EDUNOVA
ŠKOLA INFORMATIKE I MANAGEMENTA

Program obrazovanja za stjecanje mikrokvalifikacije web programiranje

GRUPA WP7

Dokumentacija završne provjere

Platforma za organizaciju rada (ZORA)

git URL aplikacije: `https://github.com/tnebes/zora`

URL produkcije: `https://zora.draucode.com`

Tomislav Nebes

Osijek, ožujak 2025.

Sadržaj

1 Uvod

Platforma za organizaciju rada, nadalje ZORA, je web aplikacija koja omogućava organizaciju i planiranje posla općenito. Ona omogućava korisnicima da dodaju zadatke koje kasnije na temelju informacija dostupnih u zadaćama (*eng. tasks*) mogu biti obavljeni. Motivacija za kreiranje ZORE je postojanje Adobe proizvoda naziva Workfront koji je također alat za organizaciju posla, s time da posjeduje značajno više funkcionalnosti nego što ZORA trenutno ima.

Glavne funkcionalnosti ZORE uključuju kreiranje zadataka, dodjeljivanje zadataka korisnicima, praćenje napretka zadataka te generiranje prikaza stanja zadataka. Aplikacija bi posebno bila korisna za timove koji rade na projektima i trebaju organizirati svoje zadatke i rokove.

Iz tehnološke perspektive, ZORA je izgrađena pomoću platforme ASP.NET Core 9.0 s korištenjem Entity Framework (EF) Core za komunikaciju s bazom podataka Microsoft SQL Server. Korisničko sučelje je izgrađeno pomoću HTML, CSS i TypeScript. Dakle, *eng. frontend* je izgrađen pomoću Angular razvojnog okvira (eng. framework), a *eng. backend* pomoću ASP.NET Core.

ZORA ima za cilj omogućiti jednostavno i efikasno upravljanje zadacima, s mogućnošću prilagodbe za različite veličine timova i projekata. Planovi za daljnji razvoj uključuju implementaciju novih značajki poput integracije s drugim alatima, povezivanja zadataka s projektima, naprednih alata za praćenje napretka te optimizacije za mobilne uređaje. S tehničke strane, prioriteti su unapređenje performansi, modernizacija korisničkog sučelja i poboljšanje mobilne kompatibilnosti.

2 Vježbe

Rad na ZORA aplikaciji se vršio postepeno tijekom cijelog edukacijskog smjera. Vježbe su se odnosile na razvoj ERA dijagrama, *backend* aplikacije te *frontend* aplikacije. U sljedećim poglavljima će biti opisane vježbe i njihov rezultati.

2.1 ERA dijagram (Vježba 1)

Cilj prve vježbe je bio napisati *Structured Query Language*, nadalje SQL, skriptu koja je omogućila kreiranje baze podataka. U svrhu provjere ispravnosti skripte je korišten Microsoft SQL Server Management Studio (SSMS) pomoću kojeg je izvršena vizualna provjera kreirane baze podataka kao i unos testnih podataka u tablice.

Baza podataka je kreirana u dvije faze. Prva faza uključuje inicijalnu postavku baze podataka i korisničkih dozvola:

```
CREATE DATABASE zora;
ALTER DATABASE zora COLLATE Latin1_General_100_CI_AS_SC_UTF8;
CREATE LOGIN [zora_service] WITH PASSWORD =
   'your_strong_password_here';
CREATE USER [zora_service] FOR LOGIN [zora_service];
```

Druga faza uključuje kreiranje tablica i relacija. Glavne tablice uključuju:

- `zora_users` - pohranjuje podatke o korisnicima
- `zora_roles` - definira uloge u sustavu
- `zora_work_items` - centralna tablica za upravljanje zadacima
- `zora_projects` i `zora_tasks` - specifične vrste radnih stavki

Primjer kreiranja tablice za korisnike:

```
CREATE TABLE zora_users (
    id BIGINT IDENTITY(1,1) PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    created_at DATETIME2 DEFAULT GETDATE(),
    deleted BIT DEFAULT 0 NOT NULL,
    CONSTRAINT CHK_User_Email CHECK (email LIKE '%_@__%.__%')
);
```

