QuizApp - Probleme bei der Entwicklung

Datenaustausch Server <-> App

Das Senden von Datenbankdaten vom Server an die App stellte ein Problem dar. JPA realisiert die Beziehungen zwischen Datenbanktabellen mittels zweiseitigen Objekt-Referenzen.

Beispiel: Beziehung zwischen Quiz und Quiz-Ersteller (User)

- Datenbank-Tabelle
  Quiz hat einen Foreign-Key zur User Tabelle in der Spalte *creator*.
- JPA Entity Objekte
- Quiz hat eine Referenz auf das User-Objekt in der Variable *creator*. Zusätzlich hat das User-Objekt eine Liste mit Referenzen auf die Quiz-Objekte, die er erstellt hat, in der Variable *createdQuizzes*.

Diese Struktur verursachte mehrere Probleme.

Nicht serialisierbar

Zum Senden von Objekten an die App werden diese zuvor als JSON serialisiert. Wenn Objekte jedoch beidseitig verbunden sind (A referenziert auf B, B referenziert auf A), tritt eine Endlosschleife beim Erstellen des JSONs auf.

Automatisches Laden

Damit die App möglichst userfreundlich und schnell funktioniert, sollen nur so wenig Daten wie möglich zwischen dem Server und der App ausgetauscht werden. JPA jedoch lädt, beim Erstellen eines Objektes aus der Datenbank, zusätzlich auch andere Objekte, die mit einer Beziehung verbunden sind. Das passiert auch, wenn diese Beziehungen gar nicht von der App benötigt werden.

Lösung: Setzen des Fetch-Types der Datenbank- bzw. Entity-Beziehungen auf LAZY.

Nachteil: Wenn diese Beziehungen jedoch doch benötigt werden, müssen sie explizit in der JPQL Query angegeben werden (“SELECT… JOIN FETCH…”).

Serialisieren von Teil-Objekten

Es werden nur ganze Objekte aus der Datenbank geladen. Jedoch müssen nicht immer alle Variablen eines Objektes an die App gesendet werden. Deshalb müssen manche Variablen beim Umwandeln in JSON ignoriert werden.

Lösung 1): Mit der @Expose Annotation von GSON können Variablen eines Objektes im JSON angezeigt/versteckt werden. Das kann aber nur global für die ganze Klasse eingestellt werden, und nicht für spezielle Use-Cases.

Lösung 2): Mit Type Adaptern kann eingestellt werden, wie welche Daten eines Objektes wann in das JSON geschrieben werden sollen. Dabei muss jedoch für die meisten Use-Cases ein eigener Type Adapter erstellt werden.

Fazit

Während der Entwicklung des Projektes war nicht immer klar, wann welche Daten an die App gesendet wurden bzw. gesendet werden sollen (wegen falschen JOIN-Abfragen, Fetch-Types und Gson Type Adaptern). Das führte zu vielen Exceptions, da versucht wurde auf Daten zuzugreifen, die die App noch nicht erhalten hat.

Automatisches Entladen von GUI Elementen und Variablen

Problem

Wenn bestimmte (GUI-) Objekte nicht mehr zum Anzeigen benötigt werden, werden sie von Android automatisch aus dem RAM-Speicher entladen (Objekte werden null). Das kann passieren, wenn man zwischen den Tabs wechselt oder die App minimiert.

Das Problem dabei ist, dass man erkennen muss, wann die Objekte entladen wurden und wann sie wieder geladen werden sollen. Das ist zusätzlich schwierig, wenn es sich um Datenbank-Daten handelt. Diese sollen nämlich nicht jedes Mal wieder vom Server geholt werden.

Lösung

Manche Datenbank-Daten werden auf dem ROM-Speicher des Android Endgerät mittels SharedPreferences zwischengespeichert. Nur wenn die App erwartet, dass sich die Daten in der Datenbank geändert haben, werden diese Daten wieder vom Server angefragt (z.B. beim Start der App oder beim Erstellen eines neuen Quiz).

Zusätzlich werden manche (kleine) Variablen als statisch gekennzeichnet, damit sie nicht entladen werden. Diese werden verwendet, um zu speichern, ob die Datenbank-Daten nach dem Start der App bereits geladen wurden.

Erweiterungsmöglichkeiten

Realisierung der Kann-Kriterien

- Variable Anzahl an Antworten pro Frage
- Anzahl von Punkten für jede gewählte Antwort frei festlegbar
- Veröffentlichungsdatum für ein Quiz frei festlegen
- Punkte-Bonus an die schnellsten Quiz-Löser vergeben (je früher das Quiz gelöst wird, desto mehr Bonuspunkte bekommt man)
- In der Quizze-Liste hervorheben, ob man das Quiz selbst erstellt hat oder nur daran teilnimmt (bzw. teilgenommen hat)
- Mehr Push-Notifications (wenn alle Teilnehmer das Quiz ausgefüllt haben, neuer Highscore erreicht wurde, etc.)

Freunde-System

Ein Freunde-System ist sinnvoll, wenn man größere Server (mit vielen Usern) verwenden will.

- Hinzufügen von Freunden via Username
- Evtl. Erstellen von Gruppen von Freunden
- Neue Quizze nicht mehr automatisch für alle User freigeben. Anstatt dessen könnte man eines von den Folgenden auswählen:
    - Alle User auf dem Server
    - Alle Freunde
    - Bestimmte Freunde (von der Freundesliste auswählbar)
    - Bestimmte Gruppe(n)

Sonstiges

- Globale Ranglisten
    - Gesamtpunkte-Rangordnung von Usern/Freunden, die die gleichen Quizze wie man selbst ausgefüllt hat
    - Für einen bestimmten Zeitrahmen (letzte Woche, letzten Monat, immer, …)
- Löschen/Archivieren von Quizze (damit Quizze-Liste nicht ewig lang wird)
- (GUI) Verbesserter Ladebildschirm
- Push-Notifications einzeln deaktivierbar
- Speichern von Anmeldedaten in den SharedPreferences („Auto-Login“)