Inhaltsverzeichnis

Projektidee

In diesem Projekt wird eine REST-API für Sneaker entwickelt, die als Schnittstelle für einen Client dient, um Daten zu verschiedenen Sneaker-Modellen, Marken und ihren prominenten Botschaftern abzurufen. Die API bietet strukturierte Informationen und ermöglicht es, Sneaker-Daten im JSON-Format abzurufen und in andere Systeme oder Anwendungen zu integrieren. Dies bietet nicht nur eine umfassende Übersicht über verschiedene Sneaker-Modelle, sondern auch interessante Details zur Geschichte und den Prominenten, die bestimmte Marken repräsentieren.

Die API basiert auf einem Datenmodell mit drei zentralen Entitäten: **Sneaker**, **Marken** und **Prominente/Ambassadors**. Die Entität *Sneaker* enthält wichtige Merkmale wie die Marke, den Namen des Sneakers, den Retailpreis und eine Beschreibung. Die Entität *Marke* repräsentiert die jeweiligen Sneaker-Marken und beinhaltet eine eindeutige ID sowie den Markennamen. Schließlich gibt es die Entität *Celebrities/Ambassadors*, die Informationen über bekannte Persönlichkeiten enthält, die als Botschafter für bestimmte Sneaker-Marken auftreten. Für jeden Prominenten werden der Name, das Alter, eine kurze Beschreibung und die zugehörige Marke gespeichert.

Durch diese API können Clients gezielte Anfragen stellen, um etwa eine Liste aller Sneaker einer bestimmten Marke, Informationen zu den Botschaftern oder detaillierte Daten zu einzelnen Sneaker-Modellen abzurufen. Ziel ist es, eine benutzerfreundliche und flexible API zu schaffen, die Daten zu Sneakern und ihren kulturellen Hintergründen effizient verfügbar macht.

Anforderungskatalog/USER STORIES

1. **Sneaker-Informationen verwalten**
    - Erstellung von Sneaker-Einträgen: Die API ermöglicht das Hinzufügen neuer Sneaker mit den Attributen Marke, Name, Retailpreis und Beschreibung.
    - Abrufen von Sneaker-Details: Die API stellt eine Funktion zur Verfügung, um Informationen zu einzelnen Sneakern basierend auf ihrer ID oder anderen Kriterien abzurufen.
    - Aktualisierung von Sneaker-Daten macht weniger einen Sinn, da sich die Daten nach der Veröffentlichung im Markt sich schwer ändern lassen.
    - Löschen von Sneaker-Einträgen macht ebenfalls weniger einen Sinn, da der Sneaker nie komplett aus dem Markt entfernt wird.
2. **Marken-Informationen verwalten**
    - Erstellung von Marken-Einträgen: Ermöglicht das Hinzufügen neuer Marken in das System mit den Attributen ID und Markenname.
    - Abrufen von Marken-Daten: Die API bietet die Möglichkeit, eine Liste aller Marken oder Details zu einer spezifischen Marke abzurufen.
    - Aktualisierung von Marken-Informationen ist nicht geeignet, da sich die Eigenschaften wie der Name und der Founded Date sich nicht ändern.
    - Löschen von Marken ist ebenfalls nicht geeignet.
3. **Verwaltung von Prominenten und Marken-Botschaftern**
    - Erstellung von Prominenten/Botschafter-Einträgen: Die API ermöglicht das Hinzufügen neuer Prominenter, die als Botschafter für eine Marke stehen. Attribute umfassen ID, Name, Alter, Beschreibung und die zugehörige Marke.
    - Abrufen von Botschafter-Informationen: Clients können Informationen über bestimmte Botschafter abrufen, z.B. Name, Alter und die Marke, für die sie werben.
    - Aktualisierung von Botschafter-Daten: Änderungen in den Botschafter-Informationen, wie das Alter oder die Beschreibung und aktuelle Marke die er repräsentiert und werbt, können in der API aktualisiert werden.
    - Löschen von Botschafter-Einträgen: Die API unterstützt das Entfernen von Prominenten/Botschaftern aus der Datenbank.

User Story 1: Celebrity erstellen

Als Client
möchte ich einen neuen Celebrity-Eintrag in der Datenbank erstellen
damit ich Informationen über neue Prominente speichern und abrufen kann.

