Razvoj integriranog sustava za vođenje projekata i upravljanje zadacima

Viljušić, Luka

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski studij računarstva

Završni rad

Razvoj integriranog sustava za vođenje projekta i upravljanje zadacima

Rijeka, rujan 2024.

Luka Viljušić
0069093456

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski studij računarstva

Završni rad

Razvoj integriranog sustava za vođenje projekta i upravljanje zadacima

Mentor: izv.prof.dr.sc. Goran Mauša

Komentor: Zvonimir Matić, mag. ing. comp.

Rijeka, rujan 2024.

Luka Viljušić
0069093456

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 15.03.2024.

Zavod: Zavod za računarstvo

Predmet: Programsko inženjerstvo

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Luka Viljušić (0069093456)**

Studij: Sveučilišni prijediplomski studij računarstva (1035)

Zadatak: **Razvoj integriranog sustava za vođenje projekata i upravljanje zadacima / Developmet of integrated project and issue management system**

Opis zadatka:

Proučiti tehnologije ASP.NET Core, Entity Framework Core, React.js i TypeScript za razvoj web sustava koji osigurava učinkovito vođenje projekata i podržava suradnju između timova. Predložiti model sustava koji registriranim korisnicima omogućuje izradu i upravljanje virtualnim prostorima za timski rad te pruža mogućnost povezivanja sa sustavom za praćenje zadataka, problema i grešaka poput GitHub Issues i YouTrack. Provesti implementaciju i testiranje predloženog modela te komentirati prednosti i nedostatke upotrebe zadanih tehnologija.

Rad mora biti napisan prema Uputama za pisanja diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 20.03.2024.

Predsjednik povjerenstva za
završni ispit:
prof. dr. sc. Miroslav Joler

Mentor:
izv. prof. Goran Mauša

Komentor:
Zvonimir Matić

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2024.

_____

Luka Viljušić

Zahvala

Zahvaljujem mentoru Goranu Mauši i komentoru Zvonimiru Matiću na korisnim raspravama i savjetima. Zahvaljujem svojoj curi i obitelji na podršci tijekom studiranja.

Sadržaj

1 Uvod

U današnjem poslovnom svijetu, gdje složenost projekata i potrebe za timskom suradnjom neprestano rastu, ključna stavka za ostvarivanje uspjeha je dobra organizacija zadataka i učinkovito vođenje projekta. Učinkovito upravljanje projektima ne samo da povećava produktivnost, već omogućuje i bolje praćenje napretka te poboljšava komunikaciju unutar timova. U tu svrhu razvijeni su brojni alati i sustavi koji pomažu organizacijama da usmjere svoje resurse i vrijeme prema postizanju ciljeva, a pritom osiguravaju da svi članovi tima budu usklađeni i informirani.

Postojeći alati za upravljanje projektima, poput Asane, Trella, Jira, i Microsoft Project, široko su prihvaćeni zbog svoje sposobnosti da olakšaju planiranje, praćenje i koordinaciju zadataka. Asana i Trello ističu se svojim vizualnim pristupom i jednostavnošću korištenja, dok Jira pruža duboku integraciju s razvojnim procesima i kompleksnijim funkcijama za praćenje pogrešaka i upravljanje projektima. Microsoft Project nudi robusne mogućnosti za detaljno planiranje i praćenje napretka, ali često zahtijeva dodatnu obuku za učinkovito korištenje. Ovi alati su ključni za optimizaciju radnih procesa i omogućuju organizacijama da učinkovito upravljaju svojim projektima, no često su skupi, složeni ili ne nude sve potrebne funkcionalnosti za specifične potrebe korisnika.

Ovaj završni rad bavi se izradom web sustava za vođenje projekta i upravljanje zadacima, te je cilj omogućiti korisnicima jednostavno stvaranje i praćenje zadataka. U suradnji s tvrtkom InCubis definiran je zadatak izrade web sustava u kojem se korisnik može prijaviti, voditi projekte, stvarati i upravljati zadacima te dijeliti projekte s drugim korisnicima.

