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 (u nastavku ZORA) jest web-aplikacija namijenjena svakodnevnom olakšavanju planiranja, organizacije i upravljanja radnim zadatcima korisnicima i njihovim timovima. ZORA se može opisati kao digitalna interaktivna ploča koja omogućuje praćenje zadataka i rokova na jednom centraliziranom mjestu. Korisnici mogu jednostavno stvarati nove zadatke (primjerice „Završiti prezentaciju za sastanak“), definirati osobe odgovorne za izvršenje, postaviti rokove, pratiti tijek rada te označavati zadatke kao dovršene. Voditelji timova imaju pregled aktivnosti cijelog tima u stvarnom vremenu, dok članovi jasno uočavaju svoje obveze i vremenske rokove. Inspiracija za razvoj platforme ZORA proizlazi iz Adobeovog proizvoda Workfront, koji predstavlja profesionalni alat za upravljanje radom, s napomenom kako ZORA trenutačno pruža temeljni skup funkcionalnosti usmjeren prvenstveno na jednostavnost korištenja, bez posebnih prednosti nad navedenim alatom.

Glavne funkcionalnosti platforme ZORA uključuju stvaranje novih zadataka, dodjeljivanje istih korisnicima, detaljno praćenje njihovog statusa te generiranje preglednih izvještaja i vizualnih prikaza trenutačnog stanja zadataka. Aplikacija je osobito korisna za timove angažirane na projektima koji zahtijevaju jasno strukturiranje zadataka i upravljanje vremenskim okvirima.

Iz perspektive korištenih tehnologija, platforma ZORA implementirana je uz pomoć razvojnog okvira ASP.NET Core 9.0, s primjenom Entity Framework (EF) Core biblioteke za rad s relacijskom bazom podataka Microsoft SQL Server. Korisničko sučelje (*eng. frontend*) izrađeno je pomoću Angular razvojnog okvira uz korištenje HTML-a, CSS-a i programskog jezika TypeScript, dok je poslužiteljska logika (*eng. backend*) realizirana pomoću ASP.NET Core. Sustav već podržava autentifikaciju i autorizaciju korisnika, uključujući prijavu putem korisničkih računa te upravljanje korisničkim ulogama i pravima pristupa.

ZORA ima za cilj osigurati jednostavno i učinkovito upravljanje zadatcima, prilagodljivo različitim veličinama timova i zahtjevnosti projekata. Planirani budući razvoj platforme obuhvaća implementaciju dodatnih funkcionalnosti kao što su integracije s drugim popularnim alatima, mogućnost povezivanja zadataka s odgovarajućim projektima, naprednije opcije za analizu napretka zadataka te detaljniju optimizaciju korisničkog iskustva na mobilnim uređajima. Iz tehničke perspektive, prioritetni smjerovi razvoja obuhvaćaju poboljšanje ukupnih performansi sustava, dodatnu modernizaciju korisničkog sučelja te temeljitiju prilagodbu i kompatibilnost mobilnim platformama.

2 Vježbe

Razvoj platforme ZORA odvijao se postupno tijekom cijelog edukacijskog smjera, a bio je podijeljen u nekoliko ključnih vježbi koje su uključivale oblikovanje ERA dijagrama, razvoj *backend* dijela aplikacije te implementaciju *frontend* sučelja. U nastavku će detaljno biti opisane pojedine vježbe zajedno s postignutim rezultatima.

2.1 ERA dijagram (Vježba 1)

Cilj prve vježbe bio je osmisliti i izraditi *Structured Query Language* (SQL) skriptu za kreiranje baze podataka aplikacije. Skripta je testirana korištenjem alata Microsoft SQL Server Management Studio (SSMS) gdje je izvršena vizualna provjera strukture baze te ručni unos testnih podataka.

Dizajn baze podataka zasnivao se na unaprijed osmišljenim temeljnim zahtjevima, pri čemu je bila izražena potreba za modularnošću sustava kroz implementaciju relacija tipa više-na-više. Djelomična inspiracija pri oblikovanju baze dolazi iz koncepata Big Data, ali isključivo na osnovnoj razini. Također, uvedena je apstraktna tablica `WorkItem` kako bi se pojednostavilo pretraživanje zajedničkih atributa različitih radnih jedinica poput zadataka, projekata i programa. Važno je napomenuti da se implementacija entiteta projekta i programa tek planira u budućim verzijama sustava.

Baza podataka kreirana je u dvije faze. Prva faza uključivala je inicijalno postavljanje same baze, uključujući konfiguraciju baze i korisničkih prava pristupa:

```
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];
```

U drugoj fazi izrađene su tablice i definirane njihove međusobne relacije. Glavne tablice uključuju korisnike (`zora_users`), uloge (`zora_roles`), radne stavke (`zora_work_items`) te specijalizirane tablice poput projekata i zadataka.

Primjer stvaranja tablice za pohranu korisničkih podataka:

```
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 '%_@%.%')
```

```
);
```

Za osiguranje integriteta podataka korišten je *soft-delete* pristup kroz `deleted` zastavicu, a definirane su i odgovarajuće relacije preko vanjskih ključeva:

```
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)
);
```

Kao dio inicijalnog postavljanja baze, kreirani su administratorski korisnik te osnovne uloge u sustavu. Baza je dodatno optimizirana indeksima, posebno za aktivne (neobrisane) podatke korištenjem uvjeta `WHERE deleted = 0` radi ubrzavanja učestalih upita.