Baza podataka implementira soft delete mehanizam kroz `deleted` zastavicu u većini tablica. Relacije između tablica su definirane kroz vanjske ključeve, što osigurava referencijalni integritet podataka. Primjer relacije između korisnika i uloga:

```
CREATE TABLE zora_user_roles (
    user_id BIGINT NOT NULL,
    role_id BIGINT NOT NULL,
    PRIMARY KEY (user_id, role_id),
    FOREIGN KEY (user_id) REFERENCES zora_users(id),
    FOREIGN KEY (role_id) REFERENCES zora_roles(id)
);
```

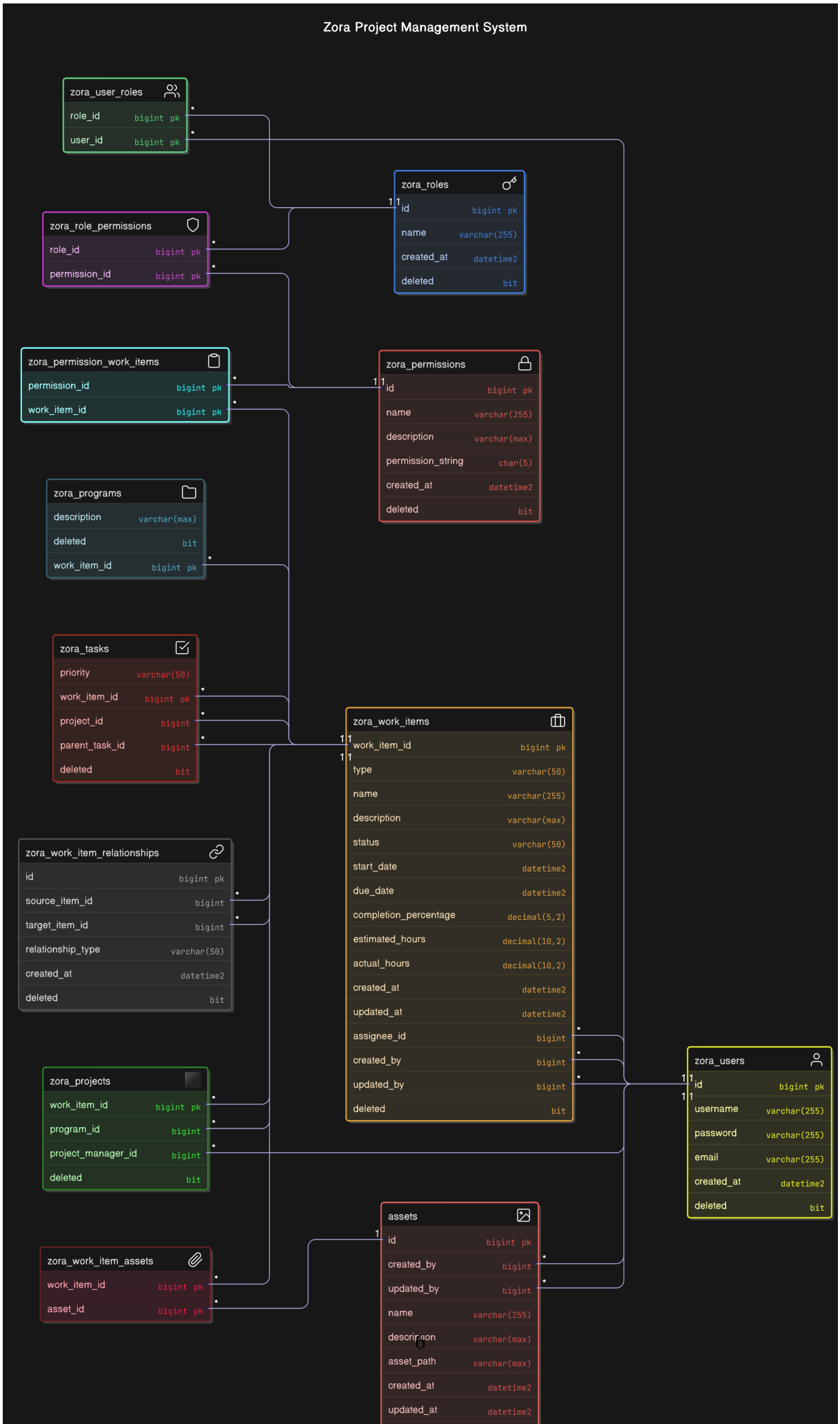
Nakon kreiranja strukture, dodani su početni podaci, uključujući administratorskog korisnika i osnovne uloge:

```
INSERT INTO zora_users (username, password, email)
VALUES ('tnebes',
   '$2a$12$u8XHrlyC7Y1X96jTawebd.Li3uK9Z.PImu6Vy9KD.rtMNnmz3SKY.',
        'tnebes@draucode.com');
INSERT INTO zora_roles (name) VALUES ('Admin');
```