Kroz ovaj rad upoznaju se tehnologije i alati potrebni za izradu web sustava, uključujući programski kod, metodologije razvoja te postupke implementacije i tes-

tiranja. Poseban naglasak stavlja se na ASP.NET Core, razvojni okvir za izradu web aplikacija u C#, te Entity Framework Core, biblioteku za komunikaciju s bazom podataka koja omogućuje jednostavnije mapiranje objekata na relacijske baze podataka. Nadalje, obrađuje se React.js, biblioteka za izradu dinamičkih i responzivnih korisničkih sučelja, i TypeScript, nadogradnja na JavaScript jezik koja osigurava provjeru tipa, te pomaže u otkrivanju grešaka tijekom razvoja i olakšava održavanje koda. Također, rad pruža analizu prednosti i nedostataka upotrebe korištenih tehnologija.

2 Specifikacija zahtjeva

Specifikacija zahtjeva jedan je od ključnih dokumenata u procesu razvoja softverskog rješenja, koja definira sve potrebne funkcionalnosti, sigurnosne zahtjeve, vanjska sučelja, te tehnologije i alate koji će biti korišteni. Ovo poglavlje pruža detaljan pregled i opis zahtjeva za web aplikaciju "Work Manager System".

2.1 Funkcionalnost i sigurnost

Web aplikacija "Work Management System" omogućava korisnicima kreiranje različitih zadataka, kojima se dodjeljuje tip zadatka (npr. pogreška, dokumentacija). Pored tipa zadatka, svakom zadatku se može dodijeliti stanje, omogućavajući praćenje trenutnog statusa (npr. započeto, u tijeku, završeno) i određivanje krajnjeg roka za izvršenje zadatka. Svaki zadatak je smješten u određeni prostor (eng.space), koji predstavlja jedan projekt. U okviru određenog prostor može se dodati više korisnika putem njihovih e-mail adresa. Također, svakom zadatku se može dodijeliti odgovorna osoba za njegovo izvršenje.

Osim toga, aplikacija pruža mogućnost detaljnog praćenja napretka projekata kroz integrirani sustav koji uključuje funkcionalnosti za upravljanje zadacima i praćenje rokova. Korisnici trebaju moći postaviti i dodijeliti zadatke unutar prostora, označiti i pratiti njihovo stanje, omogućavajući timovima učinkovitu suradnju. Korisnici trebaju moći intuitivno upravljati svojim zadacima, dodjeljivati odgovornosti i pratiti rokove bez potrebe za dodatnim objašnjenjima ili čitanjem priručnika. Sučelje aplikacije trebalo bi biti dizajnirano na način koji omogućava korisnicima da odmah razumiju kako koristiti sve funkcionalnosti kroz jasno označene komponente

i vizualne upute.

Dodatno, aplikacija je dizajnirana s naglaskom na sigurnost, uključujući zaštitu od uobičajenih web ranjivosti poput XSS i CSRF napada, te koristi moderne tehnike za autentifikaciju i autorizaciju korisnika, odnosno JSON Web Tokens (JWT). Korištenje JWT omogućava sigurno upravljanje sesijama korisnika, bez potrebe za pohranjivanjem stanja na serveru, poboljšavajući skalabilnost i sigurnost sustava.

2.2 Vanjska sučelja

Sustav za vođenje projekata i upravljanje zadacima omogućuje vanjsku interakciju s ljudima te s vanjskim hardverom poput miša i tipkovnice.

Aplikacija je dizajnirana prvenstveno za stolna i prijenosna računala, no njena responzivnost omogućava upotrebu i na mobilnim uređajima. Korisničko sučelje bi trebala biti intuitivna i prilagođena različitim veličinama ekrana, omogućavajući pristup funkcionalnostima aplikacije s bilo kojeg uređaja.

2.3 Tehnologije i alati

Sama aplikacija može se podijeliti u dva dijela: *frontend* i *backend*. *Backend* bi trebao biti razvijen koristeći programski jezik C# i ASP.NET Core tehnologiju, koja omogućuje izgradnju robusnog, sigurnog i skalabilnog sustava. ASP.NET Core pruža podršku za razvoj RESTful API-ja, autentifikaciju i autorizaciju korisnika, kao i integraciju s raznim bazama podataka, povećavajući funkcionalnost i pouzdanost aplikacije [1].

