This method provides the functionality to get a question.
     * @param question_id The method requires the id of the question.
     * @return The method returns the question, or null in case of not finding
         the question.
     * @throws ResourceNotFoundException The method throws an exception case
         of the question cannot be found.
     * @throws OptimisticLockException The method throws an exception case of
         the match was updated.
     */
    Question getQuestion(Long question_id) throws ResourceNotFoundException;

    /**
     * This method provides the functionality to start a round.
     * @param category The method requires the category that should be used in
         the round.
     * @param match The method requires the match that contains the round.
     * @return The method returns the round.
     * @throws OptimisticLockException The method throws an exception case of
         the match was updated.
     * @throws ResourceNotFoundException The method throws an exception case
         of the match cannot be found.
     */
    Round startRound(Category category, Match match) throws
        OptimisticLockException, ResourceNotFoundException;
}
```

VocabularyService

Zwischen „Web“ und der „VocabularyManagement“ Komponente wird die Schnittstelle durch das „VocabularyService“ Interfaces definiert. Dieses verfügt über folgende Funktionalitäten:

```
package vocab.services;

import org.springframework.web.multipart.MultipartFile;
import vocab.domain.Book;
import vocab.domain.Category;
import vocab.exceptions.BadInputFileException;
import vocab.exceptions.ItemNotFoundException;
import java.io.File;
import java.util.List;

/**
 * This interface is can be used to manage the vocabulary.
 * @version 0.1
 */
public interface VocabularyService {
    /**
     * This method provides the access to all books written in a gameDirection.
     * @return The method returns a List instance of Books representing all
         playable Books.
     */
    List<Book> getBooks();

    /**
     * This method can be used to insert a file into the database.
     * @param file The method requires the file that should be inserted.
     * @return The method returns a boolean which indicates that te database
         was changed.
     * @throws BadInputFileException The method throws a exception if the
         provided file is not using the standard vocabulary format.
     */
    Boolean addFile(File file) throws BadInputFileException;

    /**
     * This method is used to get a category.
     * @param id The method requires the id representing the category.
     * @return The method returns the category, or null in case of not finding
         the requested category.
     * @throws ItemNotFoundException The method throws an exception in case
         there is no book with a matching id.
```

```
     */
    Category getCategory(Long id) throws ItemNotFoundException;

    /**
     * This method is used to get a book.
     * @param id The method requires the id representing the book.
     * @return The method returns the book, or null in case of not finding the
         requested book.
     * @throws ItemNotFoundException The method throws an exception in case
         there is no book with a matching id.
     */
    Book getBook(Long id) throws ItemNotFoundException;

    /**
     * This method is used to insert a multipart file into the database.
     * @param file The method requires the multipart file that should be
         inserted.
     * @return The method returns a boolean which indicates that te database
         was changed.
     * @throws BadInputFileException The method throws a exception if the
         provided file is not using the standard vocabulary format.
     */
    Boolean addMultipartFileHelper(MultipartFile file) throws
        BadInputFileException;
}
```

3 Konzeptionelles Datenmodell

Im Folgenden ist das verwendete Datenbankmodell zu sehen.