Baza podataka je optimizirana kroz indekse i ograničenja, što osigurava brz pristup podacima i valjanost unosa. Primjer optimizacije kroz indekse:

```
CREATE UNIQUE INDEX IX_User_Username ON zora_users(username) WHERE
   deleted = 0;
CREATE UNIQUE INDEX IX_User_Email ON zora_users(email) WHERE deleted
   = 0;
```

Ova implementacija omogućava skalabilnu i sigurnu pohranu podataka za ZORA aplikaciju, s mogućnošću daljnjeg proširenja i optimizacije.


Zora Project Management System
zora_user_roles
role_id bigint pk
user_id bigint pk
zora_roles
id bigint pk
name varchar(255)
created_at datetime2
deleted bit
zora_role_permissions
role_id bigint pk
permission_id bigint pk
zora_permission_work_items
permission_id bigint pk
work_item_id bigint pk
zora_permissions
id bigint pk
name varchar(255)
description varchar(max)
permission_string char(5)
created_at datetime2
deleted bit
zora_programs
description varchar(max)
deleted bit
work_item_id bigint pk
zora_tasks
priority varchar(50)
work_item_id bigint pk
project_id bigint
parent_task_id bigint
deleted bit
zora_work_items
work_item_id bigint pk
type varchar(50)
name varchar(255)
description varchar(max)
status varchar(50)
start_date datetime2
due_date datetime2
completion_percentage decimal(5,2)
estimated_hours decimal(10,2)
actual_hours decimal(10,2)
created_at datetime2
updated_at datetime2
assignee_id bigint
created_by bigint
updated_by bigint
deleted bit
zora_work_item_relationships
id bigint pk
source_item_id bigint
target_item_id bigint
relationship_type varchar(50)
created_at datetime2
deleted bit
zora_projects
work_item_id bigint pk
program_id bigint
project_manager_id bigint
deleted bit
zora_users
id bigint pk
username varchar(255)
password varchar(255)
email varchar(255)
created_at datetime2
deleted bit
assets
id bigint pk
created_by bigint
updated_by bigint
name varchar(255)
description varchar(max)
asset_path varchar(max)
created_at datetime2
updated_at datetime2
zora_work_item_assets
work_item_id bigint pk
asset_id bigint pk

ERA dijagram prikazuje strukturu baze podataka ZORA aplikacije. Svaka tablica ima svoje specifične atribute i relacije s drugim tablicama:

- **zora_users** sadrži osnovne podatke o korisnicima sustava:
  - jedinstveni identifikator, korisničko ime, lozinku i email
  - vremenske oznake kreiranja i soft-delete zastavicu
  - povezuje se s ulogama kroz `zora_user_roles`
- **zora_roles** i **zora_permissions** definiraju sigurnosni model:
  - uloge (`roles`) definiraju skupine korisnika
  - dozvole (`permissions`) koriste 5-bitni string za definiranje prava[1]
  - veza many-to-many između uloga i dozvola kroz `zora_role_permissions`
- **zora_work_items** je centralna tablica sustava:
  - sadrži zajedničke atribute za sve vrste radnih stavki
  - prati status, napredak, procijenjene i stvarne sate rada
  - povezuje se s korisnikom kojem je zadatak dodijeljen (`assignee_id`)
  - prati tko je kreirao i ažurirao stavku (`created_by`, `updated_by`)
- **zora_programs**, **zora_projects** i **zora_tasks** definiraju planiranu hijerarhijsku strukturu[2]:
  - programi će moći sadržavati više projekata, omogućavajući grupiranje povezanih projekata
  - projekti će pripadati programu i imati definiranog voditelja projekta
  - zadaci su trenutno implementirani kao osnovne radne jedinice, s planiranom podrškom za hijerarhijsku organizaciju