Tijekom razvoja naišlo se na specifične izazove vezane uz nove značajke Microsoft SQL Servera, posebice vezano uz korištenje postavke `SET QUOTED_IDENTIFIER ON`. Ova postavka određuje kako se interpretiraju navodnici oko identifikatora objekata u SQL skriptama. Problem se očitovao kroz neočekivana ponašanja u izvršavanju skripata, osobito pri kreiranju i modifikaciji tablica i indeksa. Kako bi se riješili navedeni problemi, bilo je nužno pažljivo prilagoditi SQL skripte, osiguravajući konzistentno postavljanje `SET QUOTED_IDENTIFIER ON` prije izvršavanja naredbi koje zahtijevaju strogo definirane identifikatore objekata. Time se osigurala kompatibilnost i pravilno izvršavanje svih skripata unutar nove verzije Microsoft SQL Servera.

ERA dijagram izrađen je korištenjem alata Eraser.io (`https://www.eraser.io/ai`) te jasno prikazuje strukturu baze podataka, uključujući entitete, njihove atribute i definirane relacije. Standard imenovanja objekata u bazi slijedio je najbolje prakse iz područja razvoja baza podataka, čime je osigurana jasnoća i održivost koda i strukture u budućem radu:


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

2.2 Backend (Vježba 2)

Backend aplikacije implementiran je kao REST API koristeći razvojni okvir ASP.NET Core 9.0, a strukturiran je u nekoliko funkcionalnih područja radi bolje preglednosti i održivosti sustava.

2.2.1 Opis API-ja

API je dizajniran prema REST načelima, jasno definirajući odgovarajuće HTTP endpointe za specifične funkcionalnosti aplikacije.

- **Autentifikacija i autorizacija**
  - `POST /api/v1/authentication/token` - generira JWT token za prijavljenog korisnika.
  - `GET /api/v1/authentication/check` - provjerava valjanost JWT tokena.
  - `GET /api/v1/authentication/current-user` - vraća podatke trenutnog korisnika.
  - `POST /api/v1/authorisation/is-authorised` - provjerava korisničke dozvole.
  - `GET /api/v1/authorisation/is-admin` - provjerava administratorske privilegije korisnika.
- **Upravljanje zadacima**
  - `GET /api/v1/tasks` - dohvaća listu zadataka s podrškom za filtriranje i sortiranje.
  - `POST /api/v1/tasks` - kreira novi zadatak.
  - `GET /api/v1/tasks/id` - dohvaća detalje pojedinog zadatka.
  - `PUT /api/v1/tasks/id` - ažurira postojeći zadatak.
  - `DELETE /api/v1/tasks/id` - deaktivira zadatak.
  - `POST /api/v1/tasks/id/assign` - dodjeljuje zadatak korisniku.
  - `POST /api/v1/tasks/id/complete` - označava zadatak kao završen.
  - `GET /api/v1/tasks/id/assets` - dohvaća resurse povezane sa zadatkom.
  - `GET /api/v1/tasks/priority/userId` - dohvaća prioritetne zadatke za korisnika.
  - `GET /api/v1/tasks/find` - pretražuje zadatke prema zadanim kriterijima.
  - `GET /api/v1/tasks/search` - napredno pretraživanje zadataka s više parametara.
- **Upravljanje korisnicima**

  - `GET /api/v1/users` - dohvaća listu korisnika s podrškom za filtriranje i sortiranje.
  - `POST /api/v1/users` - kreira novog korisnika.
  - `GET /api/v1/users/id` - dohvaća detalje pojedinog korisnika.
  - `PUT /api/v1/users/id` - ažurira korisničke podatke.
  - `DELETE /api/v1/users/id` - deaktivira korisnika.
  - `GET /api/v1/users/find` - pretražuje korisnike prema zadanim kriterijima.
  - `GET /api/v1/users/search` - napredno pretraživanje korisnika s više parametara.

- **Uloge i dozvole**
  - `GET /api/v1/roles` - dohvaća listu uloga s podrškom za filtriranje i sortiranje.
  - `POST /api/v1/roles` - kreira novu ulogu.
  - `PUT /api/v1/roles/id` - ažurira postojeću ulogu.
  - `DELETE /api/v1/roles/id` - deaktivira ulogu.
  - `GET /api/v1/roles/find` - pretražuje uloge prema zadanim kriterijima.
  - `GET /api/v1/roles/search` - napredno pretraživanje uloga s više parametara.
  - `GET /api/v1/permissions` - dohvaća listu dozvola s podrškom za filtriranje i sortiranje.
  - `POST /api/v1/permissions` - kreira novu dozvolu.
  - `PUT /api/v1/permissions/id` - ažurira postojeću dozvolu.
  - `DELETE /api/v1/permissions/id` - deaktivira dozvolu.
  - `GET /api/v1/permissions/find` - pretražuje dozvole prema zadanim kriterijima.
  - `GET /api/v1/permissions/search` - napredno pretraživanje dozvola s više parametara.

- **Upravljanje resursima (assets)**
  - `GET /api/v1/assets` - dohvaća listu resursa s podrškom za filtriranje i sortiranje.
  - `POST /api/v1/assets` - omogućuje učitavanje novog resursa.
  - `GET /api/v1/assets/id` - dohvaća detalje pojedinog resursa.
  - `PUT /api/v1/assets/id` - ažurira postojeći resurs.
  - `DELETE /api/v1/assets/id` - deaktivira resurs.
  - `GET /api/v1/assets/id/download` - omogućuje preuzimanje resursa.
  - `GET /api/v1/assets/id/preview` - omogućuje pregled resursa.

  - `GET /api/v1/assets/find` - pretražuje resurse prema zadanim kriterijima.
  - `GET /api/v1/assets/search` - napredno pretraživanje resursa s više parametara.

- **Razvojni alati**
  - `GET /api/v1/seed` - inicijalizira bazu podataka testnim podacima.

2.2.2 Verzioniranje i HTTP statusni kodovi

API je verzioniran korištenjem prefiksa `v1`, što omogućava buduće promjene bez prekidanja rada postojećih klijenata. API prati REST standarde korištenjem semantičkih HTTP statusnih kodova (200, 201, 400, 401, 403, 404 i 500) za prikazivanje stanja zahtjeva.

