ToDo Web Application

1. Osnovno o aplikaciji

ToDo Application predstavlja aplikaciju za upravljanje zadacima. Njena osnovna svrha jeste pruzanje korisnicima mogucnosti da kreiraju svoje taskove unutar aplikacije, da upravljaju njima i da vode efikasno evidenciju o tome koji su task izvrsili, koji task je isteko (ako ga nisu oznacili kao izvrsen) i koliko taskova im je preostalo.

Aplikacija podrzava osnovne **CRUD (Create, Read, Update i Delete) operacije,** a osim toga podrzane su uloge **administratora** i **customera.**

U okviru aplikacije realizovano je i filtriranje taskova, koje korisnicima omogucava da filtriraju svoje taskove na osnovu odredjenih svojstava, kao sto su naziv taska, datum, prioritet, kategorija i status.

2. Tehnologije koriscene u razvoju

- **ASP .NET Web API (.NET 9):** koriscen je za implementaicju **backend** dela aplikacije. Web API pruza **RESTful** servis koji omogucava kreiranje, citanje, azuriranje i brisanje podataka iz baze
- **React:** koriscen je za implementaciju **frontend** dela aplikacije. Koristio sam ga jer pruza brzu i dinamicnu izradu UI – a.
- **MSSQL:** relaciona baza podataka za cuvanje informacija o korisnicima i taskovima.
- **Cloudinary:** koriscen je za skladistenje profilnih slika korisnika.

3. Arhitektura sistema

Arhitektura backend-a organizovana je kao **slojevita (layered) arhitektura.** Podeljena je na 4 glavna sloja + infrastrukturu. (EL, DAL, BLL i API sloj):

1. EL (Entities Layer /Shared Layer):
    - U okviru ovog sloja nalaze se modeli koji predstavljaju entitete u bazi podataka, kao i DTO klase. DTO koristim da smanjim kolicinu podataka koji se prenose izmedju klijenta I servera.
    - **TextNormalizer:** metoda koja mi sluzi za da normalizuje string za pretragu: ukloni (ć, č, đ, š, ž...), svede tekst na mala slova. Nalazi se unutar EL jer je genericka metoda, moze se iskoristiti gde god mi treba normalizacija teksta (ne samo za pretragu taskova).

2. DAL (Data Access Layer):

- Sadrzi **ApplicationDbContext** i njegov factory **ApplicationDbContextFactory:** koristi **factory pattern** (da klasa koja koristi objekat ne mora da zna kako se tacno objekat kreira), koji je biran za kreiranje migracija.
- **Repository pattern:** ako zelim npr da dobijem sve taskove, ne pisem SQL direktno po svim tabelama, nego pozovem metodu repozitorijuma koja to radi. Imam:
- **Repository<T>** koji je genericki repozitorijum I implementira osnovne CRUD operacije za bilo koji entitet T. Ideja je da **ne pisem svaki put iste metode za svaku tabelu**, vec da ih **generalizujem.**
    - **TaskRepository:** repozitorijum za entitet ToDoTask i **nasledjuje Repository** i implementira dodatne metode koje su specificne za taskove. Omogucava naprednu logiku (u ovom slucaju filtriranje npr.)
- **UnitOfWork:** koordinise rad vise repozitorijuma kroz jednu transakciju. Znaci ako nesto ne uspe, baza ostaje cista.
    - Odgovoran je za komunikaciju sa bazom i za **centralizovani pristup repozitorijumima.**
    - Funkcionise kao **wrapper,** umesto da instanciram svaki repozitorijum posebno, **UoW** sadrzi sve potrebne repozitorijume kao properties. Tako da, ako mi nekada zatrebaju vise repozitorijuma u okviru iste logike, mogu da iskoristim samo jednu instancu UoW.
- **Extensions:** nove metode koje dodajem postojecim klasama bez da menjam njihov kod. Koristim extension da bi pretraga i filter taskova bili odvojeni od glavnog repozitorijuma, ali i dalje dostupni kao da je prirodna metoda IQueryable<ToDoTask>. Sadrzi
    - **SeedDataExtensions** koji se pokrece svaki put kada se aplikacija startuje, proverava da li podaci koje unosi postoje, ako ne unosi ih (test podaci)
    - **ApplySearch:** extension metoda na IQueryable<ToDoTasks>. Koristi se da primenim filtere ako je korisnik uneo neke.