[1] Sustav dozvola koristi 5-bitni string gdje svaka pozicija predstavlja određenu razinu pristupa: `00000` (None), `00001` (Read), `00010` (Write), `00100` (Create), `01000` (Delete), `10000` (Admin). Na primjer, korisnik s dozvolom `00101` ima prava čitanja i kreiranja. Implementirano kroz `PermissionFlag` enum u C# kodu gdje su dozvole definirane kao bitwise flags: None = 0, Read = 1, Write = 2, Create = 4, Delete = 8, Admin = 16.

[2] Hijerarhijska struktura programa i projekata je dio planiranog proširenja sustava. Trenutna implementacija podržava osnovne zadatke, dok će buduća nadogradnja omogućiti potpunu hijerarhijsku organizaciju rada.

- **zora_work_item_relationships** omogućava fleksibilno povezivanje:
  - definira veze između različitih radnih stavki
  - svaka veza ima svoj tip i vremenske oznake
  - omogućava praćenje zavisnosti između zadataka
- **assets** i **zora_work_item_assets**:
  - pohranjuju podatke o datotekama i resursima
  - omogućavaju povezivanje resursa s radnim stavkama
  - prate metapodatke o resursima i njihovim izmjenama

Sve tablice implementiraju `soft-delete` mehanizam kroz `deleted` zastavicu, što omogućava očuvanje povijesti podataka. Vremenske oznake (`created_at`, `updated_at`) i reference na korisnike koji su izvršili promjene osiguravaju mogućnost praćenja promjena u sustavu. Indeksi su postavljeni na ključne stupce koji se često koriste u pretraživanju, što optimizira performanse upita.

2.2 Backend (Vježba 2)

Backend aplikacije je implementiran kao REST API koristeći ASP.NET Core 9.0. API je organiziran u nekoliko glavnih područja koja pokrivaju različite funkcionalnosti sustava.

2.2.1 Opis API-ja

U ovome poglavlju će biti opisan API za backend aplikaciju. Za svaku funkcionalnost je definiran odgovarajući HTTP endpoint i njegova svojstva. U nastavku slijede sve funkcionalnosti koje API podržava, grupirane po sličnosti:

- **Autentifikacija i Autorizacija**:
  - `/api/v1/authentication/token` - generira JWT token za prijavljenog korisnika
  - `/api/v1/authentication/check` - provjerava valjanost postojećeg tokena
  - `/api/v1/authentication/current-user` - dohvaća podatke trenutnog korisnika
  - `/api/v1/authorisation/is-authorised` - provjerava ima li korisnik određenu dozvolu
  - `/api/v1/authorisation/is-admin` - provjerava je li korisnik administrator
- **Upravljanje Zadacima**:
  - `GET /api/v1/tasks` - dohvaća listu zadataka s paginacijom
  - `POST /api/v1/tasks` - kreira novi zadatak
  - `PUT /api/v1/tasks/{id}` - ažurira postojeći zadatak
  - `DELETE /api/v1/tasks/{id}` - briše zadatak (soft delete)
  - `POST /api/v1/tasks/{id}/assign` - dodjeljuje zadatak korisniku
  - `POST /api/v1/tasks/{id}/complete` - označava zadatak kao završen
  - `GET /api/v1/tasks/find` - napredno pretraživanje zadataka po više kriterija
  - `GET /api/v1/tasks/search` - brzo pretraživanje zadataka po ključnoj riječi
  - `GET /api/v1/tasks/priority/{userId}` - dohvaća prioritetne zadatke korisnika

