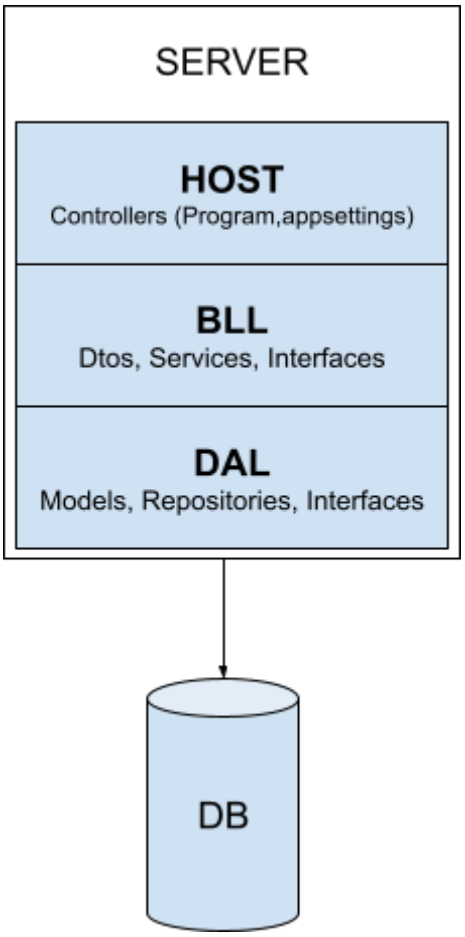
סיכום צד שרת



המטרה הכללית בחלוקה לשכבות היא להפריד בין:

- גישה למסד הנתונים
- הלוגיקה של האפליקציה
- תקשורת עם העולם החיצוני (API)

🧩 DAL – שכבת גישה לנתונים

השכבה שמתעסקת אך ורק במסד הנתונים .

Models

אלו המחלקות מרכיבות ע״י Entity Framework בהתאם לטבלאות במסד.

דוגמה:
משתמש, שאלה, תשובה, קטגוריה...

כל מחלקה = טבלה.

Repositories

לכל טבלה יש מחלקת Repository שמבצעת פעולות מול ה־DB כגון:

- GetAll
- GetById
- Add
- Update
- Delete

הרעיון:
הקונטרולר והשירותים לא מדברים ישירות עם ה־DB, אלא דרך מחלקות אלו בשכבת ה DAL.

למשל:

UserRepository → מטפל בגישה לנתוני משתמשים בלבד

Interfaces

לכל Repository יש ממשק המקביל לו, למשל:

IUserRepository מקביל ל UserRepository

הממשק מגדיר:
מה אפשר לבצע, אבל לא איך.

זה מאפשר:

- החלפת מימוש בלי לשנות קוד
- בדיקות קלות
- תלות בממשק ולא במחלקה

🧠 BL – Business Logic

כאן נמצאות כל הפעולות הלוגיות והבדיקות של המערכת.

DTOS

✔ DTO לא מכיל קשרים וירטואליים
✔ לא מחזיר שדות רגישים
✔ מונע חשיפת מבנה ה־DB האמיתי
✔ מאפשר לשלוט בדיוק אילו נתונים נשלחים החוצה

ובנוסף — יש פעולות המרה:

- מ־model ל־DTO
- מ־DTO ל־model

Services

כאן נמצאת הלוגיקה האמיתית של האפליקציה.

למשל:

- UserService – פעולות שמשתמש רגיל יכול לעשות
- ManagerService – פעולות שמנהל המערכת יכול לבצע

השירותים משתמשים ב־repositories כדי לקבל נתונים ולעבד אותם.

דוגמא לפעולות:

- בדיקת תשובה נכונה או לא
- חישוב ניקוד
- יצירת משתמש חדש
- שליפת שאלות לשחקן

Interfaces (של השירותים)

לכל Service יש ממשק, לדו':

IUserService

IManagerService

🌐 HOST – שכבת ה־API / התקשרות חיצונית

Controllers

כאן מגיעות בקשות מהלקוח (frontend / אפליקציה / דפדפן)

למשל:

user/login משתמש-התחברות

user/answers משתמש-צפיה בתשובות

manager/addQuestion מנהל-הוספת שאלה

הקונטרולר:

- מקבל בקשה HTTP
- מעביר לשירות הרלוונטי
- מחזיר תשובה JSON

חשוב:
הקונטרולר לא מבצע לוגיקה
הוא רק מחבר בין העולם החיצוני לבין ה־Services

Program.cs

הוא מריץ:

✔ חיבור למסד הנתונים

✔ רישום השירותים למערכת

✔ הזרקת תלות

✔ הפעלת Swagger

✔ מיפוי הקונטרולרים

הסבר נוסף בקצרה על EF, DI, אסינכרוניות

Entity Framework (EF)

זהו ORM —מערכת שמתרגמת אובייקטים למחלקות ולטבלאות.

במקום לכתוב:

SELECT * FROM Users WHERE Id = 5

אפשר לכתוב:

db.Users.First(u => u.Id == 5)

יתרונות:
✔ חוסך טעויות ב־SQL
✔ מקצר זמן פיתוח
✔ ממפה טבלאות למחלקות
✔ מבצע שמירת נתונים בצורה נוחה
✔ עוזר עם קשרי טבלאות

🔄 אסינכרוניות (Async)

כשעושים פעולות מול DB — הן לוקחות זמן.

אם לא נשתמש באסינכרוניות — התוכנה תיעצר ותמתין.

במקום:

```
var users = userRepository.GetAll();
```

עדיף:

```
var users = await userRepository.GetAllAsync();
```

יתרונות:

✔ השרת לא נתקע

✔ מאפשר לטפל בבקשות נוספות בזמן ההמתנה

✔ משפר ביצועים

🚰 Dependency Injection – DI

זה הרעיון של הזרקת תלות.

במקום ש־UserService יצור UserRepository בעצמו:

```
var repo = new UserRepository();
```

אנחנו מזריקים לו את התלות:

```
public UserService(IUserRepository userRepo)
```

וה־Program.cs מבצע את ההזרקה:

```
builder.Services.AddScoped<IUserRepository, UserRepository>();
```

יתרונות:

✔ הפרדת שכבות

✔ בדיקות קלות (אפשר להזריק Repository מזויף)

✔ אפשר להחליף מימוש בלי לגעת בקוד