Implementirane su i napredne funkcionalnosti poput paginacije, pretraživanja (`SearchTerm`), sortiranja (`SortColumn`), te podrška za upload i preuzimanje datoteka, JWT autentifikaciju, sustav dozvola i soft-delete pristup za očuvanje povijesti podataka.

2.2.3 Tehnička implementacija

Backend je realiziran korištenjem arhitekture Clean Architecture i Domain-Driven Design principa koji omogućuju jasno razdvajanje poslovne logike od infrastrukture. CQRS pattern odvaja čitanje i pisanje, dok Repository pattern standardizira pristup podacima.

Dependency Injection osiguran je ASP.NET Core DI containerom. Database context konfiguriran je kao scoped servis, konfiguracijske postavke su singleton, a handleri transient servisi.

Stack uključuje Entity Framework Core 9.0 za pristup SQL Server bazi podataka, AutoMapper za mapiranje, JWT za autentifikaciju, Serilog za logiranje i Swashbuckle za dokumentaciju API-ja.

Sigurnost je implementirana putem JWT tokena s refresh token podrškom, BCrypt hashing algoritmom za pohranu lozinki, HTTPS komunikacijom, zaštitom od SQL injection napada i konfiguriranjem CORS politike.

Performanse su optimizirane response cachingom, korištenjem eager i lazy loading strategija te optimizacijom upita i indeksa. Implementiran je i sustav logiranja grešaka i audit trail za praćenje promjena u sustavu.

2.3 Frontend (Vježba 3)

U ovom poglavlju opisana je implementacija frontend aplikacije u razvojnom okviru Angular 19, kao i predviđeni način korištenja aplikacije.

2.3.1 Opis frontend aplikacije

Frontend aplikacija razvijena je koristeći Angular 19 s Angular Material bibliotekom koja osigurava vizualno konzistentan i responzivan dizajn. Za implementaciju aplikacijske logike korišten je programski jezik TypeScript, uz primjenu RxJS biblioteke za reaktivno programiranje. Upravljanje obrascima izvedeno je putem Angular Reactive Forms modula, navigacija kroz Angular Router, a HTTP komunikacija s backendom realizirana je pomoću Angular HttpClient modula.

2.3.1.1 Struktura projekta

Projekt prati najbolje prakse Angular zajednice, s jasno definiranim modulima radi bolje organizacije:

- `app.module.ts` – glavni modul aplikacije.
- `control-panel.module.ts` – administratorske funkcionalnosti.
- `tasks.module.ts` – upravljanje zadacima.
- `login.module.ts` – autentifikacija korisnika.
- `shared.module.ts` – zajedničke komponente i servisi.

Ključne komponente uključuju osnovnu `app.component`, navigacijske komponente `nav-menu.component` i `sidebar.component`, te komponente specifične za entitete poput korisnika, uloga, dozvola, zadataka i korisničkog profila.

Sigurnosne značajke aplikacije uključuju HTTPS konfiguraciju (`aspnetcore-https.js`), guards za zaštitu ruta (`auth.guard.ts`, `admin.guard.ts`) te HTTP interceptore za automatsko dodavanje JWT tokena (`authentication.interceptor.ts`).

2.3.1.2 Arhitektura aplikacije

Primijenjena je Clean Architecture struktura s jasnim odvojenim slojevima:

- **Prezentacijski sloj** – komponente, Reactive Forms, Angular Material komponente.
- **Poslovna logika** – Angular servisi, guards, interceptori, validatori.
- **Upravljanje stanjem** – RxJS Observables, Subjects, BehaviorSubjects.

2.3.1.3 Integracija s backendom

Komunikacija s backend API-jem ostvarena je HTTP interceptorima za rukovanje JWT autentifikacijom, tipiziranim sučeljima za sigurnost tipova, globalnim rukovanjem greškama, te retry mehanizmima za HTTP zahtjeve.

2.3.1.4 Korisničko iskustvo

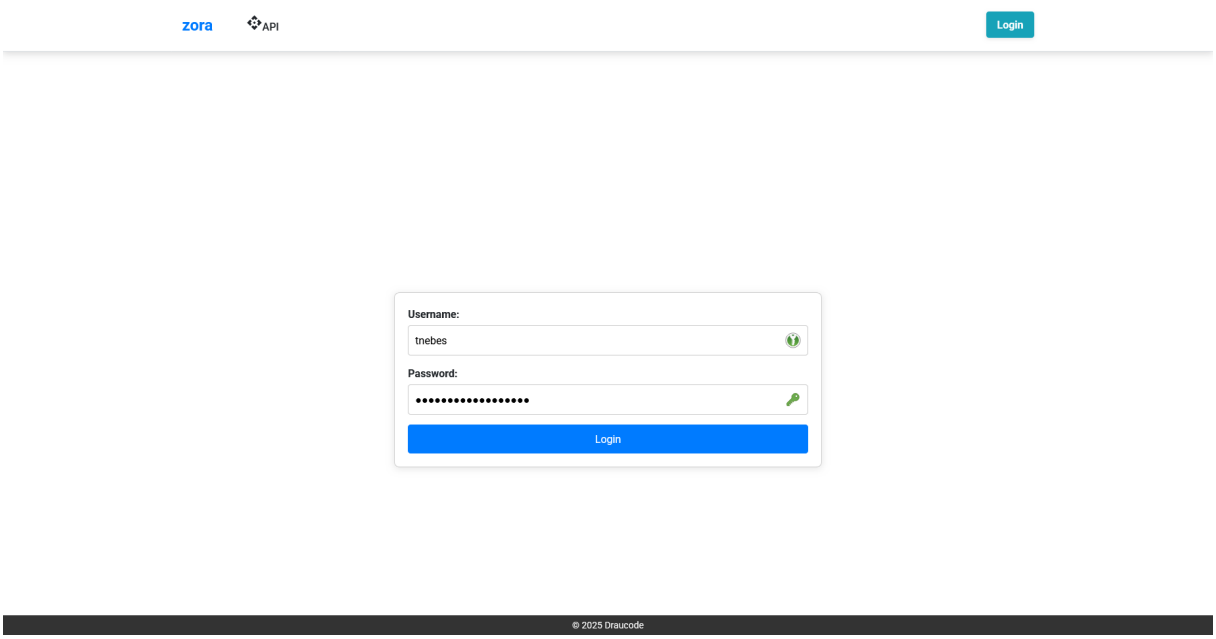
Korisničko sučelje dizajnirano je s naglaskom na responzivnost, intuitivnu navigaciju i konzistentnost kroz Angular Material. Notifikacije i snackbar poruke pružaju korisnicima pravovremene povratne informacije, dok se funkcionalnosti aplikacije prilagođavaju korisničkim ulogama i pravima pristupa.