S druge strane, za *frontend* su korišteni React i Typescript, koji omogućuju stvaranje atraktivnog, dinamičkog i responzivnog korisničkog sučelja. React omogućuje učinkovito upravljanje stanjem aplikacije i komponentama [2], dok Typescript dodaje statičku tipizaciju, povećavajući sigurnost koda i olakšava održavanje [3]. Osim toga, za stiliziranje sučelja korišten je CSS-in-JS pristup, koji omogućuje modularnost i ponovnu upotrebu stilova.

Integracija između *frontenda* i *backenda* ostvaruje se putem RESTful API-ja, osiguravajući glatku komunikaciju i prijenos podataka. Ova arhitektura omogućuje fleksibilnost u razvoju, jer *frontend* i *backend* mogu biti razvijani i testirani odvojeno, ali funkcioniraju zajedno kao cjeloviti sustav.

3 Tehničko oblikovanje

Tijekom razvoja ove web aplikacije, kako je navedeno u specifikaciji zahtjeva, korišteni su programski jezici C# za *backend* i Typescript za *frontend*. Osim toga, za olakšanje razvoja web sustava korišteni su dodatni paketi i alati. U ovom dijelu detaljnije će se objasniti sve tehnologije koje su korištene.

3.1 C#

C# je objektno orijentirani programski jezik razvijen od strane Microsoft-a, pod vodstvom danskog softverskog inženjera Andersa Hejlsberga, u sklopu Microsoft .NET radnog okvira. Prva verzija jezika predstavljena je 2000. godine, a njegov razvoj kontinuirano se nastavlja sve do danas. Najnovija verzija, C# 12, objavljena je u studenom 2023. godine. C# vuče korijene iz obitelji programskih jezika, odnosno C i C++, a inspiraciju je uglavnom crpio iz C++ i Jave. [4]

Neke od glavnih značajki ovog programskog jezika, kako sam tvorac Andersa Hejlsberg govori su [4]:

- Objektno orijentirani jezik: C# podržava objektno orijentirane principe kao što su nasljeđivanje, polimorfizam, enkapsulacija i apstrakcija.
- Komponentno orijentirano programiranje: C# uključuje podršku za komponentno orijentirano programiranje, omogućujući stvaranje samostalnih i samodeskriptivnih paketa funkcionalnosti.
- Verzioniranje: Dizajn jezika uključuje mehanizme za kompatibilnu evoluciju programa i knjižnica tijekom vremena, smanjujući rizik od prekida rada pro-

grama prilikom uvođenja novijih verzija ovisnih knjižnica.

- Rukovanje iznimkama: Strukturiran i proširiv pristup otkrivanju i ispravljanju grešaka.
- Tipna sigurnost: Dizajn jezika onemogućava čitanje iz neinicijaliziranih varijabli, indeksiranje polja izvan njihovih granica ili izvođenje neprovjerenih pretvorbi tipova.
- Ujedinjeni sustav tipova: Svi C# tipovi, uključujući primitivne tipove kao što su *int* i *double*, nasljeđuju se od jednog korijenskog objektnog tipa, omogućujući konzistentno rukovanje vrijednostima svih tipova.
- Podrška za referentne i vrijednosne tipove: Omogućuje dinamičku dodjelu objekata kao i linijsko pohranjivanje laganih struktura.
- Garbage Collection: Automatsko upravljanje memorijom putem garbage collectora koji oslobađa memoriju zauzetu neiskorištenim objektima.

C# se koristi za različite svrhe, uključujući razvoj igara putem Unity platforme, izradu mobilnih i desktop aplikacija, te web aplikacija.