Akzeptanzkriterien:

- Der Endpunkt /celebrity akzeptiert eine POST-Anfrage mit dem Parameter name (erforderlich).
- Der Celebrity wird mit einer eindeutigen ID in der Datenbank gespeichert.
- Wird kein Name übergeben, gibt die API eine Fehlermeldung zurück.
- Bei erfolgreichem Speichern wird eine Bestätigung ("Saved") zurückgegeben.

User Story 2: Alle Celebrities abrufen

Als Client
möchte ich alle in der Datenbank gespeicherten Celebrity-Einträge abrufen können
damit ich eine vollständige Liste der Prominenten zur Verfügung habe.

Akzeptanzkriterien:

- Der Endpunkt /celebrity akzeptiert eine GET-Anfrage und gibt eine Liste aller Celebrity-Einträge im JSON-Format zurück.
- Die zurückgegebene Liste enthält für jeden Celebrity mindestens die Felder id, name, age, description und brandId.
- Falls keine Celebrities in der Datenbank vorhanden sind, wird eine leere Liste zurückgegeben.
- Die Anfrage ist erfolgreich, wenn der Statuscode 200 OK zurückgegeben wird.

User Story 3: Celebrity aktualisieren

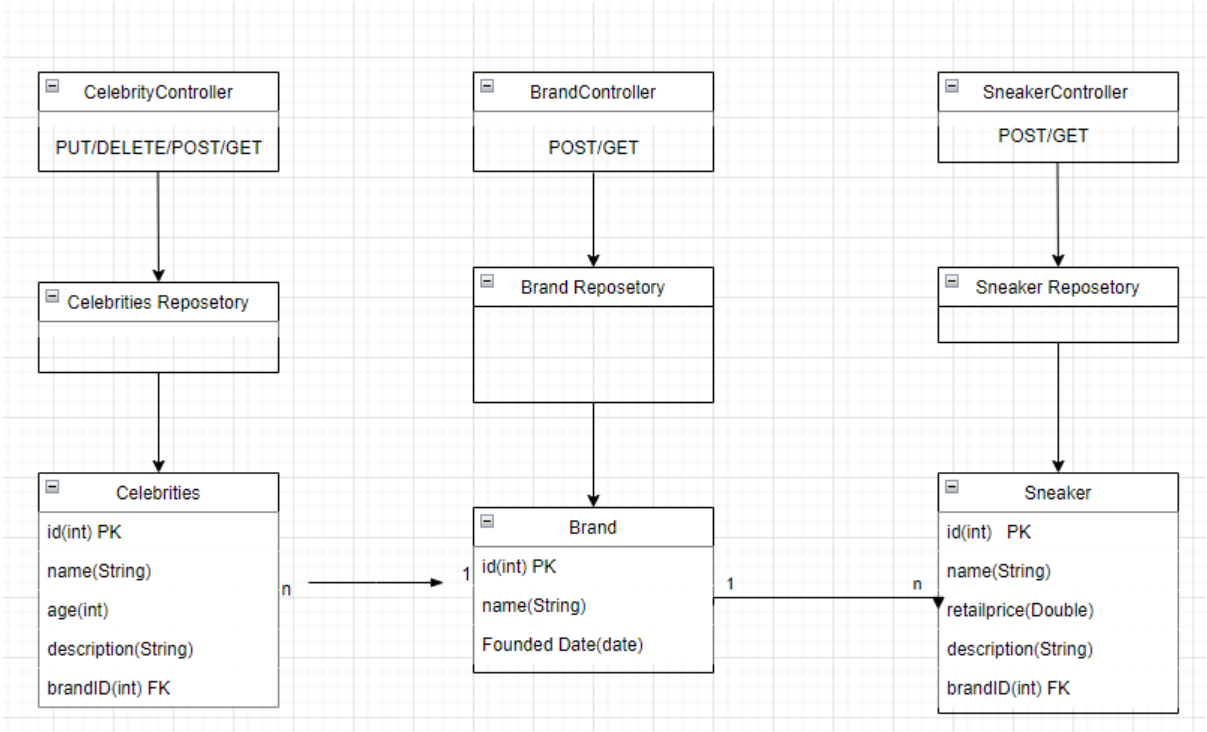
Als Client
möchte ich die Informationen eines bestehenden Celebrity-Eintrags ändern können
damit ich sicherstellen kann, dass die Daten aktuell sind.