2.3.2 Zamišljeni način korištenja aplikacije

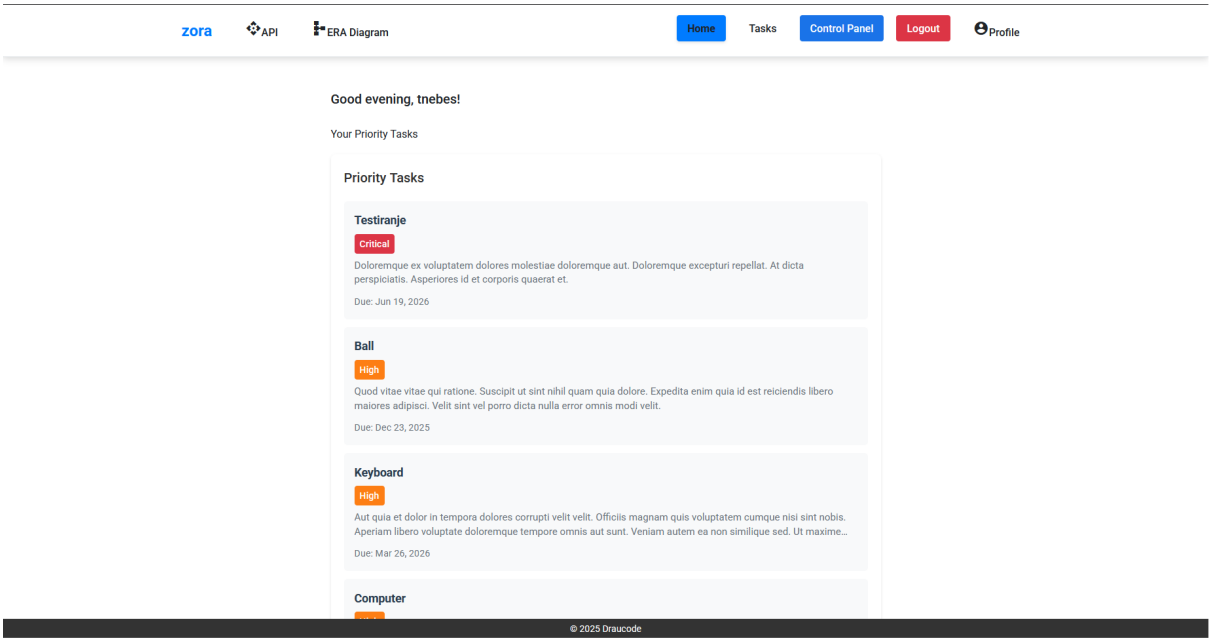
Sljedeći primjer opisuje tipičan scenarij korištenja frontend aplikacije kroz sljedeće korake:

- **Prijava korisnika** – autentifikacija i pristup sustavu.
- **Kreiranje novog zadatka** – unos i postavljanje detalja zadatka.
- **Dodjela zadatka (ovlasti) drugom korisniku** – dodjela zadataka određenom korisniku.
- **Prikaz zadatka** – pregled i praćenje statusa zadataka.
- **Dodavanje resursa** – povezivanje dodatnih datoteka i resursa sa zadatkom.

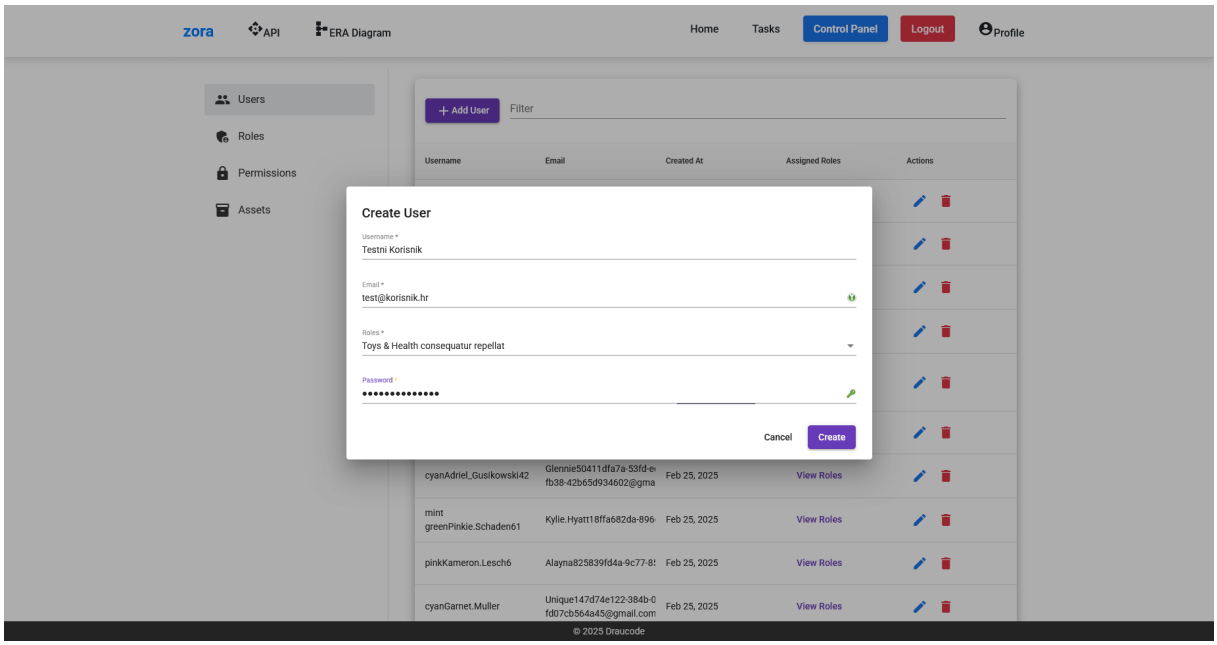
Slijedi vizualni prikaz korisničkog sučelja aplikacije pri kreiranju novog korisnik, kreiranju novog zadatka, dodavanju resursa, dodjeljivanje ovlasti drugom korisniku za pregled zadatka.