3. BLL (Business Logic Layer):

    - Sadrzi servise: **AuthService** (upravljanje korisnicima); **TaskService** (upravljanje taskovima); **EmailService** (reset lozinke); **CloudinaryService** (upload slike).
    - Svaki servis ima svoj **interfejs** (definise sta servis mora da radi) i **implementaciju** (definise kako servis to radi) – **polimorfizam** I **apstrakcija**
    - BLL sadrzi **poslovnu logiku**

4. API sloj:
    - Sadrzi **kontrolere** koji primaju HTTP zahteve, validiraju ih i pozivaju odgovarajuci servis iz BLL – a I vracaju odgovor na kraju.
    - **Kontroleri** su ulazna tacka sistema i njihova logika je u BLL
5. Frontend
    - Koristim **RTK Query** koji je deo **Redux Toolkit – a** I predstavlja **data – fetching** I **caching biblioteku.** Resava probleme povlacenja/slanja podatka sa/na server.
    - Svaki **endpoint** moze biti **query:** GET, dohvatanje podataka sa servera; **mutation:** POST, PUT, DELETE, menja podatke
    - **Slices:** koristim ih da cuvam sve podatke koji su vezani za UI ili sesiju. Imaju **reducers:** funkcije koje definisu kako se stanje aplikacije menja kao odgovor na akcije.
    - **Store:** centar Redux arhitekture. Sadrzi i **middleware** funkcije koje omogucavaju modifikaciju akcija pre nego sto one stignu do reducer-a.

4. Kako se aplikacija pokrece (lokalno):

1. **Preuzimanje projekta sa Github-a**:
    - Pomocu komande: **git clone** https://github.com/DusanM998/ToDoApplication.git
    - Pravi lokalnu kopiju repozitorijuma na racunaru
2. **Backend** deo nalazi se u folderu ToDoApp, potrebno je otvoriti solution koji se nalazi u okviru tog foldera (**ToDoApp.slnx**).
3. **Primena migracija.**
    - Posto migracije vec postoje, potrebno ih je samo primeniti, tako sto se pozicionirate u okviru **DAL** (jer se tu nalazi ApplicationDbContext) i migracije. (cd DAL)
    - Zatim je potrebno pokrenuti komandu unutar Developer Power Shell ili Package Manage Console: **dotnet ef database update** kako bi se kreirala baza i primenile migracije
4. **Pokretanje backend servera:**
    - Izabrati ToDoApp projekat i pokrenuti ga komandom dotnet run
    - Backend je metode mogu se testirati na: **https://localhost:7070/scalar/v1**
5. **Frontend**
    - Frontend deo aplikacije nalazi se u folderu ToDoAppFrontend
    - Potrebno je u Visual Studio Code otvoriti istoimeni workspace koji se nalazi u tom folderu

- Zatim otvoriti terminal i pozicionirati se u **to-do-app** (**cd to-do-app**)
- Potrebno je zatim instalirati sve dependencies: **npm install –save –legacy-peer-deps** (--legacy-peer-deps se koristi da se izbegnu konflikti izmedju pojedinih verzija biblioteka)
- Frontend se pokrece default na: **http://localhost:5173/**

5. Test podaci

Aplikacija koristi automatsko seedovanje podataka prilikom prvog pokretanja, sto znaci da se kreiraju automatski dva korisnika (**Admin** i **Customer**), kao I njihovi pocetni taskovi.

Da bi se to kreiralo potrebno je samo pokrenuti aplikaciju lokalno, a metod **EnsureSeedDataAsync** se izvrsava pri startovanju aplikacije i seeduje podatke u bazu.

Test kredencijali:

(Demo korisnici su unapred kreirani na serveru i mogu se odmah koristiti za prijavu)

- **Admin:**
  - username: admin@test.com
  - password: Admin123!
- **Customer:**
  - username: customer@test.com
  - password: Customer123!

6. Kljucne odluke i kompromisi

1. **Admin:**

- **Odluka:** Iako je mozda i suvisno da u ovakvom tipu aplikacije postoji administrator, odlucio sam da ga dodam zbog demonstracije kako se kreiraju korisnici sa rolama I kako se moze kontrolisati pristup razlicitim stranicama.
- **Admin** ima mogucnost samo da pregleda sve taskove za sve korisnike, kao i da vidi sve registrovane korisnike u okviru aplikacije

2. **Optimizacija Query – a:**

- Umesto da se svi podaci ucitavaju u memoriju, koristi se **IQueryable** za filtriranje u bazi.
- Benefiti toga su da se upiti izvrsavaju direktno u bazi cime se samo filtrirani rezultati vracaju i memorija aplikacije nije opterecena podacima koji mozda u tom trenutku nisu potrebni.

3. **Paginacija:**

- Kako bih izbego da stranice budu predugacke i da korisnik ne bi morao da skroluje kroz njih “beskonacno” ako ima previse taskova, implementirao sam paginacju tako da vracam samo npr. 6 (u mom slucaju), taskova po stranici.