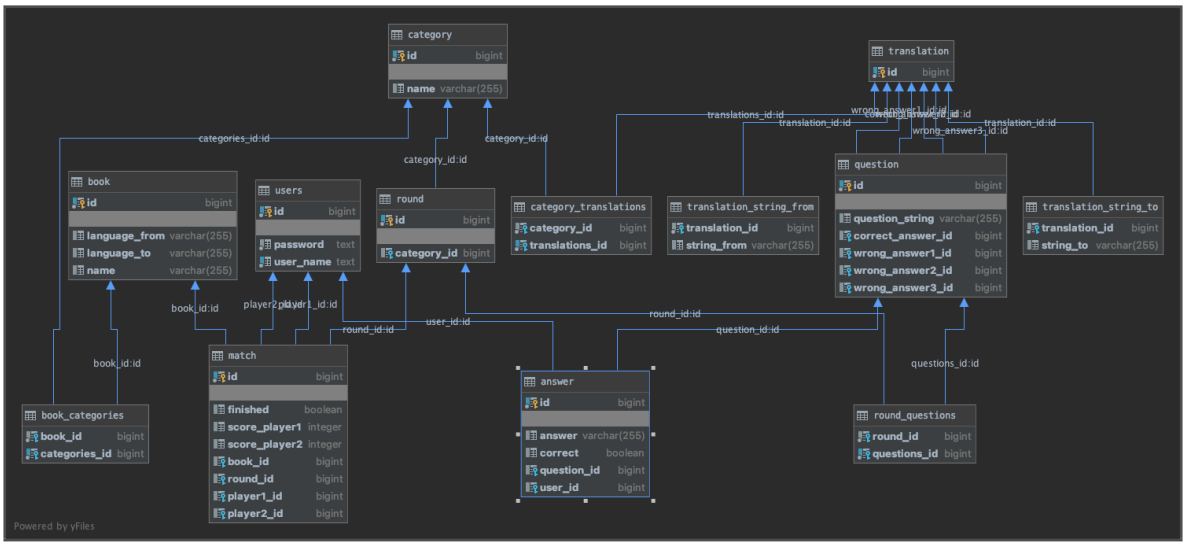

Abb. 3.1: Datenmodell

4 Präsentationsschicht

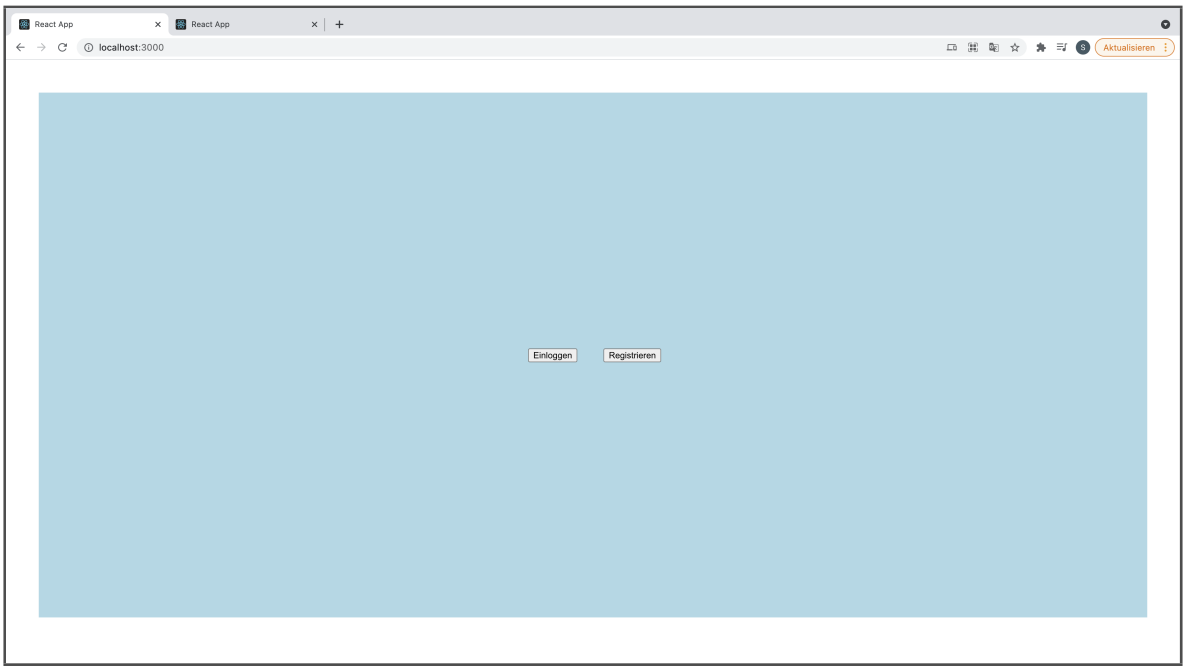