  - `GET /api/v1/tasks/{id}/assets` - dohvaća resurse povezane sa zadatkom

- **Upravljanje Korisnicima**:
  - `GET /api/v1/users` - dohvaća listu korisnika s paginacijom
  - `POST /api/v1/users` - kreira novog korisnika
  - `PUT /api/v1/users/{id}` - ažurira podatke postojećeg korisnika
  - `DELETE /api/v1/users/{id}` - deaktivira korisnika (soft delete)
  - `GET /api/v1/users/search` - brzo pretraživanje korisnika po ključnoj riječi
  - `GET /api/v1/users/find` - napredno pretraživanje korisnika po više kriterija

- **Upravljanje Ulogama i Dozvolama**:
  - `GET /api/v1/roles` - dohvaća sve dostupne uloge
  - `POST /api/v1/roles` - kreira novu ulogu
  - `GET /api/v1/roles/search` - brzo pretraživanje uloga po nazivu
  - `GET /api/v1/roles/find` - napredno pretraživanje uloga po kriterijima
  - `GET /api/v1/permissions` - dohvaća sve dostupne dozvole
  - `POST /api/v1/permissions` - kreira novu dozvolu
  - `GET /api/v1/permissions/search` - brzo pretraživanje dozvola po nazivu
  - `GET /api/v1/permissions/find` - napredno pretraživanje dozvola

- **Upravljanje Resursima**:
  - `GET /api/v1/assets` - dohvaća listu resursa s paginacijom
  - `POST /api/v1/assets` - učitava novi resurs
  - `PUT /api/v1/assets/{id}` - ažurira metapodatke resursa
  - `DELETE /api/v1/assets/{id}` - briše resurs (soft delete)
  - `GET /api/v1/assets/{id}/download` - preuzima resurs
  - `GET /api/v1/assets/{id}/preview` - dohvaća pregled resursa

  - `GET /api/v1/assets/search` - brzo pretraživanje resursa po nazivu
  - `GET /api/v1/assets/find` - napredno pretraživanje resursa po kriterijima

- **Pomoćne Funkcionalnosti**:
  - `GET /api/v1/seed` - inicijalizira bazu podataka s testnim podacima (samo za razvoj)

Kratak opis ruta:

- `/authentication/*` - autentifikacija korisnika i upravljanje sesijama
- `/tasks/*` - CRUD operacije nad zadacima, uključujući pretraživanje i filtriranje
- `/users/*` - upravljanje korisničkim računima i profilima
- `/roles/*` - definiranje i upravljanje korisničkim ulogama
- `/permissions/*` - upravljanje dozvolama i pravima pristupa
- `/assets/*` - upravljanje datotekama i resursima
- `/seed` - inicijalizacija testnih podataka (samo razvoj)

2.2.2 Dodatne informacije o API-ju

U ovome poglavlju će biti opisane dodatne informacije o API-ju, kao što su verzioniranje, statusni kodovi i dodatne funkcionalnosti.

Kako bi se pridržavalo standardnih REST API praksi, API je verzioniran koristeći `v1` prefiks. To omogućuje buduće promjene API-ja bez narušavanja postojećih klijenata. Dodatno, API vraća JSON odgovore s HTTP statusnim kodovima.

HTTP Status kodovi:

- 200 - uspješan zahtjev
- 201 - uspješno kreiranje
- 400 - neispravan zahtjev
- 401 - neautoriziran pristup
- 403 - zabranjen pristup
- 404 - resurs nije pronađen

- 500 - interna greška

Napredne funkcionalnosti:

- Paginacija rezultata
- Pretraživanje kroz `SearchTerm`
- Sortiranje kroz `SortColumn`
- Upload i preuzimanje datoteka
- JWT autentifikacija
- Kontrola pristupa kroz uloge
- Soft-delete mehanizam
- Praćenje promjena

2.2.3 Tehnička Implementacija

U ovome poglavlju će biti opisana tehnička implementacija backend aplikacije kao i značajke koje su u skladu s najboljim praksama u industriji.

