Razvoj Web Aplikacija

Predavanje 3

Spremanje stanja u bazu podataka

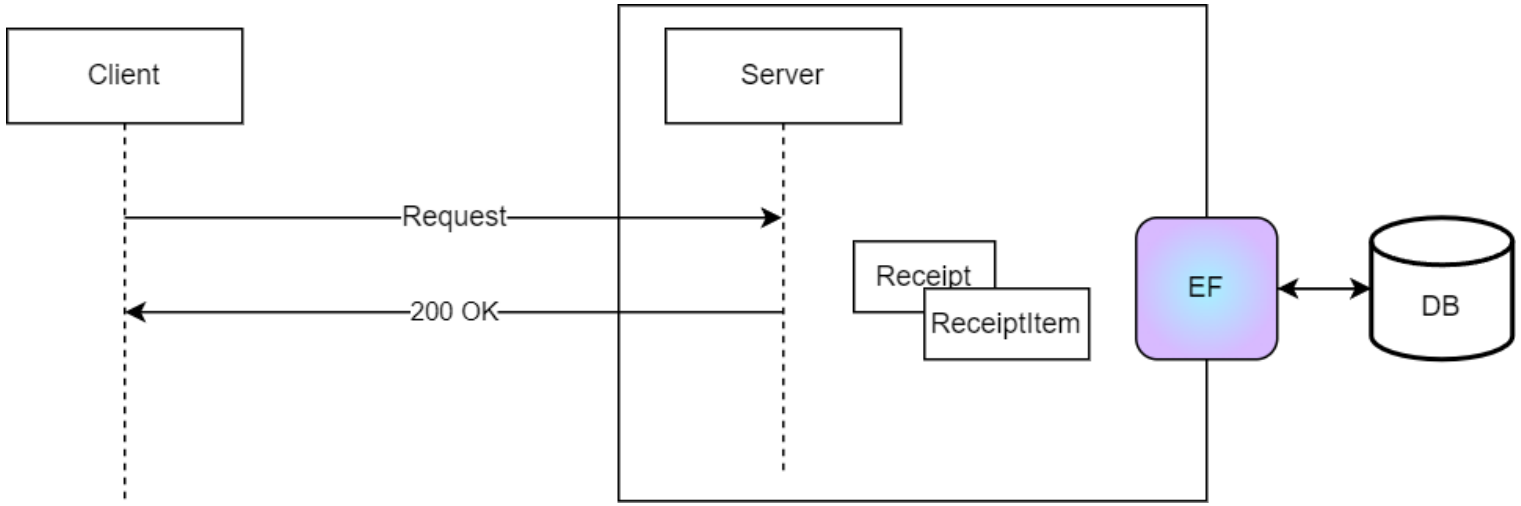
Danas

- Pohrana stanja u bazi podataka
- Entity Framework
- Baza podataka i 3-slojna arhitektura
- Pristup bazi podataka i konfiguracija

Stanje i baza podataka

- Da biste podatke spremili u bazu podataka, potrebno je znati strukturu podataka i relacije
- .Net Core podržava komponentu **ADO.NET** – komponentu za pristup podacima
- ADO.NET je centriran oko baze podataka kao takve, te razumije pojmove tablica, stupaca i redaka
- Te koncepte zatim treba pretvoriti u objekte nad kojima se izvode operacije i konačno vratiti podatke iz tih objekata u tablice baze podataka (mapiranje podataka)
- **Entity Framework** (EF) je komponenta za pristup podacima koja koristi obične objekte za upravljanje podacima – izbjegava se mapiranje podataka
  - Stvarnost je drugačija, ali je još uvijek pojednostavljenje s obzirom na ADO.NET

Stanje u bazi podataka (Entity Framework)