Akzeptanzkriterien:

- Der Endpunkt /celebrity/{id} akzeptiert eine PUT-Anfrage mit den Parametern name, age und description (mindestens name ist erforderlich).

- Wenn der Celebrity-Eintrag mit der angegebenen ID nicht existiert, gibt die API eine Fehlermeldung zurück.
- Bei erfolgreicher Aktualisierung wird eine Bestätigung ("Updated") zurückgegeben.
- Falls das Update erfolgreich ist, wird der Statuscode 200 OK zurückgegeben.

Klassendiagramm



REST-Schnittstellen

In meiner Sneaker API gibt es mehrere REST-Schnittstellen (Endpoints), um mit den Daten der Celebrity-Ressource zu interagieren. Im Folgenden sind die wichtigsten Endpunkte, ihre Funktionen und die Datentypen, die sie verwenden, beschrieben.

Datentypen

Die wichtigsten Datentypen in dieser API sind die Felder der Celebrity-Klasse. Diese Attribute beschreiben die Struktur der Celebrity-Daten und definieren, wie sie in der Datenbank und den Antworten dargestellt werden.

Celebrity-Datentyp

- **id** (Long): Eindeutige ID des Celebrity. Wird automatisch von der Datenbank vergeben und dient als Primärschlüssel.
- **name** (String): Der Name des Prominenten.
- **age** (Integer): Das Alter des Prominenten (optional).
- **description** (String): Eine Beschreibung des Prominenten (optional).
- **brandId** (Long): Fremdschlüssel, der auf die zugehörige Marke verweist.

1. POST /celebrity

- **Beschreibung**: Erstellt einen neuen Celebrity-Eintrag in der Datenbank.
- **HTTP-Methode**: POST
- **Anfrageparameter**:
    - name (String, erforderlich): Der Name des Prominenten.
- **Antwort**:
    - Status 200 OK: Der Eintrag wurde erfolgreich erstellt, Rückgabe einer Bestätigung ("Saved").
    - Bei Fehlern wird eine CelebrityCouldNotBeSavedException ausgelöst.

2. GET /celebrity

- **Beschreibung**: Ruft alle Celebrity-Einträge aus der Datenbank ab.
- **HTTP-Methode**: GET
- **Anfrageparameter**: Keine
- **Antwort**:
    - Status 200 OK: Gibt eine Liste von Celebrity-Objekten zurück.

- Bei Fehlern wird eine CelebrityLoadException ausgelöst.

3. PUT /celebrity/{id}

- **Beschreibung**: Aktualisiert die Daten eines bestehenden Celebrity-Eintrags anhand der ID.
- **HTTP-Methode**: PUT
- **Pfadparameter**:
  - id (Integer, erforderlich): Die ID des zu aktualisierenden Celebrity.
- **Anfrageparameter**:
  - name (String, erforderlich): Der neue Name des Prominenten.
  - age (Integer, optional): Das neue Alter des Prominenten.
  - description (String, optional): Eine neue Beschreibung des Prominenten.

- **Antwort**:
  - Status 200 OK: Der Celebrity-Eintrag wurde erfolgreich aktualisiert, Rückgabe einer Bestätigung ("Updated").
  - Falls der Celebrity nicht gefunden wird, wird eine CelebrityNotFoundException ausgelöst.

4. DELETE /celebrity/{id}

- **Beschreibung**: Löscht einen Celebrity-Eintrag aus der Datenbank anhand der ID.
- **HTTP-Methode**: DELETE
- **Pfadparameter**:
  - id (Long, erforderlich): Die ID des zu löschenden Celebrity.
- **Antwort**:
  - Status 200 OK: Der Celebrity-Eintrag wurde erfolgreich gelöscht, Rückgabe einer Bestätigung ("Deleted").
  - Falls der Celebrity nicht gefunden wird, wird eine CelebrityNotFoundException ausgelöst.