Jedna od najpoznatijih igara napisanih u C# pomoću Unity platforme su "Cuphead" i "Fall Guys". "Cuphead" je poznat po svom retro stilu i izazovnim borbama protiv neprijatelja, dok je "Fall Guys" postao globalni hit zbog svoje zabavne i kaotične mehanike za više igrača. [5]

Osim igara, pomoću C# je napravljeno puno poznatih aplikacija koje se svakodnevno koriste kao što su Microsoft Office Suite, Microsoft Skype, Microsoft Teams i mnoge druge. [6]

3.2 .NET

.NET je besplatna platforma otvorenog koda koja omogućuje izradu različitih vrsta aplikacija. U 2002. godini Microsoft je izbacio prvu inačicu .NET framework-a 1.0, dok je sadašnja verzija .NET framework 4.8. .NET pruža skup alata i biblioteka koji omogućuju razvoj i izvršavanje različitih vrsta softverskih aplikacija. Ovaj okvir

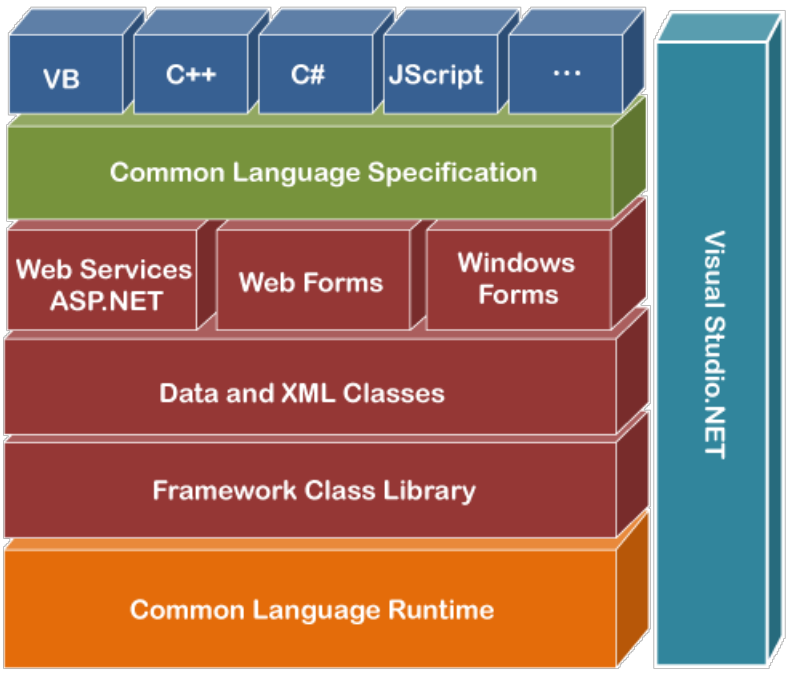
podržava više programskih jezika kao što su C#, Visual Basic.NET, i F#, što ga čini fleksibilnim za programere koji rade u različitim okruženjima. Glavni cilj .NET okvira je olakšati razvoj i održavanje aplikacija, te osigurati interoperabilnost između različitih komponenti softvera. .NET se izvršava na virtualnoj mašini koja omogućava pokretanje aplikacija na različitim platformama, uključujući Windows, Linux i macOS.

.NET radni okvir sastoji se od 8 komponenta prikazanih na slici 3.1: [7]

- CLR (Common Language Runtime) – CLR je odgovoran za upravljanje izvršavanje koda napisano na bilo kojem programskom jeziku
- CTS (Common Type System) – CTS predstavlja kakav se tip podataka i vrijednost može definirati i upravljati u memoriji računala tijekom izvođenja.
- BCL (Base Class Library)
- CLS (Common Language Specification) – CLS definira skup pravila i propisa koje bi trebale slijediti sve jezike koji spadaju pod .NET okvir.
- FCL (Framework Class Library) – FCL pruža funkcionalnosti sustava u .NET okviru, što uključuje klase, sučelja i tipove podataka
- .NET Assemblies – sadrže logički prevedeni kod u infrastrukturi zajedničkog jezika (CLI), koji se koristi za implementaciju, sigurnost i verzioniranje.
- XML Web Services
- Window Services

Arhitektura .NET Frameworka prikazana na slici 3.1 obuhvaća različite slojeve i komponente koje omogućuju razvoj aplikacija u više programskih jezika. Na vrhu slike nalaze se različiti programski jezici poput VB (Visual Basic), C++, C#, JScript i drugi, koji se mogu koristiti unutar .NET okruženja. Common Language Specification (CLS) osigurava interoperabilnost između tih jezika. Razvojne tehnologije kao što su Web Services ASP.NET, Web Forms i Windows Forms omogućuju izradu web i desktop aplikacija. Data and XML Classes sloj omogućuje rad s podacima i XML-om, dok Framework Class Library (FCL) pruža osnovne funkcionalnosti potrebne za razvoj aplikacija. Common Language Runtime (CLR) je temeljni sloj koji

omogućuje izvršavanje .NET aplikacija. Visual Studio .NET je integrirano razvojno okruženje koje podržava sve navedene slojeve i tehnologije, omogućujući programerima učinkovit razvoj aplikacija.