Abb. 4.1: Startbildschirm

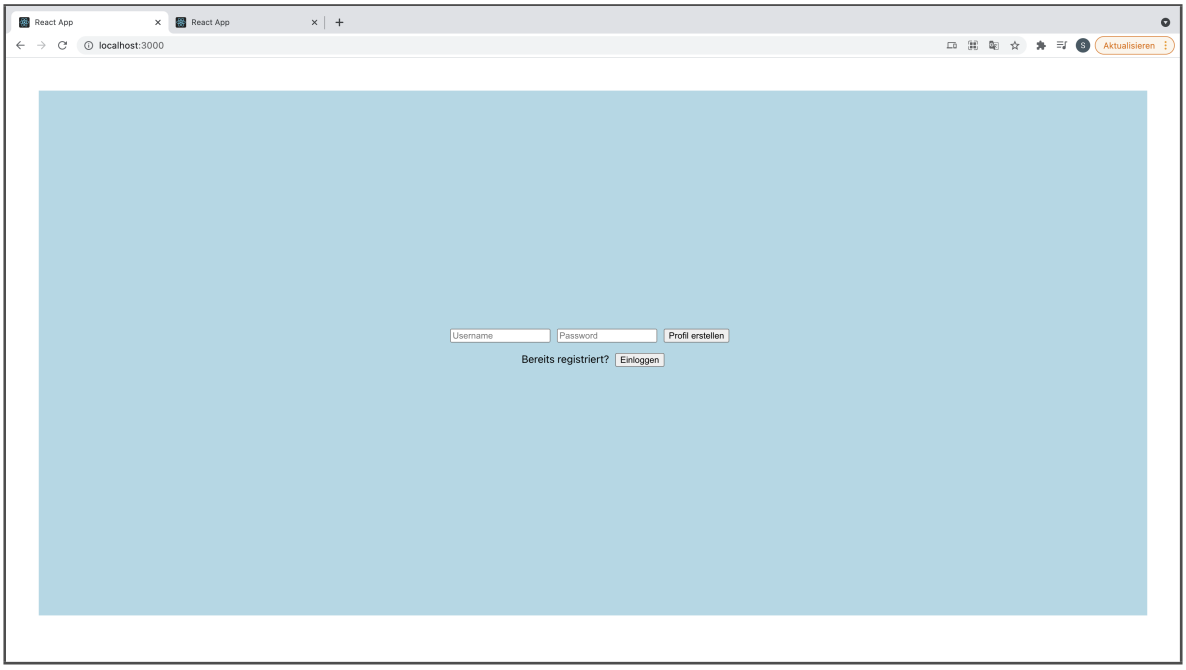

Abb. 4.2: Registrierung

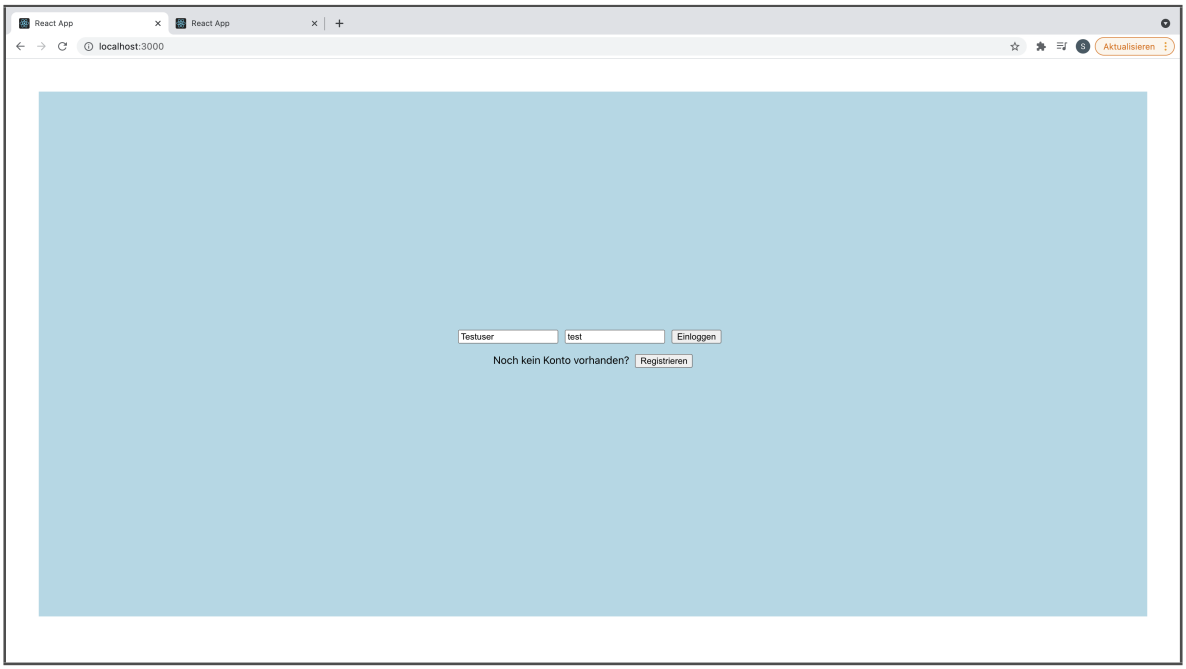