Testplan

Testplan für manuelles Testen

User Story 1: Celebrity erstellen

Beschreibung: Der Client möchte einen neuen Celebrity-Eintrag in der Datenbank erstellen.

Positivtests:

Testfall: Hinzufügen eines Celebrities mit vollständigen Details

- **Eingabe**: name=John Doe, age=35, description="Berühmter Schauspieler"
- **Erwartetes Ergebnis**: Status 200 OK, Nachricht "Saved", Celebrity wird mit allen Details gespeichert.

Negativtests:

Testfall: Hinzufügen eines Celebrities ohne Namen

- **Eingabe**: kein name-Parameter
- **Erwartetes Ergebnis**: Status 400 Bad Request, Fehlermeldung, da name erforderlich ist.

User Story 2: Alle Celebrities abrufen

Beschreibung: Der Client möchte alle in der Datenbank gespeicherten Celebrity-Einträge abrufen.

Positivtests:

Testfall: Abrufen aller Celebrities bei nicht-leerer Datenbank (User Story 2)

- **Eingabe**: Keine (nur GET-Anfrage)
- **Erwartetes Ergebnis**: Status 200 OK, Liste aller Celebrities wird im JSON-Format zurückgegeben.

Negativtests:

Testfall: Abrufen eines Celebrities mit einer ungültigen ID

- **Eingabe**: id=-1 oder id=999 (nicht existent)
- **Erwartetes Ergebnis**: Status 404 Not Found, Fehlermeldung, dass der Celebrity nicht existiert.
- **Eingabe**: Keine (nur GET-Anfrage)
- **Erwartetes Ergebnis**: Status 200 OK, leere Liste wird zurückgegeben.

User Story 3: Celebrity aktualisieren

Beschreibung: Der Client möchte die Informationen eines bestehenden Celebrity-Eintrags ändern.

Positivtests:

Testfall: Erfolgreiche Aktualisierung eines Celebrity-Eintrags (User Story 3)

- **Eingabe**: id=1, name=John Updated, age=36, description="Berühmter Schauspieler und Regisseur"
- **Erwartetes Ergebnis**: Status 200 OK, Nachricht "Updated", Celebrity wird mit den neuen Details aktualisiert.

Negativtests:

Testfall: Aktualisieren eines nicht existierenden Celebritys

- **Eingabe**: id=999, name=Nonexistent
- **Erwartetes Ergebnis**: Status 404 Not Found, Fehlermeldung, dass der Celebrity nicht existiert.

Hilfestellung

- Hilfe vom Dozenten
- ChatGPT
- Google
- GitHub

Selbstreflexion der Arbeit

In diesem Projekt habe ich viele neue Erkenntnisse gewonnen und wertvolle Erfahrungen gesammelt. Besonders habe ich gelernt, wie anspruchsvoll es sein kann, eine vollständige Applikation von Grund auf zu entwickeln und gleichzeitig die verschiedenen Anforderungen, wie Dokumentation und Tests, im Blick zu behalten. Ein wichtiger Punkt, den ich rückblickend unterschätzt habe, war der Zeitaufwand. Ich hätte mehr Zeit für die Planung und Entwicklung der Applikation einplanen sollen und auch zuhause am Projekt arbeiten sollen, um in der Schule gezielter an den Feinheiten arbeiten zu können. Dadurch hat mir die Zeit am Donnerstag nicht gereicht, um alle Punkte abzuschließen.

Während der Entwicklung kam es zudem mehrfach zu technischen Problemen, die dazu führten, dass die Applikation nicht wie gewünscht funktionierte. Diese Schwierigkeiten kosteten mich mehr Zeit als erwartet, wodurch wichtige Aspekte wie die Installationsanleitung, den Testplan, die Validierung von Eingabedaten und die Prüfung der Datenspeicherung letztlich nicht mehr vollständig umgesetzt werden konnten. Auch geplante Erweiterungen musste ich aus Zeitmangel zurückstellen. Insgesamt hat mir dieses Projekt verdeutlicht, wie wichtig eine realistische Zeiteinschätzung und frühzeitige Planung sind. Für zukünftige Projekte nehme ich mir vor, zeitlich strukturierter zu arbeiten und ausreichend Pufferzeit für unerwartete Probleme einzuplanen.