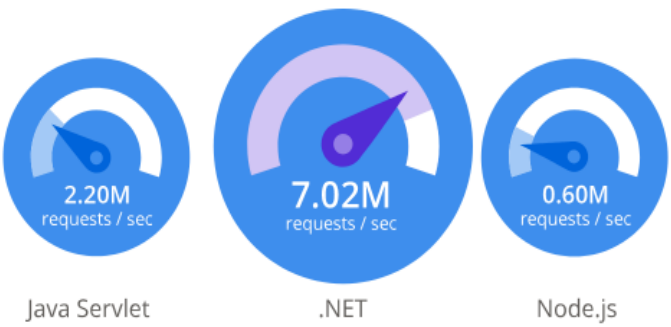
Slika 3.1 Detaljan prikaz arhitekture .NET radnog okvira, koji prikazuje kako se .NET okruženje integrira s različitim servisima
Slika preuzeta sa:
`https://www.javatpoint.com/vb-net-dot-net-framework-introduction`

3.3 ASP.NET i ASP.NET Core

ASP.NET je web okvir nastao od strane Microsoft-a 2002. godine kao dio .NET razvojnog okvira. ASP.NET proširuje .NET s alatima i knjižnicama za olakšavanje razvoja web aplikacija. Neke od glavnih značajki koje ASP.NET nadodaje na .NET su sintaksa za predloške web-aplikacijama, poznatija kao Razor. Nadalje, ASP.NET omogućava sustav autentifikacije koji uključuje knjižnice, bazu podataka i predloške stranica za upravljanje prijavama, uključujući višefaktorsku autentifikaciju i vanjsku autentifikaciju s Googleom, X i drugim uslugama. Također, ASP.NET pruža isticanje

sintakse, automatsko dovršavanje koda i druge funkcionalnosti specifične za razvoj web stranica.

ASP.NET Core je razvijen kao nadogradnja na ASP.NET s ciljem poboljšanja performansi, učinkovitosti i smanjenja potrošnje računalnih resursa. Slika 3.2 prikazuje usporedbu brzine ASP.NET Core-a s drugim radnim okvirima.



Slika 3.2 Usporedba brzine radnih okvira
Slika preuzeta sa: `https://dotnet.microsoft.com/en-us/apps/aspnet`

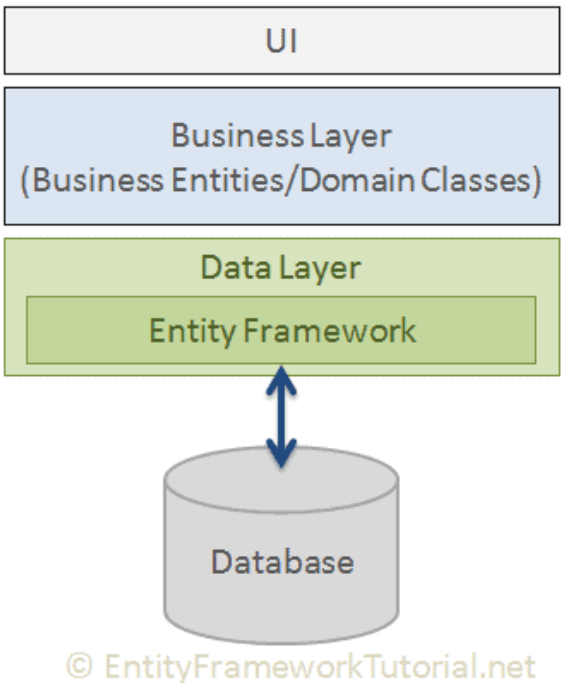
Osim poboljšanja performansi ASP.NET Core nudi i paralelno postojanje više verzija na istom serveru. To omogućuje da jedna aplikacija koristi najnoviju verziju, dok druge aplikacije mogu nastaviti raditi na verzijama na kojima su testirane. Ova značajka je posebno korisna za organizacije koje žele postupno nadograđivati svoje aplikacije bez prekida rada postojećih sustava.