Osnovna pravila

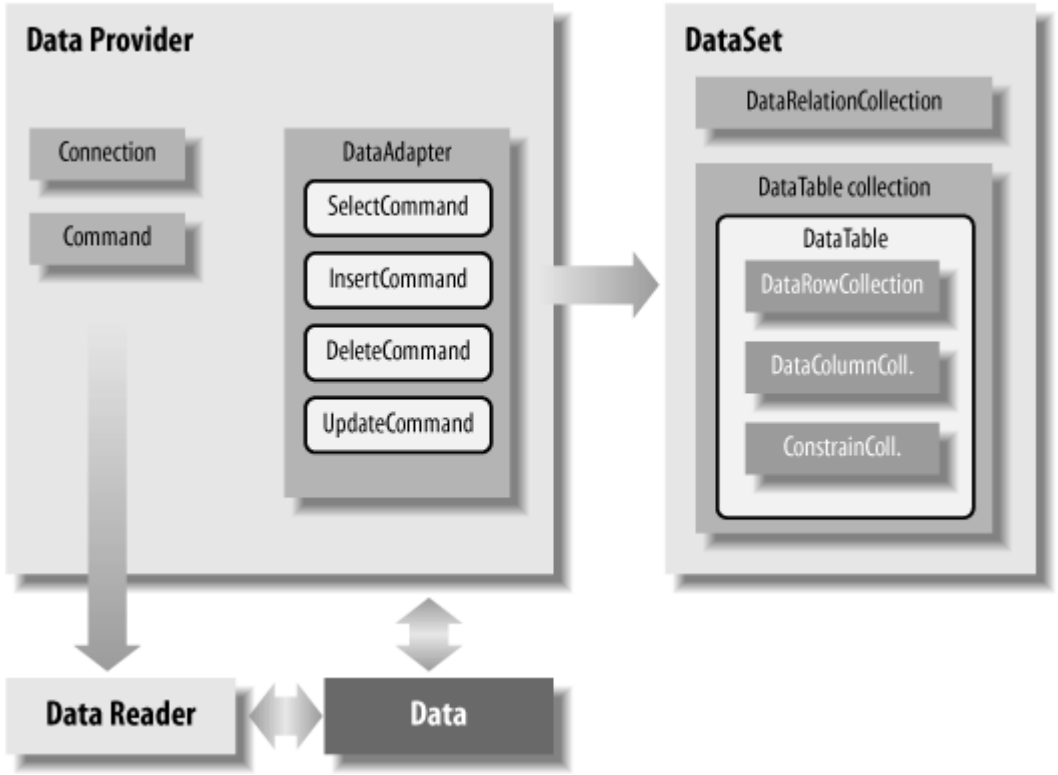
- Poslužiteljska aplikacija povezuje se s bazom podataka kada joj je to potrebno i prekida vezu što je prije moguće
  - U našoj aplikaciji postoji potreba za connection stringom koji mora biti podesiv, a ne hardkodiran!
    ➔ appsettings.json, odjeljak ConnectionString
- *Web API:* podaci se dohvaćaju pomoću HTTP GET zahtjeva, podaci se mijenjaju pomoću HTTP POST/PUT/DELETE zahtjeva
- *MVC:* podaci se dohvaćaju pomoću HTTP GET zahtjeva, podaci se mijenjaju pomoću HTTP POST zahtjeva putem obrasca ili putem HTTP POST/PUT/DELETE korištenjem Ajax tehnike
- Kada poslužitelj dohvaća podatke, koristi bazu podataka
- Kada poslužitelj modificira podatke, pohranjuje ih u bazu podataka

Tehnologije

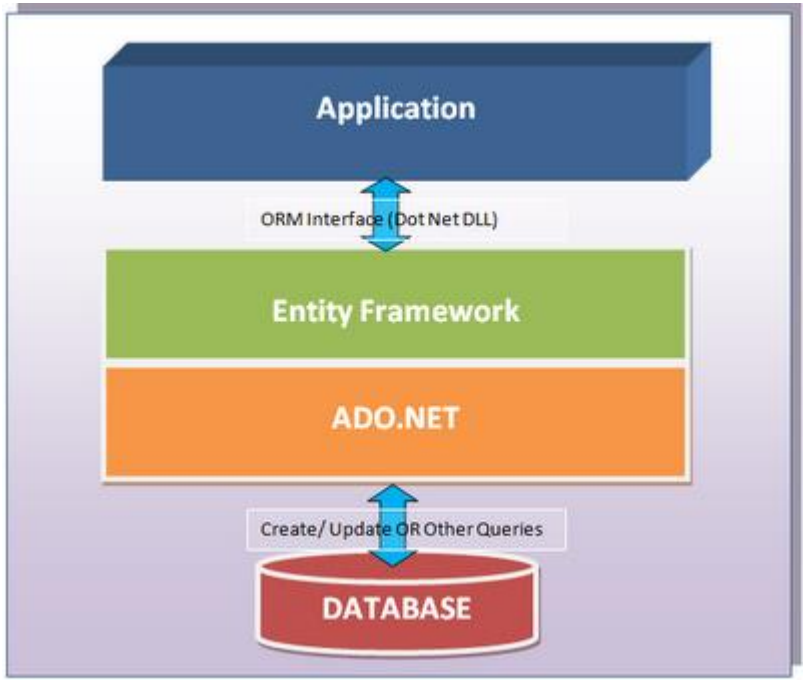
- U ASP.NET Core-u možemo koristiti sljedeće tehnologije za pristup bazi podataka:
  - ADO.NET
  - Entity Framework (EF) – ovisi o ADO.NET
- ADO.NET radi s tablicama, stupcima, recima, ključevima i drugim konceptima centriranim oko baze podataka, preslikavajući ih izravno u radno okruženje
- EF koristi OOP koncepte (objekte) za predstavljanje podataka u bazi
  - Polje podataka u bazi je svojstvo EF objekta
  - Redak podataka u bazi je instanca EF objekta
  - Tablica je predstavljena kao kolekcija u EF-u
  - Sama baza podataka (ili njezin dio) je predstavljena kontekstom baze podataka