Abb. 4.3: Einloggen

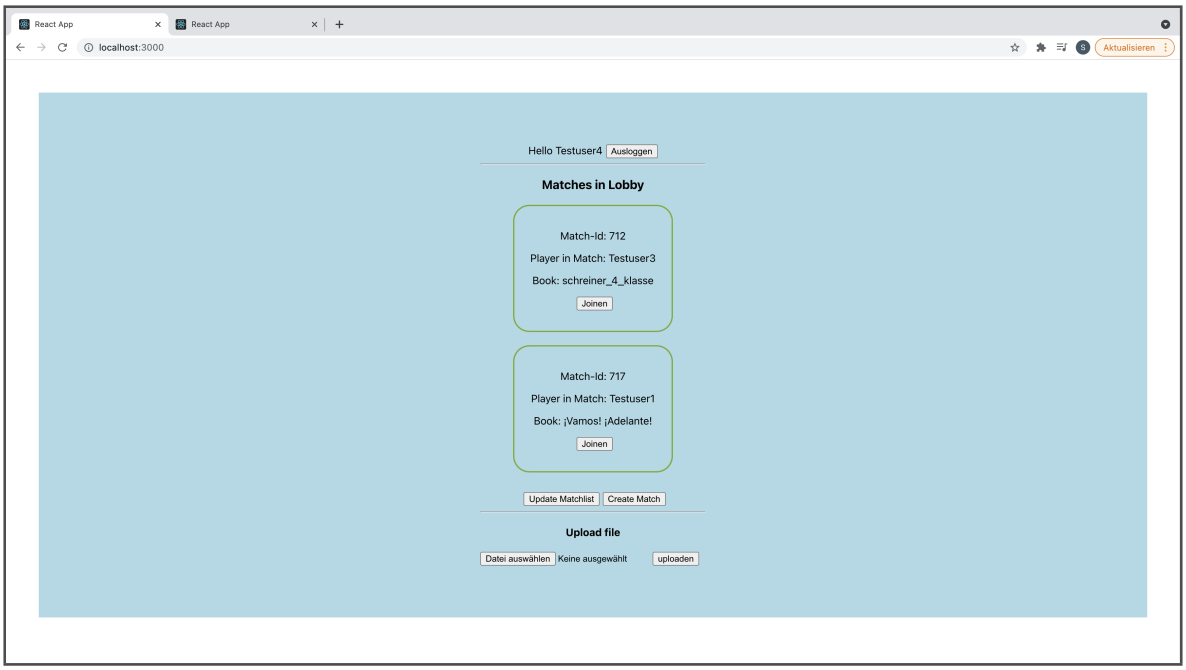

Abb. 4.4: Match-Lobby

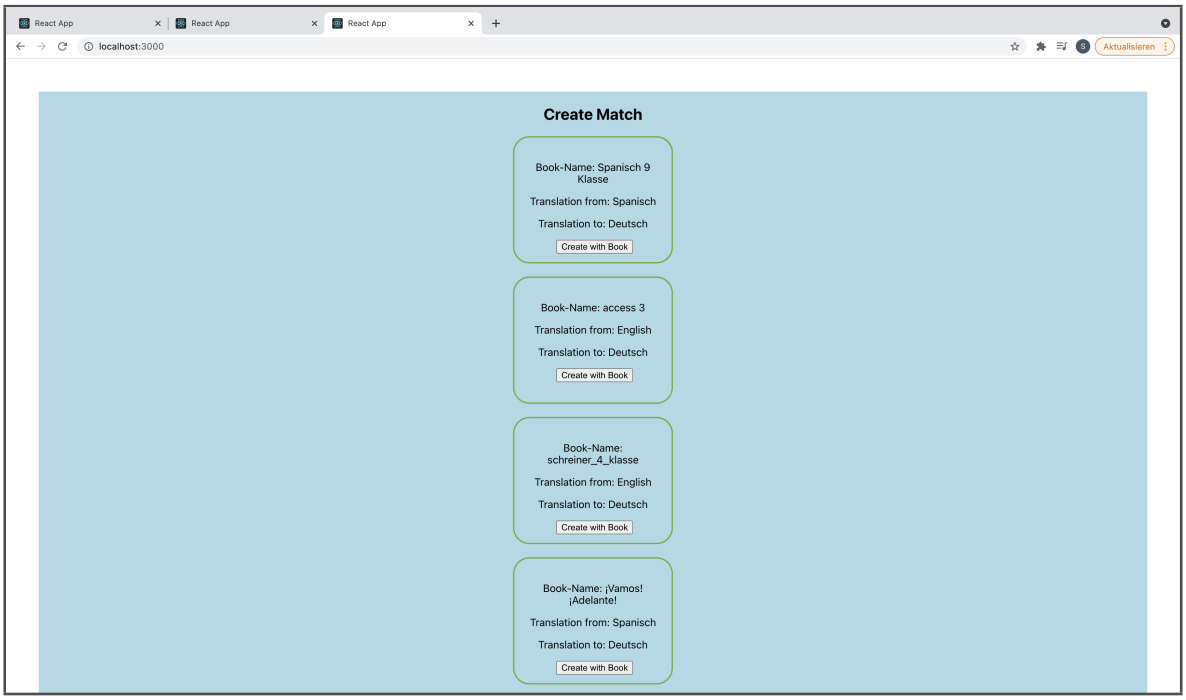

Abb. 4.5: Match erstellen