3.4 Dodatni paketi

3.4.1 Entity framework

Entity framework je ORM (objektno orijentirani mapper) okvir razvijen od strane Microsoft-a kako bi se olakšao rad s bazama podataka u sklopu .NET aplikacija. Prije uvođenja Entity framework-a, programeri su morali ručno pisati kod za pristup bazi podataka koristeći ADO.NET ili slične tehnologije. Taj proces je uključivao

otvaranje veze sa bazom podataka, izvođenje SQL upita, te pretvaranje podataka između relacijskih tablica i objekata u aplikaciji. Ovaj proces je bio zamoran i sklon pogreškama. Zbog toga je Microsoft predstavio Entity framework kako bi automatizirao i olakšao ove zadatke. Korištenjem EF-a, programeri mogu raditi s podacima koristeći objektno-orijentirani pristup, gdje se relacijske tablice mapiraju na domenske klase u aplikaciji, kako je prikazano na slici 3.3. To omogućuje rad s podacima na višoj razini apstrakcije, eliminirajući potrebu za većinom ručno pisanog koda za pristup podacima. [8]



Slika 3.3 Položaj Entity framework-a u .NET aplikaciji
Slika preuzeta sa: `https://www.entityframeworktutorial.net/entityframework6/what-is-entityframework.aspx`

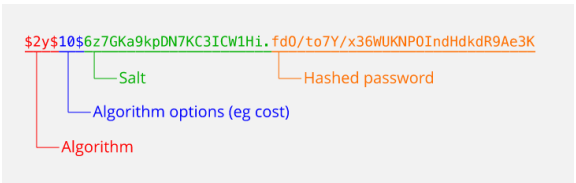
3.4.2 AutoMapper

AutoMapper je knjižnica u .NET okruženju koja služi za automatsko mapiranje objekata jedne klase na objekte druge klase. Ovaj alat je osmišljen kako bi riješio problem repetitivnog i dosadnog pisanja koda za mapiranje svojstava između različitih obje-

kata, što je često potrebno prilikom transformacije podataka između slojeva aplikacije (npr. između sloja poslovne logike i sloja podataka). Korištenjem AutoMappera, programeri mogu značajno smanjiti količinu ručno pisanog koda, povećati produktivnost i smanjiti mogućnost grešaka. AutoMapper omogućuje jednostavno definiranje pravila mapiranja te podržava kompleksne scenarije, uključujući mapiranje kolekcija i rad s različitim ORM-ovima kao što su Entity framework i ADO.NET.[9]

3.4.3 BCrypt.NET

BCrypt je popularna funkcija za hashiranje lozinki temeljena na algoritmu šifre *Blowfish*, stvorena za operativni sustav OpenBSD, koja je predstavljena 1999. godine. Koristi se za sigurno pohranjivanje lozinki kako bi se spriječilo njihovo kompromitiranje u slučaju proboja podataka. BCrypt automatski generira slučajne vrijednosti koje čine hashove jedinstvenima i težim za probiti metodom *brute force* napada. Također je prilagodljiva funkcija, što znači da može povećati računalnu složenost izračuna hashova kroz svoj parametar radnog faktora. Ova prilagodljivost osigurava da hashiranje ostane učinkovito čak i s povećanjem računalne snage tijekom vremena. BCrypt je dobro testiran i smatra se sigurnijim od mnogih drugih dostupnih algoritama za hashiranje, što ga čini savršenim izborom za zaštitu lozinki u modernim aplikacijama.[10]



Slika 3.4 Primjer BCrypt-a za pohranjivanje lozinke
Slika preuzeta sa: `https://kompot.petrsu.ru/php-manual/faq.passwords.html`

3.4.4 xUnit.net i Moq

xUnit.net je besplatan, otvoreni alat za jedinično testiranje u .NET framework. Predstavlja najnoviju tehnologiju za jedinično testiranje jezika kao što su C#, F#,

VB.NET i drugi .NET jezici. xUnit.net radi s alatima poput ReSharper, CodeRush, TestDriven.NET i Xamarin, te je dio .NET Foundation. Omogućava programerima pisanje i izvršavanje jediničnih testova za .NET aplikacije.