- Obuhvatit ćemo osnove Entity Framework pristupa bazi podataka
- Koristit ćemo EF Core varijantu prilagođenu za rad u .Net Core okruženju

ADO.NET i EF – usporedba



USES



Entity Framework je Object/Relationship Mapper (ORM) temeljen na ADO.NET technology

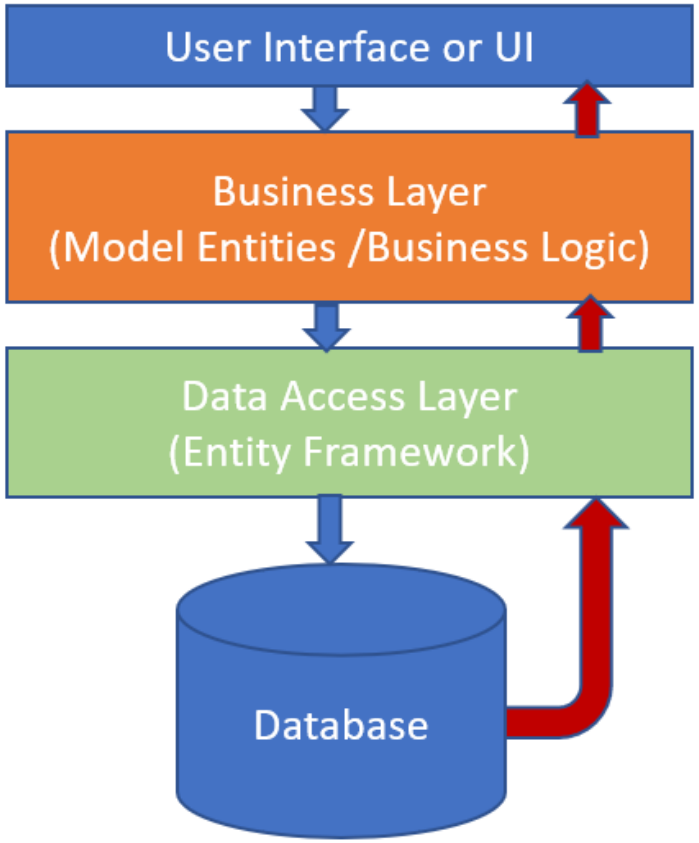
Entity Framework koncepti

- **Entitet** – središnji koncept, enkapsulacija podataka (iz baze podataka) korištenjem klase (predstavlja strukturu baze podataka) i klasa instanci (predstavljaju zapise baze podataka)
- **Database context** – apstraktno predstavlja bazu podataka ili njen dio, u smislu da sadrži skupove entiteta (tablice iz baze podataka) i odnose (relacije) između njih
- **Code first** – mogućnost programskog stvaranja strukture podataka i primjene te strukture na bazu podataka tako da se u njoj stvarno kreiraju tablice i relacije između njih
- **Database first** – mogućnost korištenja postojeće baze podataka u kodu, ali na način da se ne mijenja struktura tablica i relacija u bazi (ovo ćemo koristiti na vježbama)
- **POCO** – skraćeno od Plain Old Class Object, klasa koja ne nasljeđuje drugu klasu