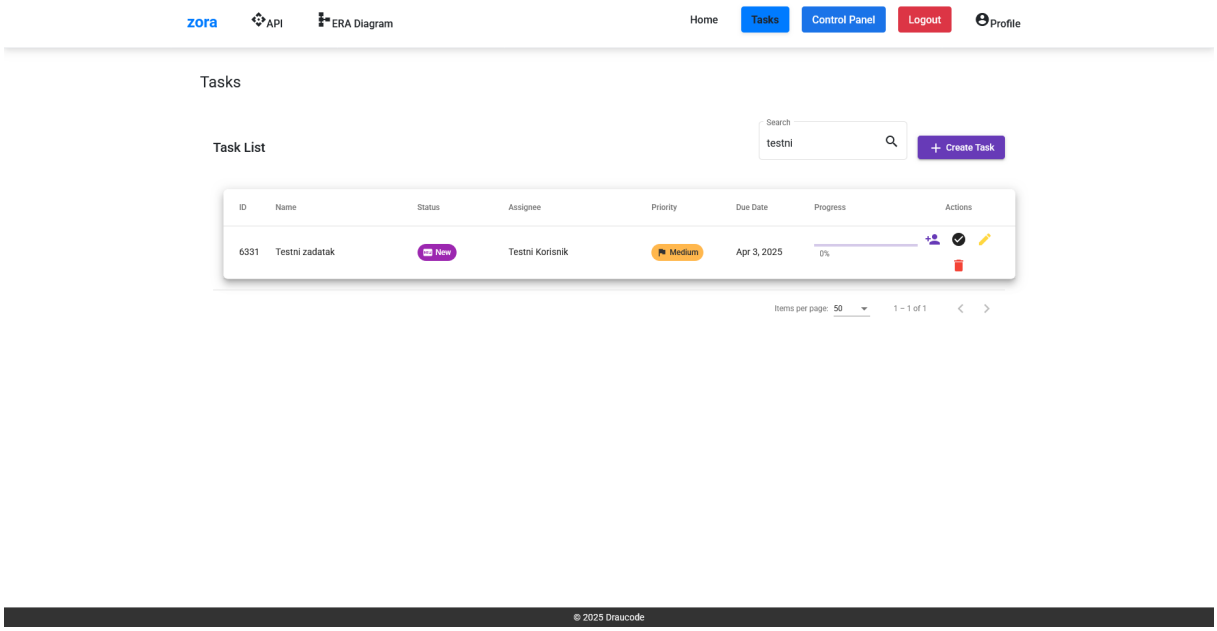
Slika 1: Prijava korisnika



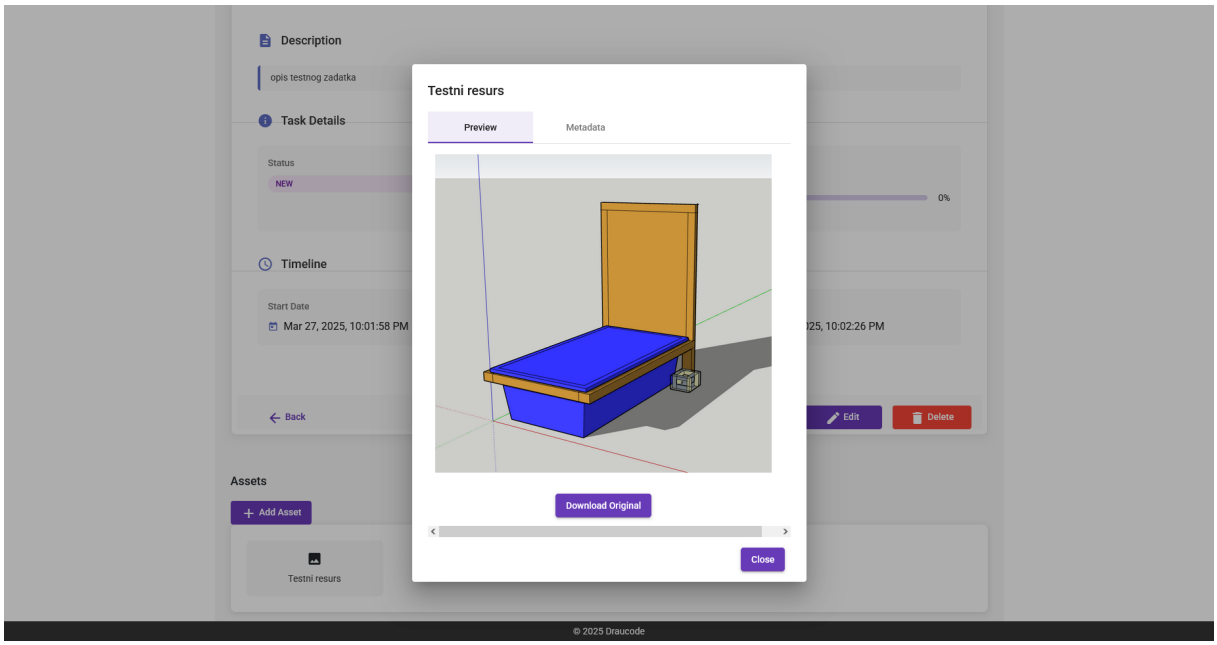
Slika 2: Prikaz korisničkog sučelja ZORA aplikacije