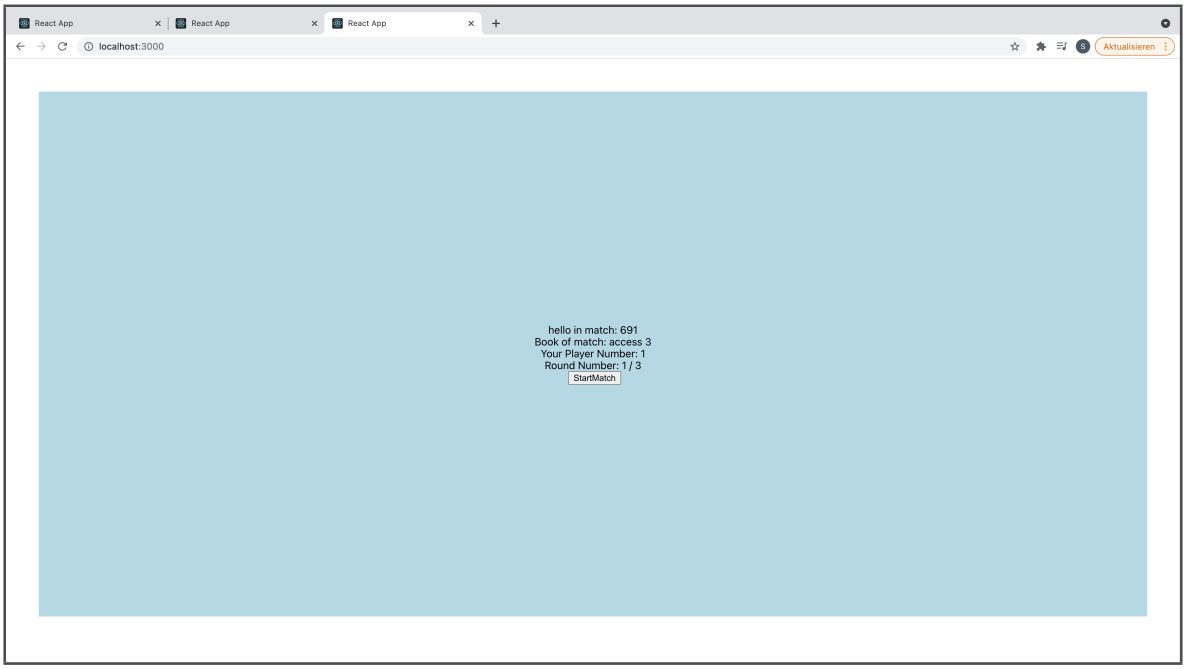

Abb. 4.6: Match starten

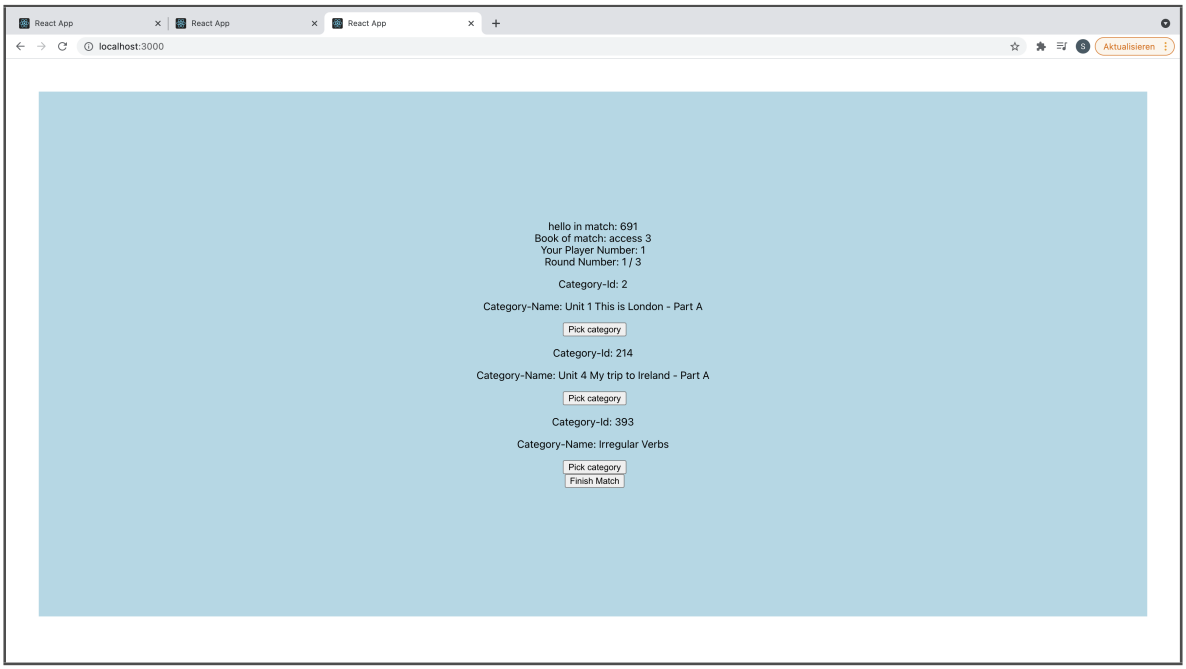