Backend aplikacija je implementirana koristeći Clean Architecture pristup i Domain-Driven Design principe, što omogućava jasno odvajanje poslovne logike od infrastrukturnog koda. Za odvajanje operacija čitanja i pisanja korišten je CQRS pattern, dok Repository pattern osigurava konzistentan pristup podacima kroz cijelu aplikaciju.

Dependency Injection je implementiran koristeći ugrađeni DI container ASP.NET Core okvira. Database context je registriran kao scoped servis, osiguravajući jedan kontekst po HTTP zahtjevu. Konfiguracijske postavke su implementirane kao singleton servisi, dok su handleri za obradu zahtjeva registrirani kao transient servisi, stvarajući novu instancu za svaki zahtjev.

Sigurnosne Mjere

Autentifikacija korisnika implementirana je pomoću JWT (*eng. JSON Web Token*) mehanizma s podrškom za refresh tokene. Lozinke se pohranjuju u bazu podataka koristeći Bcrypt algoritam za hashiranje, što pruža snažnu zaštitu protiv napada grubom silom (*eng. brute force attack*). Sustav automatski deaktivira tokene nakon isteka vremena valjanosti ili prilikom brisanja korisničkog računa, čime se sprječava neovlašteni pristup.

Autorizacija je implementirana kroz višeslojni sustav kontrole pristupa. Na najvišoj razini, sustav koristi kontrolu pristupa temeljenu na ulogama (RBAC), koja se dodatno granulira kroz sustav pojedinačnih dozvola. Svaki kontroler implementira validaciju pristupa, osigurava-

jući da korisnici mogu pristupiti samo resursima za koje imaju odgovarajuće dozvole.

Za zaštitu podataka, sva komunikacija između klijenta i servera odvija se preko HTTPS protokola. Implementirana je zaštita protiv SQL injection napada kroz parametrizirane upite i ORM sloj. CORS politika je konfigurirana da dozvoljava pristup samo s poznatih i ovlaštenih domena.

Performanse i Skalabilnost

Optimizacija performansi postignuta je implementacijom nekoliko ključnih mehanizama. *Response caching* smanjuje opterećenje servera kroz privremenu pohranu često traženih podataka. *Eager i lazy loading strategije* se koriste ovisno o složenosti upita, gdje se za jednostavne upite koristi eager loading za smanjenje broja database roundtrips, dok se za složenije upite koristi lazy loading kako bi se izbjeglo nepotrebno učitavanje podataka.

Query optimizacija je implementirana kroz pažljivo dizajnirane indekse i optimizirane upite, što rezultira bržim odgovorima sustava. Za praćenje rada sustava, implementiran je sustav *error logginga* koji bilježi sve greške i iznimke, te *audit trailing* koji prati sve značajne promjene u sustavu, omogućavajući rekonstrukciju događaja i analizu korištenja sustava.

2.3 Frontend (Vježba 3)

U ovome poglavlju će biti opisana implementacija frontend aplikacije u Angularu kao i zamišljeni način korištenja aplikacije.

2.3.1 Opis frontend aplikacije

Frontend aplikacija je implementirana u Angularu i koristi Angular Material biblioteku za izgled i interakciju. Aplikacija je podijeljena na više modula koji su implementirani u skladu s Clean Architecture principima. Tehnološki stack frontend aplikacije uključuje:

- Angular 17 kao glavni razvojni okvir
- TypeScript za pisanje aplikacijske logike
- Angular Material za komponente korisničkog sučelja
- RxJS za reaktivno programiranje

Struktura Projekta

Projekt je organiziran prema Angular najboljim praksama, s jasno odvojenim modulima i komponentama:

- **Konfiguracija Projekta**:
  - `angular.json` - glavna konfiguracija Angular aplikacije
  - `tsconfig.json` - TypeScript kompilacijske opcije
  - `proxy.conf.js` - konfiguracija proxy servera za razvoj
- **Sigurnost**:
  - `aspnetcore-https.js` - konfiguracija HTTPS protokola
  - Integracija s backend autentifikacijom

Arhitektura Aplikacije

Aplikacija je strukturirana prema Clean Architecture principima, s jasno odvojenim slojevima:

