שיפורים מוצעים למערכת ניהול משתמשים

1. הוספת מיקרו-שירות מבוסס SQS לפעולות על יוזרים

יצירת Microservice נפרד שמאזין ל-SQS Queue. כל פעולה (CREATE, UPDATE, DELETE) תישלח לתור לפי type ותבוצע בצורה אסינכרונית כדי להבטיח אמינות ועמידות בפני תקלות.

2. הוספת ניהול שגיאות (Error Handling)

עטיפת כל הרזולברים ב-try/catch, החזרת שגיאות ברורות ל-Client דרך GraphQL errors והפרדה בין שגיאות ידועות לשגיאות מערכת.

3. הוספת לוגים (Logging)

שימוש ב-logger כמו winston או pino, לוגים של בקשות, פעולות קריטיות ושגיאות, כולל requestId או userId לצורך Trace מלא.

4. ולידציות ובדיקות קיום

בדיקת קיום יוזר לפי אימייל ייחודי לפני יצירה, בדיקות קיום לפני עדכון או מחיקה, ושימוש ב-class-validator או Zod לאימות קלט.

5. מחיקה לוגית במקום מחיקה פיזית

שימוש בשדה כמו isDeleted או status לעדכון סטטוס מחוק במקום מחיקה מה-DB. מאפשר שיחזור, עקביות ושמירת היסטוריה.

6. Pagination בשאילתות

הוספת תמיכה בפרמטרי limit ו-offset או cursor-based pagination לצורך שיפור ביצועים וחוויית משתמש.

7. תיעוד API עם GraphQL SDL

הוספת תיאורים ברמת השדות והטייפים בסכמות GraphQL, לשם קריאות ושימוש נכון על ידי צוותים נוספים.

8. הרשאות והרמות גישה

בדיקות הרשאות בשכבת הרזולברים לפי טוקן או session. לדוגמה: רק אדמין יכול למחוק משתמשים.

9. בדיקות אוטומטיות

בדיקות יחידה לפונקציות הלוגיקה, ובדיקות אינטגרציה לזרימות CRUD מלאות על יוזרים.

10. קאשינג לשאילתות כבדות

שקילה של קאשינג לרכיבי קריאה תכופים (כמו GetUsers) ברמת Apollo או Redis.

**🧠 הצורך באיחוד פעולות מול בסיס הנתונים – למניעת Race Conditions**

במערכות מבוזרות או מרובות תהליכים (כמו שירות גרפני שרץ ברפליקציה), קיימת סכנה ממשית למצב של **Race Condition**, שבו שני תהליכים ניגשים לאותו מידע בו-זמנית ויוצרים מצב לא עקבי או לא תקף.

כדי להבטיח **עקיבות (consistency)** של הנתונים – במיוחד בעת כתיבה או עדכון – חובה לאחד פעולות תלויות למסד הנתונים בתוך **טרנזקציה אחת** או פעולה אחת אטומית.

**🧱 דוגמאות מתוך משימת Moonshot**

**✅ 1. createUser – שליפה של עיר ואז יצירת משתמש**

**בעיה:**

• שני תהליכים קוראים את אותה עיר (לדוגמה: TEL\_AVIV) מהטבלה cities, ואז כל אחד יוצר משתמש.

• אם בין הקריאה ליצירה העיר תימחק, או שמזהה העיר ישתנה (נדיר אך אפשרי אם הטבלה מנוהלת ע"י ממשק אחר) – תיווצר תקלה או שייווצר משתמש עם FK לא קיים.

**פתרון:**

• כל הפעולה (שליפת העיר + יצירת המשתמש) תתבצע בתוך **טרנזקציה אחת**.

**דוגמה:**

await AppDataSource.transaction(async (manager) => {

const city = await manager.getRepository(City).findOneBy({ name: input.city });

if (!city) throw new UserInputError("City not found");

const user = manager.getRepository(User).create({

...input,

birthDate: new Date(input.birthDate),

city

});

await manager.getRepository(User).save(user);

});

**✅ 2. updateUser – שליפה של המשתמש, עדכון, שמירה**

**בעיה:**

• תהליך אחד טוען את המשתמש, תהליך שני בינתיים מעדכן אותו → הראשון שומר על מידע לא עדכני (Last Write Wins).

• אם השדה updated\_at מתעדכן אוטומטית – אפשר לאבד עדכון קודם מבלי לדעת.

**פתרון:**

• עדכון יתבצע ישירות דרך query, או עם optimistic locking.

• במקרה הצורך – הוספת version או שימוש בשדה updated\_at להשוואה.

**דוגמה:**

await AppDataSource.transaction(async (manager) => {

const user = await manager.getRepository(User).findOne({ where: { id }, relations: ['city'] });

if (!user) throw new ApolloError("User not found");

const city = await manager.getRepository(City).findOneBy({ name: input.city });

if (!city) throw new UserInputError("City not found");

user.firstName = input.firstName;

user.lastName = input.lastName;

user.birthDate = new Date(input.birthDate);

user.city = city;

await manager.getRepository(User).save(user);

});

**✅ 3. deleteUser – מחיקה ישירה של משתמש**

**בעיה:**

• אם פעולה במקביל מנסה לעדכן את אותו משתמש בזמן שהוא נמחק – אחת מהן תיכשל או תעקוף את השנייה.

• במקרים של פעולות תלותיות (כמו לוגים, רשומות נוספות, או שליחת SNS) – חייבים שכולם ייעשו כיחידה אחת.

**פתרון:**

• גם פעולה פשוטה כמו מחיקה עדיף לארוז בטרנזקציה אם יש תלויות או פעולות נוספות.

• לדוגמה: מחיקה + שליחת התראה ל־SNS → שלא יקרה מצב שהתראה נשלחת אבל המשתמש לא באמת נמחק (או להפך).

**דוגמה:**

await AppDataSource.transaction(async (manager) => {

const result = await manager.getRepository(User).delete(id);

if (!result.affected) throw new ApolloError("User not found or already deleted");

await snsClient.send(new PublishCommand({

TopicArn: process.env.SNS\_TOPIC\_ARN,

Message: `User with ID ${id} was deleted`

}));

});

**🧠 דרכים להמנע:**

**1 שליפת רשומות שאחריהן פעולה תלויה → תמיד בטרנזקציה.**

**2 מעבר על שלבים כמו יצירה ועדכון – יש להגן באמצעות טרנזקציה או אופטימיסטיק לוקינג.**

**3 הוספת פעולות צד כמו שליחת SNS או כתיבה ללוג – רק בתוך טרנזקציה.**

**4 במערכת מבוזרת או רפליקטיבית – כל שינוי מצב (state) צריך להיעשות כאטומי.**