Arhitektura sustava

- Danas su dostupne mnoge vrste arhitektura sustava (2-slojna, 3-slojna, "onion" arhitektura, monolit, mikroservisna arhitektura)
- Svaka od njih ima svoje prednosti i mane
- 2-slojna arhitektura: korisničko sučelje odvojeno je od prostora za pohranu (baza podataka)
  - Korisničko sučelje (npr. statična web stranica)
  - Sloj za procesiranje i pohranu (naša Web API aplikacija)
- 3-slojna arhitektura
  - Dodatno razdvajanje aplikacije na slojeve
  - Sloj za pohranu (često nazvan DB sloj ili sloj za pristup podacima ili samo DAL)
  - Sam Entity Framework može se smatrati slojem pohrane
  - Srednji sloj (poslovni sloj ili BL)
  - BL radi s logikom koja je bliža onome što smatramo stvarnošću od one koja se nalazi u bazi podataka
  - *Primjer: treba li nam polje UpdatedAt u BL-u ili je to samo dio "handlinga" na niskoj razini?*

3-slojna arhitektura



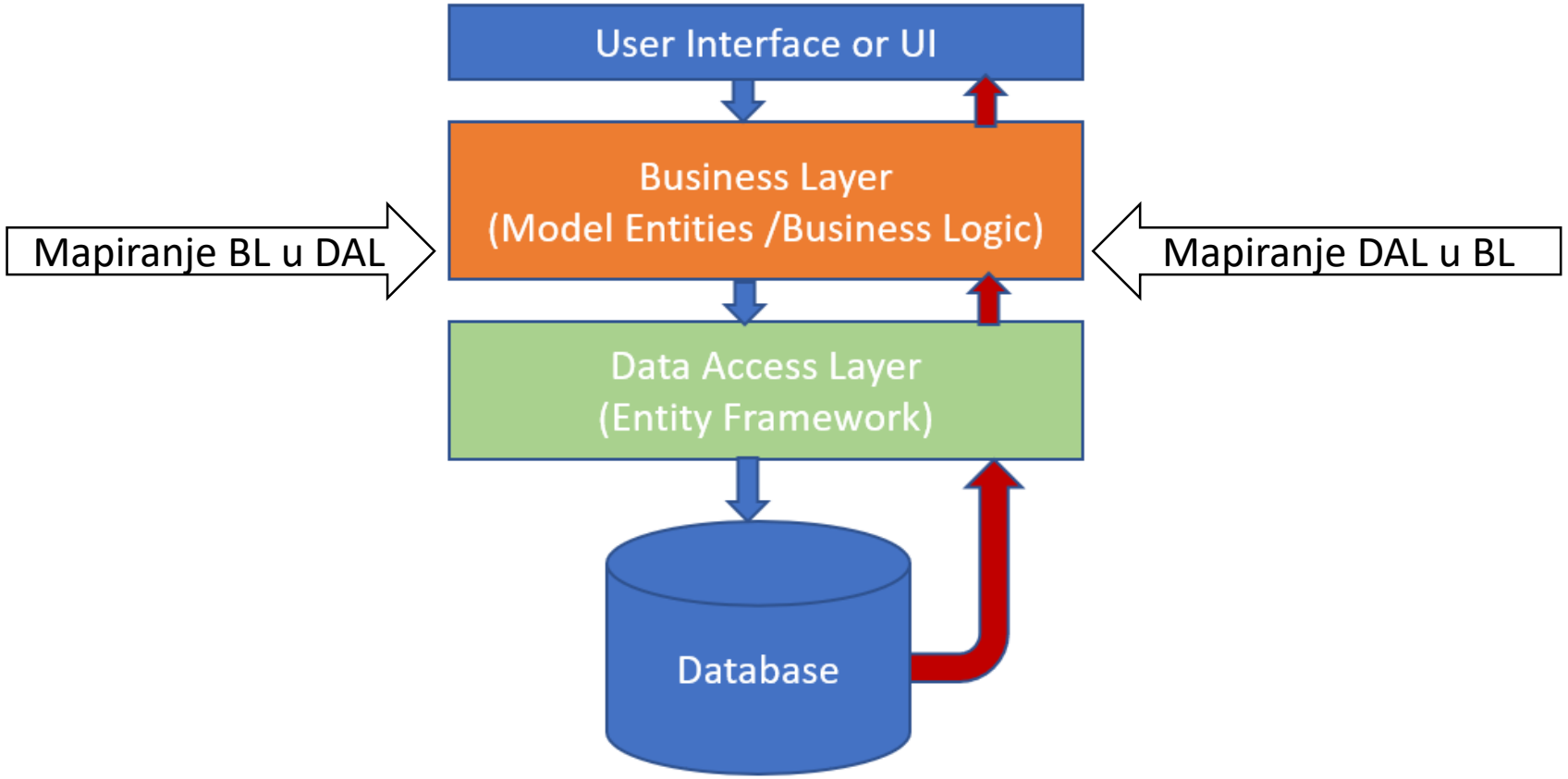
3-slojna arhitektura: Poslovni sloj

- Koristi se skraćenica BL (engl. Business layer)
- Karakteristično rukovanje BL-om prilikom čitanja podataka je uzimanje DAL modela (objekata koji predstavljaju redove baze podataka) i stvaranje BL modela iz DAL modela
- Također, prilikom pohrane podataka, BL mora stvoriti DAL modele iz BL modela
- *Primjer: svi računi s podacima o tome kada su stvoreni pohranjuju se za korisnika u bazi podataka. Također, u bazi podataka postoji saldo tekućeg računa za korisnika. Ali nema podataka kakav je bio saldo nakon što je svaki račun plaćen.*

3-slojna arhitektura: Sloj pristupa podacima

- Koristi se skraćenica DAL (engl. Data Access Layer)
- Primjer 1
  - Baza podataka sadrži podatke o konkretnim financijskim vrijednostima za pojedini mjesec i ti su podaci dostupni u DAL sloju
  - Baza podataka ne sadrži podatke o saldu, ali se ti podaci mogu izračunati u BL modelu
- Primjer 2
  - DAL (baza podataka) ima *CreatedAt* podatke o računu, ali BL ih ne koristi i ne pohranjuje ih
- Nije neuobičajeno da se mnogo informacija jednostavno kopira iz BL u DAL i obrnuto (količina i vrijednost artikla na računu, ukupni iznos računa)