Moq je popularan alat za izradu mock objekata koji se koristi zajedno s xUnit.net za jedinično testiranje. Moq omogućava simuliranje ponašanja ovisnosti koje testirani kod koristi, omogućujući izolaciju jediničnih testova.

Mock objekt je simulirani objekt koji oponaša ponašanje stvarnog objekta u kontroliranim uvjetima. Koristi se u jediničnom testiranju kako bi se izolirao kod koji se testira od njegovih ovisnosti. Na primjer, ako metoda A ovisi o metodi B, umjesto stvarne implementacije metode B koristi se mock verzija. Mock objekt može se konfigurirati da vraća određene vrijednosti kada se pozovu njegove metode, provjerava koliko puta je određena metoda pozvana ili provjerava redoslijed poziva metoda. To omogućava testerima da se usredotoče na testiranje funkcionalnosti metode A bez utjecaja metode B.

Kombinacija Moq i xUnit.net omogućava precizno testiranje funkcionalnosti aplikacija bez potrebe za stvarnim ovisnostima, čime se poboljšava učinkovitost i pouzdanost jediničnih testova. Korištenjem Moq-a, programeri mogu jednostavno kreirati mock objekte za sve vanjske ovisnosti i tako izolirati kod koji se testira, što rezultira bržim testiranjem.

3.5 React.js

React.js je JavaScript knjižnica otvorenog koda osnovana od strane Facebook-a, osmišljena sa ciljem pojednostavljivanja procesa izrade korisničkog sučelja. React omogućava izradu korisničkog sučelja koristeći komponente, koje predstavljaju manje, samostalne dijelove koda. Jedna od ključnih značajki React-a je virtualni DOM (Document Object Model), koji omogućava brže i učinkovitije ažuriranje korisničkog sučelja usporedbom novih promjena s virtualnim prikazom stvarnog DOM-a prije nego što se promjene provedu, što značajno poboljšava performanse aplikacija.

Tradicionalni način učitavanja web stranica uključuje slanje zahtjeva poslužitelju za svaku novu stranicu, što može biti neučinkovito za podatkovno intenzivne

web stranice. React koristi pristup jednostraničnih aplikacija (SPA), gdje se učitava samo jedan HTML dokument prilikom prvog zahtjeva, a zatim se ažuriraju samo potrebni dijelovi stranice pomoću JavaScript-a. Ovaj pristup, poznat kao klijentsko usmjeravanje, sprječava ponovno učitavanje cijele stranice pri svakoj promjeni, što rezultira boljim performansama i dinamičnijim korisničkim iskustvom. React također omogućava pisanje koda u JavaScript-u i TypeScript-u, pružajući fleksibilnost programerima u odabiru jezika koji najbolje odgovara njihovim potrebama i preferencijama. [2]

3.6 Typescript

TypeScript je objektno orijentirani programski jezik koji se nadograđuje na JavaScript. Kao i C#, TypeScript je nastao pod vodstvom Andersa Hejlsberga. TypeScript je osmišljen kao alat koji pruža bolje alate i iskustvo pri izradi korisničkog sučelja. Sastoji se od tri glavne komponente: jezika, TypeScript kompajlera i TypeScript jezične usluge.

Prva komponenta, jezik, uključuje sintaksu, ključne riječi i oznake tipova. TypeScript kompajler ima zadaću pretvoriti kod napisan u TypeScript-u u ekvivalentni JavaScript kod, omogućujući da se TypeScript kod može izvršavati na bilo kojoj platformi koja podržava JavaScript. Treća komponenta je TypeScript jezična usluga, koja pruža značajke poput dovršavanja dijelova koda, pomoć s formatiranjem koda, obojenja koda i druge funkcionalnosti koje olakšavaju rad programera.

