Protokoll: Monster Trading Cards Game

Describes design:

Es gibt drei Hauptservices: CardService, BattleService und LoginService.

Die Service-Klassen dienen dazu, Anfragen von Clients zu verarbeiten und die entsprechenden Aktionen durchzuführen. Sie implementieren das Interface "Service" und die Methode "handleRequest", die aufgerufen wird, wenn eine Anfrage empfangen wird.

Die Service-Klassen arbeiten eng mit den Controller-Klassen zusammen. Wenn eine Anfrage empfangen wird, wird die entsprechende Service-Klasse aufgerufen. Sie überprüft, welche Art von Anfrage es ist (GET, POST, PUT, etc.) und welcher Endpunkt angefordert wurde (z.B. /cards, /deck, /stats). Basierend auf diesen Informationen ruft sie die entsprechende Methode in der Controller-Klasse auf, um die Anfrage zu verarbeiten.

Der CardService ist für die Verwaltung von Karten zuständig. Er kann Karten anzeigen, die ein bestimmter Benutzer besitzt, und er kann ein Deck für einen Benutzer erstellen oder ändern. Der Service verwendet den CardController und den CardRepository, um Anfragen zu verarbeiten und auf die Datenbank zugreifen zu können.

Der BattleService ist für die Verwaltung von Kämpfen zuständig. Derzeit ist der Code jedoch sehr begrenzt und es gibt nur eine Funktion, die die Statistiken eines Benutzers abruft. Der Service verwendet den BattleController und BattleRepository, um Anfragen zu verarbeiten und auf die Datenbank zugreifen zu können.

Der LoginService ist für die Anmeldung und die Verwaltung von Benutzerkonten zuständig. Er kann überprüfen, ob ein Benutzer existiert, Anmeldeinformationen überprüfen und einen neuen Benutzer erstellen. Der Service verwendet den LoginController und UserRepository, um Anfragen zu verarbeiten und auf die Datenbank zugreifen zu können.

Es gibt auch einige Repositories-Klassen, die als Brücke zwischen den Controllern und der Datenbank dienen. Sie enthalten Methoden, um Daten aus der Datenbank abzurufen und zu ändern.

Es gibt auch ein paar Model-Klassen wie Card, User, Stats, die als Datenträger verwendet werden und die Daten repräsentieren, die in der Datenbank gespeichert sind und von den Repositories abgerufen werden.

Es gibt auch ein paar Hilfsfunktionen und Konstanten in den Klassen, die dazu beitragen, dass der Code sauberer und lesbarer wird und wiederholte Aufgaben automatisch erledigt werden.

Die Hilfsfunktionen sind in den verschiedenen Repository-Klassen definiert. Diese Klassen dienen dazu, die Interaktion mit der Datenbank zu verwalten. Beispiele für Hilfsfunktionen sind retrieveCards() in der CardRepository Klasse, die Karten für einen bestimmten Benutzer abruft, oder createNewUser() in der UserRepository Klasse, die einen neuen Benutzer in der Datenbank erstellt. Es gibt auch Hilfsfunktionen in den Controller-Klassen, die die Logik der Anwendung verwalten, wie z.B. displayDeck() in der CardController Klasse, die das Deck eines bestimmten Benutzers anzeigt.

Describes lessons learned

Benutzerauthentifizierung und Sicherheit: Das Spiel erfordert, dass sich die Benutzer registrieren und anmelden und beinhaltet ein tokenbasiertes Authentifizierungssystem, um Benutzeraktionen zu sichern und sensibles Daten zu schützen.

```
2 usages
public boolean isAuthenticationTokenValid() {
    String token = this.headerMap.getHeader( headerName: "Authorization");
    if (token == null) {
        return false;
    }

    String[] tokenParts = token.split( regex: " ")[1].split( regex: "-");
    if(tokenParts[1].equals("mtcgToken")){
        return tokenParts[0].equals(getPathParts().get(1));
    }
    return false;
}

1 usage
public boolean isAdminTokenValid() {
    String token = this.headerMap.getHeader( headerName: "Authorization");
    if (token == null) {
        return false;
    }

    String[] tokenParts = token.split( regex: " ")[1].split( regex: "-");
    return tokenParts[0].equals("admin") && tokenParts[1].equals("mtcgToken");
}

6 usages
public String getTokenUser() {
    String token = this.headerMap.getHeader( headerName: "Authorization");
    if (token == null) {
        return "";
    }

    String[] tokenParts = token.split( regex: " ")[1].split( regex: "-");
    if(tokenParts[1].equals("mtcgToken")){
        return tokenParts[0];
    }
    return "";
}
```

Dieser Code ist eine Klasse namens "TokenService", die dafür verantwortlich ist, die Gültigkeit von Tokens zu überprüfen, die in HTTP-Anforderungen gesendet werden. Die Klasse nimmt ein "HeaderMap"-Objekt als Konstruktorargument entgegen, das verwendet wird, um den "Authorization"-Header der Anforderung abzurufen. Es gibt drei öffentliche Methoden in dieser Klasse: "isAuthenticationTokenValid", "isAdminTokenValid" und "getTokenUser". "isAuthenticationTokenValid" Methode überprüft, ob das Token, das in der Anforderung gesendet wurde, gültig ist, indem es überprüft, ob es dem erwarteten Format entspricht und ob der erste Teil des Tokens mit einem bestimmten Wert übereinstimmt. "isAdminTokenValid" Methode überprüft, ob das Token ein gültiger Administrator-Token ist, indem es überprüft, ob der erste Teil des Tokens "admin" ist und ob der zweite Teil des Tokens "mtcgToken" ist. "getTokenUser" Methode gibt den ersten Teil des Tokens zurück, falls es gültig ist. Wenn kein Token gefunden wurde oder das Token ungültig ist, werden leere Strings zurückgegeben.