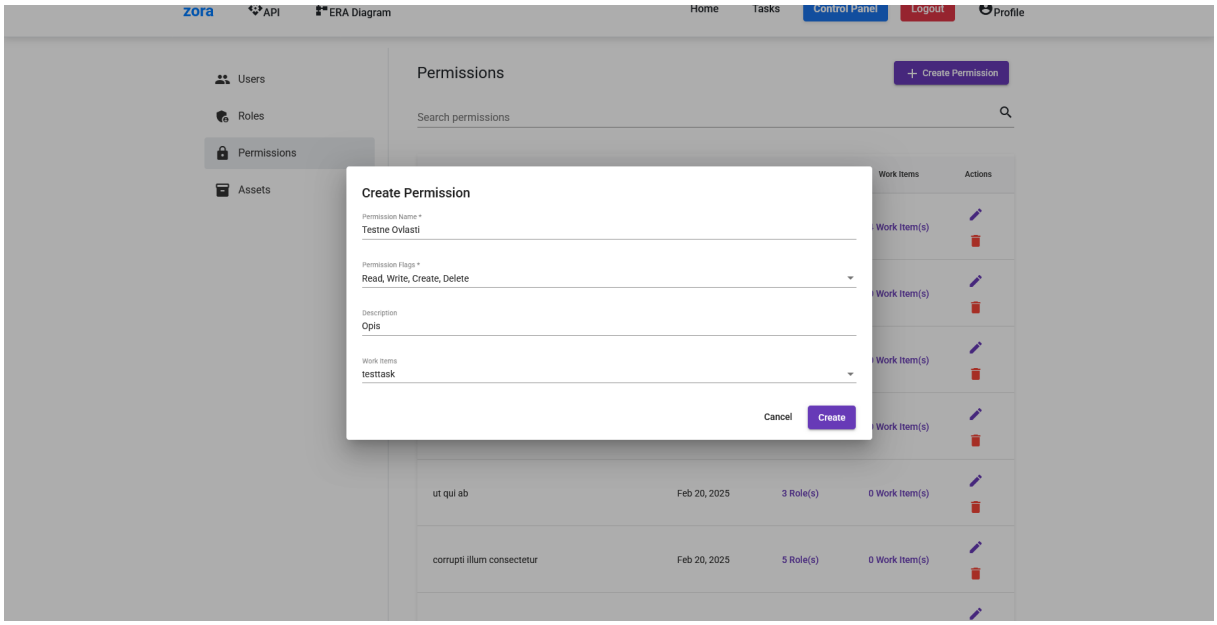
Slika 3: Kreiranje novog korisnika



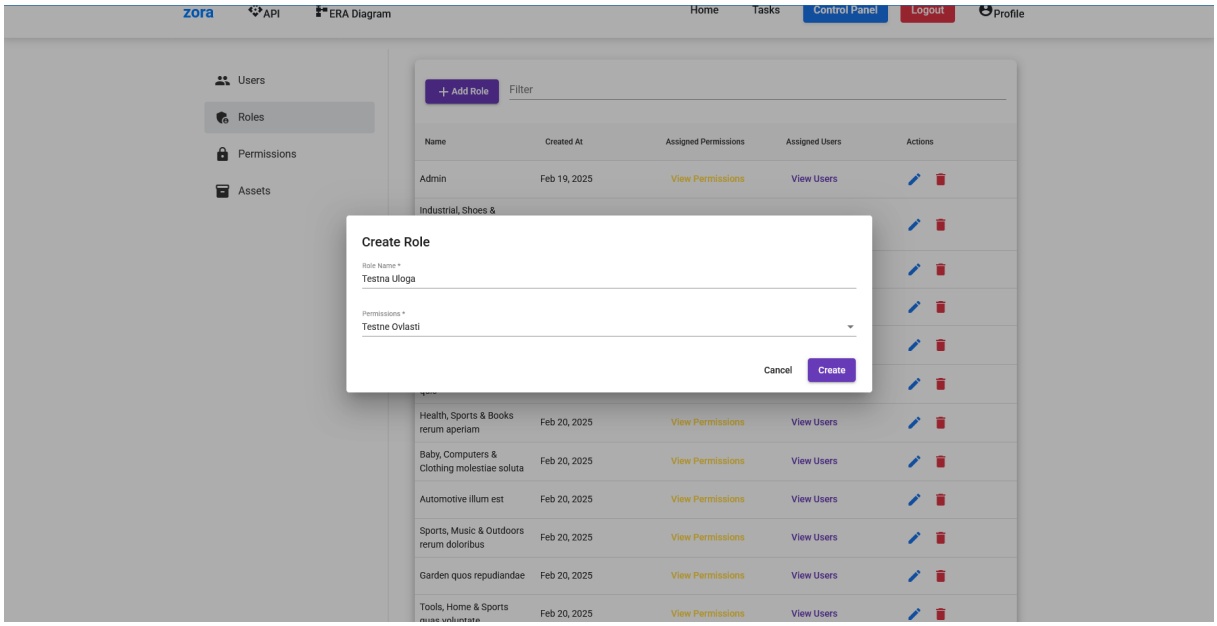
Slika 4: Kreirani novi zadatak kojemu je dodijeljen ”Testni Korisnik”



Slika 5: Kreirani novi zadatak kojemu je dodan resurs



Slika 6: Kreiranje ovlasti nad zadatkom



Slika 7: Kreiranje nove uloge s prethodno kreiranim ovlastima

3 Razvojna i Produkcijska Okruženja

ZORA aplikacija dizajnirana je za učinkovito korištenje u razvojnim i produkcijskim okruženjima, omogućavajući sigurnost i optimalne performanse u oba scenarija. Implementirana su dva ključna okruženja: *eng. Development* razvojno okruženje i *eng. Production* produkcijsko okruženje.