- Proces pretvaranja BL-a u DAL i obrnuto nazivamo mapiranje

3-slojna arhitektura: Mapiranje



Connection string

- Connection string opisuje bazu podataka s kojom se želimo povezati
- Sastoji se od niza tokena u formatu *postavka = vrijednost*
- Important settings
  - **Server** – naziv i instanca poslužitelja baze podataka (npr. SQL Server)
  - **Database** – naziv baze podataka
  - **User Id** – korisničko ime
  - **Password**
  - Umjesto korisničkog ID-a i lozinke može stajati **Trusted_Connection=true**, ako poslužitelj baze podataka vjeruje korisniku u kontekstu integrirane sigurnosti (Windows korisnik)
  - U slučaju da klijent koristi integriranu sigurnost a server nije konfiguriran da koristi certifikat, potrebno je dodati **TrustServerCertificate=True** da bi klijent zaobišao normalan mehanizam uspostave povjerenja, validaciju certifikata – ovo se ne preporuča na produkcijskom serveru

Connection string

- Dodatne postavke i vrijednosti koje koriste novije verzije Entity Frameworka
  - **MultipleActiveResultSets=True** – dopusti u istoj konekciji izvršavanje više serija (engl. batch) naredbi
- Primjeri:

```
Server=.\\MSSQL2022; Database=Task06; Trusted_Connection=True;
MultipleActiveResultSets=True; TrustServerCertificate=True;

Server=my-sql-server; Database=MyDatabase; User Id=User1234;
Password=abcd9876!#$.; MultipleActiveResultSets=True;
TrustServerCertificate=True
```

Connection string postavka

- Moramo dopustiti promjenu connection stringa
  - produkcijska baza podataka,
  - baza podataka za testiranje,
  - razvojna baza podataka...
- Koristimo appsettings.json za pohranu connection stringa
- Connection string koristi DI kontejner prilikom stvaranja DB konteksta

Primjer:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "Task06ConnStr": "server=.\\MSSQL2022;Database=Task06;Trusted_Connection=True;TrustServerCertificate=True"
  }
```

```
// Program.cs
builder.Services.AddDbContext<Task06Context>(options =>
{
  options.UseSqlServer("name=ConnectionStrings:Task06ConnStr");
});
```

Hvala na pažnji!