Datenbankverwaltung: Das Spiel erfordert eine PostgreSQL-Datenbank, um Benutzer- und Spieledaten zu speichern und abzurufen.

Integration-Tests/Unit-Test

Integrationstests sind wichtig, weil sie sicherstellen, dass die verschiedenen Komponenten einer Anwendung wie erwartet zusammenarbeiten. Sie testen die Interaktion zwischen verschiedenen Modulen und Schichten einer Anwendung und stellen sicher, dass sie ordnungsgemäß miteinander kommunizieren und Daten austauschen können. Im speziellen Fall testen diese Integrationstests, ob die Anwendung erfolgreich auf HTTP-Anfragen reagieren kann, die über die curl-Befehlszeile gesendet werden und ob die erwarteten Antworten zurückgegeben werden. Sie stellen sicher, dass die Anwendung auch unter realen Umständen und mit externen Abhängigkeiten wie einem Server ordnungsgemäß funktioniert. Unit-Tests sind Tests, die verwendet werden, um einzelne Teile des Codes (Einheiten) in einer Software-Anwendung zu überprüfen. Sie werden verwendet, um sicherzustellen, dass jeder Teil des Codes ordnungsgemäß funktioniert und dass Änderungen an dem Code nicht unbeabsichtigt negative Auswirkungen auf andere Teile des Codes haben. Unit-Tests werden in der Regel automatisch ausgeführt und ermöglichen es Entwicklern, schnell Fehler zu finden und zu beheben.

Der erste Test, testCreateUser, verwendet das curl-Kommando, um einen POST-Request an die URL "http://localhost:10001/users" zu senden und überprüft, ob die Ausgabe "User successfully created" entspricht.

Der zweite Test, testLoginUsers, verwendet eine HttpURLConnection, um einen POST-Request mit Benutzername und Passwort an die URL "http://localhost:10001/sessions" zu senden und überprüft, ob die Antwortcode 200 (OK) ist.

Der dritte Test, testSuccessfulRegistration, verwendet auch das curl-Kommando, um einen POST-Request zu senden und überprüft, ob die Ausgabe "Successfully created user" enthält.

Der vierte Test, testUnsuccessfulLogin, ist ähnlich, aber überprüft, ob die Ausgabe "Invalid login credentials" enthält. Der letzte Test, testSuccessfulPackageCreation, verwendet das curl-Kommando erneut und sendet einen POST-Request mit einer Liste von Paketen und überprüft, ob die Ausgabe "Successfully created package" enthält.

…….

Describes unique feature

Create Custom Cards:

```
public void createCustomCard(Card card, UnitOfWork unitOfWork) {
    try (PreparedStatement statement = unitOfWork.prepareStatement(INSERT_CARD_SQL)) {
        statement.setString( parameterIndex: 1, card.getName());
        statement.setString( parameterIndex: 2, card.getType());
        statement.setString( parameterIndex: 3, card.getId());
        statement.setInt( parameterIndex: 4, card.getDamage());
        statement.setString( parameterIndex: 5, card.getElement());
        statement.setString( parameterIndex: 6, card.getPackageID());
        statement.execute();
    } catch (SQLException exception) {
        throw new DataAccessException("Error occurred while creating card", exception);
    }
}
public Response createCustomCard(Request request) {
    UnitOfWork unitOfWork = new UnitOfWork();
    try {
        Card card = this.getObjectMapper().readValue(request.getBody(), Card.class);
        this.cardRepository.createCustomCard(card, unitOfWork);
        unitOfWork.commitTransaction();
        return new Response(HttpStatus.CREATED, ContentType.PLAIN_TEXT, content: "Card successfully created");
    } catch (Exception e) {
        e.printStackTrace();
    }
    unitOfWork.rollbackTransaction();
    return new Response(
            HttpStatus.INTERNAL_SERVER_ERROR,
            ContentType.JSON,
            content: "{ \"message\" : \"Server error occurred\" }"
    );
}
```

Die Methode „createCustomCard“ ist eine Methode, die es ermöglicht, eine neue Karte zu erstellen. Sie wird von einem Benutzer aufgerufen, der eine neue Karte erstellen möchte. Die Methode akzeptiert einen Request-Parameter, der die Informationen der neuen Karte enthält, die erstellt werden soll. Es wird dann eine neue Instanz des UnitOfWork erstellt, die für die Verwaltung von Datenbanktransaktionen verantwortlich ist. Die Methode versucht dann, die Karteninformationen aus dem Request-Body zu lesen und in ein Kartenobjekt zu deserialisieren. Anschließend wird die „createCustomCard“ Methode des CardRepository aufgerufen und ihm das Kartenobjekt sowie die UnitOfWork übergeben. Diese Methode führt dann eine SQL-Abfrage aus, um die neue Karte in der Datenbank zu erstellen. Wenn die Methode erfolgreich ausgeführt wird, wird die Transaktion bestätigt und es wird eine erfolgreiche Antwort zurückgegeben. Andernfalls wird die Transaktion zurückgenommen und es wird eine Fehlerantwort zurückgegeben.

Contains tracked time

Der Aufwand für dieses Projekt betrug in etwa 120 Stunden. Davon ist ca. Die Hälfte nur für Fehlersuche aufgebracht worden.

GITHUB LINK:

GitHub wurde nur selten für genutzt, da drauf vergessen wurde. Dateien wurden am Ende hinzugefügt um zu zeigen, dass mit GitHub gearbeitet werden kann. [Moe452/Monster-Card-Trading-Game: This HTTP/REST-based server is built to be a platform for trading and battling with and against each other in a magical card-game world. (github.com)]