3.1 Konfiguracija Okruženja

Aplikacija koristi konfiguracijske datoteke `appsettings.json` i `appsettings.Environment.json`, gdje `Environment` označava naziv okruženja (npr. Development ili Production).

3.1.1 Razvojno Okruženje

Razvojno okruženje (`Development`) podržava:

- Detaljno logiranje (Debug, Information, Warning, Error).
- Prikaz detaljnih grešaka s lokacijama i stack trace-ovima.
- Swagger dokumentaciju dostupnu na `/swagger`.
- Inicijalizaciju testnih podataka putem `/seed` endpointa.
- Automatsko osvježavanje aplikacije (Hot Reload).

3.1.2 Produkcijsko Okruženje

Produkcijsko okruženje (`Production`) optimizirano je za sigurnost i performanse:

- Reducirano logiranje (samo Warning i Error).
- Generičke greške bez detalja implementacije.
- Komprimirani i minimizirani resursi.
- Keširanje odgovora (response caching).

3.2 Upravljanje Tajnim Podacima

ZORA implementira sljedeće metode za upravljanje tajnim podacima:

3.2.0.1 Environment Variables

Osjetljivi podaci (connection strings, API ključevi, lozinke) dostavljaju se putem varijabli okruženja na produkcijskom serveru.

3.2.0.2 Secret Manager

Za lokalni razvoj koristi se ASP.NET Core Secret Manager, čime se tajni podaci drže izvan izvornog koda i sustava verzioniranja.

3.3 Deployment i Infrastruktura

3.3.0.1 Continuous Integration/Continuous Deployment

CI/CD pipeline je implementiran kroz Jenkins, dostupan na portu 8080. Definiran u `Jenkinsfile` datoteci, pipeline automatizira:

- **Checkout**: dohvat izvornog koda.
- **Restore**: instalaciju NuGet i npm paketa.
- **Build**: kompilaciju aplikacija.
- **Test**: pokretanje automatiziranih testova.
- **Publish**: pripremu za deployment.
- **Deploy**: kopiranje aplikacije i restart servisa.

Webhook integracija osigurava automatski deployment nakon svakog push-a u main granu.

3.3.0.2 Infrastruktura

Produkcijsko okruženje temelji se na Ubuntu VPS-u:

- Backend aplikacija pokrenuta je na Kestrel serveru kao `systemd` servis.
- Microsoft SQL Server baza nalazi se na istom serveru.
- Frontend aplikacija posluživana kao statički sadržaj.
- Nginx kao reverse proxy upravlja SSL/TLS enkripcijom kroz Let's Encrypt.

Domenska struktura uključuje glavnu aplikaciju na `draucode.com` i frontend na poddomeni `zora.draucode.com`. Nginx konfiguracija učinkovito upravlja SSL terminacijom, kompresijom i cachiranjem, usmjeravajući API zahtjeve prema Kestrelu.

Alati za održavanje:

- `fail2ban` - zaštita od brute-force napada.

- `ufw` - upravljanje firewallom.
- `logrotate` - upravljanje log datotekama.

Ova infrastruktura omogućuje robusnost, skalabilnost, visoku sigurnost te jednostavno održavanje i proširivanje sustava.

4 Tehnička implementacija specifičnih funkcionalnosti

ZORA aplikacija koristi nekoliko naprednih tehničkih pristupa za učinkovito rješavanje specifičnih razvojnih izazova. Svaka od navedenih implementacija rezultat je pažljive analize i primjene industrijski priznatih najboljih praksi.

4.1 Napredni sustav upravljanja dozvolama

4.1.0.1 Bit-masked dozvole

Sustav dozvola implementiran je korištenjem enum-a s `[Flags]` atributom, omogućavajući kombinaciju dozvola putem bitwise operacija:

```
[Flags]
public enum PermissionFlag
{
None = 0,      // 00000
Read = 1,      // 00001
Write = 2,     // 00010
Create = 4,    // 00100
Delete = 8,    // 01000
Admin = 16     // 10000
}
```

Ovaj pristup omogućava kompaktnu pohranu dozvola i brzu provjeru ovlasti putem bitwise operacija.

4.1.0.2 Hijerarhijski pristup

Dozvole slijede hijerarhiju gdje viša dozvola uključuje niže. Korisnici s administratorskim pravima automatski imaju sve dozvole.

4.1.0.3 Caching autorizacijskih provjera

Za optimizaciju performansi, rezultati autorizacijskih provjera privremeno se pohranjuju u cache, smanjujući učestale upite prema bazi podataka.

4.2 Upravljanje resursima i datotekama

4.2.0.1 Sigurno učitavanje datoteka

Ključne sigurnosne mjere uključuju generiranje jedinstvenih naziva datoteka, validaciju veličine i vrste datoteka te izolaciju stvarnih putanja datoteka od javnog pristupa.

4.2.0.2 Pretpregled datoteka

Automatsko generiranje pretpregleda implementirano je za slikovne datoteke, s asinkronim procesom radi minimalnog utjecaja na performanse sustava.