- **Prezentacijski Sloj**:
  - Angular komponente za prikaz korisničkog sučelja
  - Reactive Forms za upravljanje obrascima
  - Material Design komponente za konzistentan izgled
- **Poslovna Logika**:
  - Services za komunikaciju s API-jem
  - Guards za zaštitu ruta
  - Interceptors za upravljanje HTTP zahtjevima i dodavanje autentifikacijskih tokena
- **Upravljanje Stanjem**:
  - RxJS za reaktivno programiranje
  - Observables za asinkronu komunikaciju
  - Subjects za dijeljenje stanja

Integracija s Backendom

Komunikacija s backend API-jem je implementirana kroz:

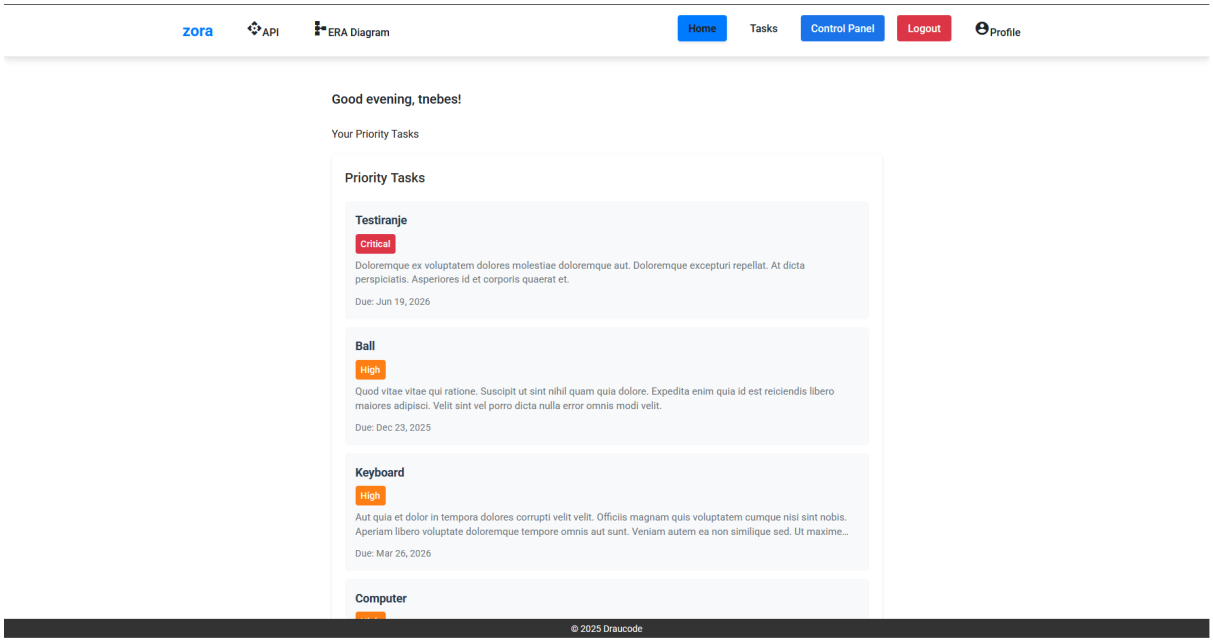
- HTTP interceptore za dodavanje autentifikacijskih tokena
- Tipizirana sučelja za API modele
- Error handling na globalnoj razini
- Retry mehanizme za neuspjele zahtjeve

Ova arhitektura omogućava skalabilnost aplikacije, jednostavno održavanje i proširivanje funkcionalnosti, uz održavanje visokih performansi i dobrog korisničkog iskustva.

2.3.2 Zamišljeni način korištenja aplikacije

U ovome poglavlju će biti opisana zamišljena upotreba frontend aplikacije. U ovome primjeru koristit će se primjer neprijavljenog korisnika koji će drugom korisniku promijeniti ovlasti kako bi taj korisnik mogao pristupiti novom radnom zadatku.

Koraci korištenja



Ovdje napisati za svaku postavljenu sliku čemu služi i kako se s aplikacijom radi.