Abb. 4.7: Kategorie auswählen

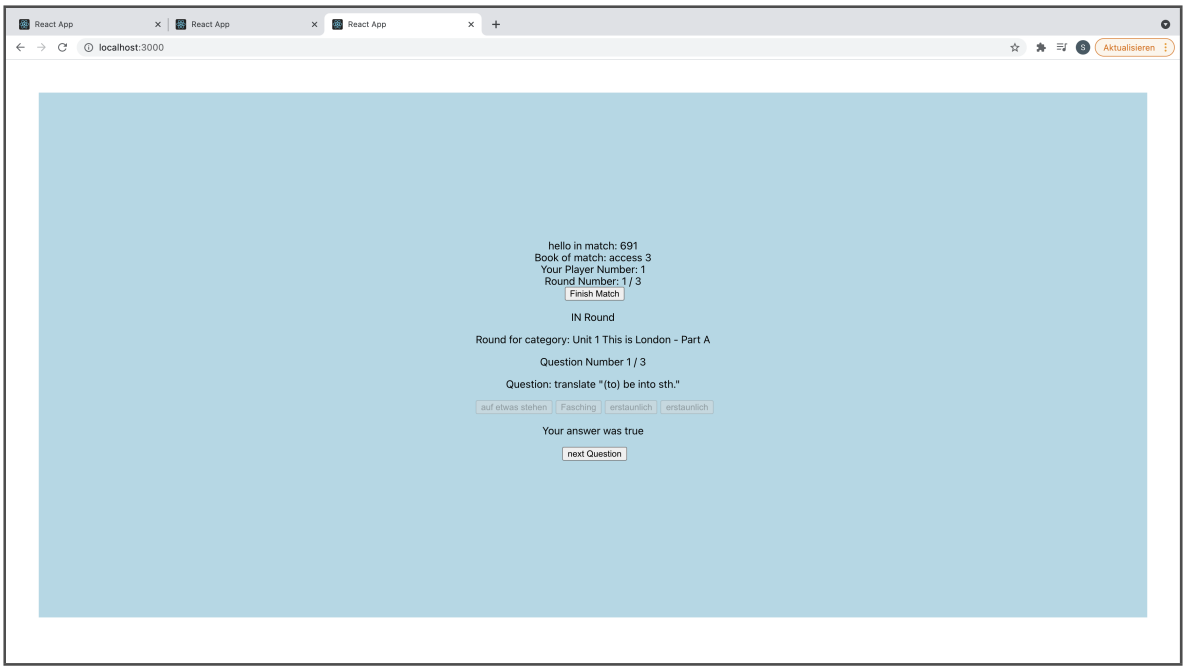

Abb. 4.8: Antwort abgegeben

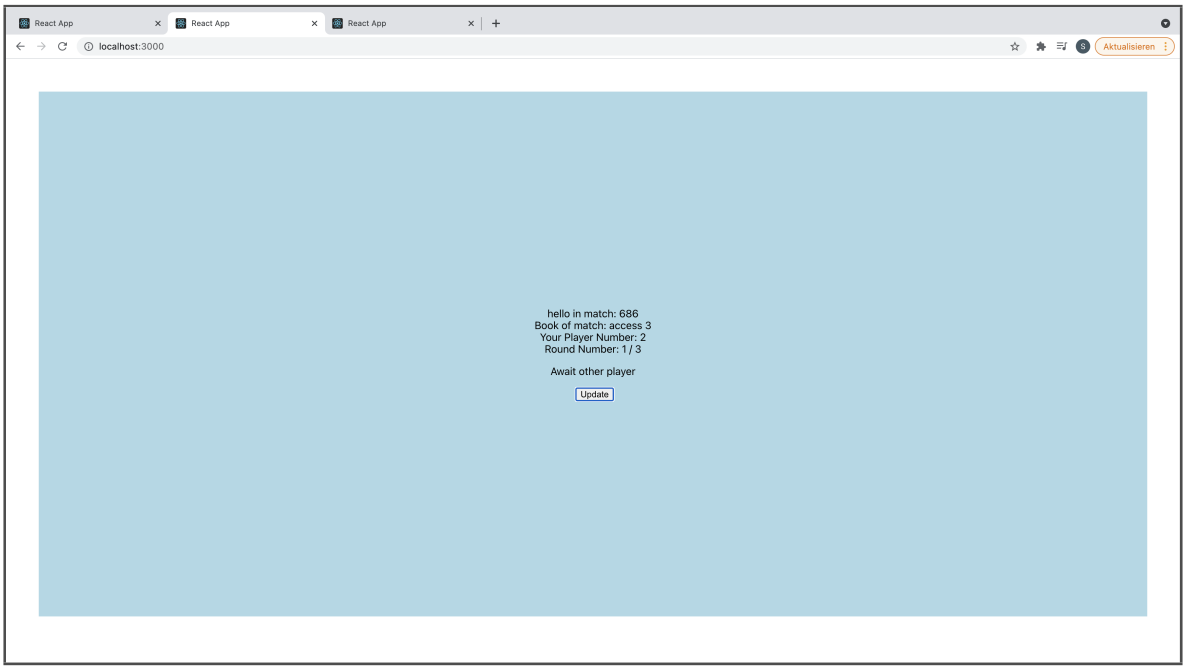