4.3 Strategija testiranja

4.3.0.1 Unit testovi

Unit testovi koriste se za izolirano testiranje pojedinačnih komponenti uz mocking vanjskih ovisnosti (Moq).

4.3.0.2 Integracijski testovi

Integracijski testovi izvode se u izoliranom in-memory testnom okruženju koristeći posebno prilagođenu konfiguraciju aplikacije. Ovo omogućuje brzo i učinkovito testiranje interakcije između različitih dijelova sustava bez potrebe za stvarnom bazom podataka ili drugim vanjskim servisima.

5 Dodatna objašnjenja i razmatranja

U ovom poglavlju razmotreni su dodatni detalji razvoja aplikacije koji nisu detaljno obrađeni u prethodnim poglavljima.

5.1 Izazovi tijekom razvoja i implementacije

Razvoj aplikacije ZORA predstavljao je značajan izazov, posebno s obzirom na opseg projekta koji je implementirao jedan programer, što je rezultiralo određenim kompromisima.

5.1.0.1 Standardizacija i konzistentnost koda

Iako su početni ciljevi uključivali strogu standardizaciju koda korištenjem sučelja poput `IBaseService` za servisni sloj i `ITaskRepository` za repozitorijski sloj, tijekom razvoja javile su se poteškoće u održavanju konzistentnosti. Specifične poslovne logike različitih servisa često su zahtijevale dodatne funkcionalnosti izvan standardnog skupa CRUD operacija, dovodeći do nekonzistentnosti u implementaciji.

Ove su razlike djelomično rezultat ograničenog vremena, ali i promjena u razumijevanju potrebnih funkcionalnosti tijekom razvoja.

5.1.0.2 Kvaliteta korisničkog sučelja

Iako frontend aplikacija uspješno ispunjava funkcionalne zahtjeve, postoje određena ograničenja vezana uz intuitivnost korisničkog sučelja. Posebno složeni moduli poput upravljanja dozvolama i ulogama zahtijevaju bolju prezentaciju kako bi korištenje aplikacije bilo jednostavnije za krajnjeg korisnika.

Komentari u kodu jasno ukazuju na specifične izazove vezane uz korisničko iskustvo, poput pretrage uloga i intuitivnog izbora dozvola, koji zahtijevaju dodatnu pažnju i stručnost UI/UX dizajnera za daljnje poboljšanje aplikacije.

5.2 Prednosti postojećeg pristupa

Unatoč navedenim izazovima, odabrani razvojni pristupi imaju značajne prednosti:

5.2.1 Olakšano proširenje funkcionalnosti

Standardizirana sučelja omogućavaju jednostavno dodavanje novih entiteta i operacija, što pojednostavljuje daljnji razvoj aplikacije.

5.2.2 Konzistentno rukovanje greškama

Biblioteka `FluentResults` omogućuje jedinstveno rukovanje greškama kroz aplikaciju, što doprinosi stabilnosti i jasnoći komunikacije s klijentima.

5.2.3 Jednostavno testiranje

Implementacija temeljena na sučeljima olakšava pisanje kvalitetnih unit testova kroz jednostavno mockanje ovisnosti, čime se povećava pouzdanost sustava.

5.2.4 Skalabilnost rješenja

Postojeća arhitektura pruža čvrstu osnovu za skaliranje i proširenje aplikacije bez značajnih izmjena postojećeg koda, osiguravajući dugoročnu održivost sustava.

ZORA aplikacija, unatoč izazovima, predstavlja robustnu i održivu platformu za budući razvoj.

6 Zaključak

ZORA aplikacija predstavlja moderan i skalabilan sustav za upravljanje zadacima koji je razvijen kroz nekoliko ključnih faza. Od početnog razvoja ERA dijagrama i strukture baze podataka, preko implementacije backend API-ja s naprednim sustavom dozvola, do razvoja frontend aplikacije s fokusom na korisničko iskustvo, svaka faza je donijela svoje specifične izazove i rješenja.

Ključne prednosti implementirane arhitekture uključuju:

- **Modularnost i održivost**: Korištenje Clean Architecture principa i jasno definiranih sučelja omogućava jednostavno održavanje i proširenje funkcionalnosti.
- **Sigurnost**: Implementacija naprednog sustava dozvola s bit-maskiranjem i hijerarhijskim pristupom osigurava preciznu kontrolu pristupa.
- **Skalabilnost**: Optimizirana baza podataka, caching mehanizmi i učinkovito upravljanje resursima omogućavaju rast aplikacije.
- **Korisničko iskustvo**: Intuitivno sučelje i responzivni dizajn pružaju dobro korisničko iskustvo na različitim platformama.

Unatoč uspješnoj implementaciji osnovnih funkcionalnosti, postoji prostor za daljnje unapređenje. Prioriteti za budući razvoj uključuju:

- **Proširenje funkcionalnosti**: Implementacija hijerarhijske strukture programa i projekata, naprednih alata za praćenje napretka i integracija s drugim sustavima.
- **Optimizacija performansi**: Daljnje unapređenje caching strategija i optimizacija upita bazi podataka.
- **Poboljšanje korisničkog iskustva**: Razvoj mobilne aplikacije i unapređenje postojećeg web sučelja kroz uključivanje UI/UX stručnjaka.
- **Testiranje i kvaliteta**: Proširenje pokrivenosti testovima i implementacija automatiziranog testiranja korisničkog sučelja.

ZORA aplikacija demonstrira kako se moderni razvojni pristupi i tehnologije mogu primijeniti za stvaranje praktičnog i učinkovitog rješenja za upravljanje zadacima. Kroz pažljivo planiranje arhitekture i implementaciju najboljih praksi, stvorena je osnova koja omogućava kontinuirani razvoj i prilagodbu promjenjivim zahtjevima korisnika.