Abb. 4.9: Auf zweiten Spieler warten

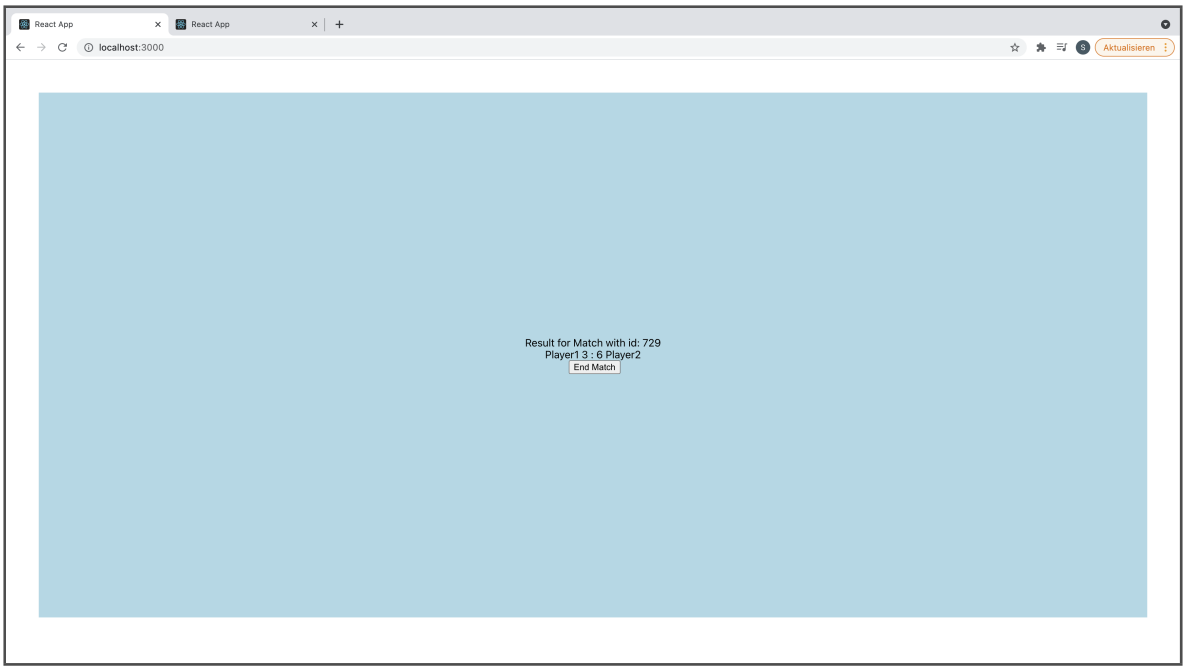

Abb. 4.10: Match fertigstellen

5 Frameworks

- Spring Boot
- JPA Hibernate
- Axios
- React

6 Ablaufumgebung

Umgebung	Version
Betriebssystem	Windows 10
Java	JDK 11
Maven	4.0.0
Datenbank	PostgreSQL 13