Jedna od ključnih prednosti TypeScript-a je statička provjera tipova koja omogućuje otkrivanje grešaka tijekom razvoja, prije izvršavanja koda. TypeScript također podržava napredne koncepte objektno orijentiranog programiranja kao što su klase, sučelja i nasljeđivanje, čime poboljšava strukturu i održivost koda. [3]

3.7 Dodatni paketi za React

React je vrlo moćna knjižnica sama po sebi, ali njezina puna funkcionalnost dolazi do izražaja kada se kombinira s dodatnim knjižnicama koje pojednostavljuju stvaranje

korisničkog sučelja.

Material UI je otvorena React komponentna biblioteka koja implementira Googleov Material Design. Uključuje sveobuhvatan skup unaprijed izrađenih komponenti koje su spremne za upotrebu. Material UI pruža širok raspon opcija za prilagodbu, što olakšava implementaciju vlastitog dizajna na temelju ponuđenih komponenti. Neke od prednosti korištenja Material UI su brza isporuka i prilagodljivost. Material UI je dio MUI Core, koji također nudi Base UI i Joy UI. [11]

4 Implementacija

U ovom poglavlju detaljno se opisuje proces implementacije web sustava, uključujući korištene alate, pristupe razvoju baze podataka te ključne komponente sustava. Fokusira se na primjenu specifičnih tehnologija i alata koji omogućuju učinkovit razvoj i upravljanje aplikacijom. Ova faza obuhvaća izbor razvojnih alata, implementaciju baze podataka, definiranje entiteta i njihovu međusobnu povezanost, kao i razvoj kontrolera i upravljanje korisničkom autentifikacijom.

4.1 Korišteni alati

Kako bi se olakšala izrada web sustava tijekom faze implementacije korišteni su sljedeći alati: Visual Studio, Visual Studio Code, Microsoft SQL Server Management Studio, Git i GitHub

4.1.1 Visual Studio

Visual Studio je integrirano razvojno okruženje (IDE) osnovano od strane Microsofta. Namijenjen je programerima kao sveobuhvatan alat za pisanje, uređivanje, *debugiranje* i izradu koda, kao i za implementaciju aplikacija. Visual Studio podržava više programskih jezika, uključujući C++, C#, JavaScript, TypeScript, Python i mnoge druge. [12]

Neke od ključnih karakteristika Visual Studija uključuju bogatu kolekciju korisnih paketa koji olakšavaju razvoj aplikacija. Također, nudi širok spektar predložaka koji pokrivaju različite vrste razvoja, kao što su igre, mobilni razvoj, web razvoj, razvoj

API-ja i mnoge druge. Ovi predlošci pružaju programerima brz početak i uštedu vremena u postavljanju osnovne infrastrukture za njihove projekte.

Visual Studio dolazi u tri glavne verzije: *Community*, *Professional* i *Enterprise*. *Community* verzija je besplatna i namijenjena je pojedinačnim programerima, studentima i otvorenim projektima. *Professional* verzija nudi dodatne alate i funkcionalnosti pogodne za manje i srednje timove, dok *Enterprise* verzija pruža najnaprednije alate i funkcionalnosti za velike organizacije i poduzeća s naprednim zahtjevima za razvoj softvera.

Za razvoj ovog web sustava korištena je *Community* verzija Visual Studija. Visual Studio korišten jer nudi korisne pakete specifične za razvoj ASP.NET sustava, što značajno olakšava izradu web sustava.

4.1.2 Visual Studio Code

Visual Studio Code je besplatni, višeplatformski uređivač koda koji podržava izgradnju aplikacija i podržava mnoge programske jezike. Jedna od glavnih karakteristika Visual Studio Code-a uključuje podršku za veliki broj paketa. Također, VS Code olakšava produktivnost putem isticanja sintakse, automatskih dodavanja zagrada, automatskog uvlačenja i ostalih funkcionalnosti. Nadalje, VS Code podržava Git, omogućavajući rad s kontrolom verzija izravno iz uređivača, uključujući pregled razlika u promjenama. Dodatno, VS Code uključuje bogatu ugrađenu podršku za Node.js razvoj s JavaScriptom i TypeScriptom.

4.1.3 Microsoft SQL Server Management Studio

Microsoft SQL Server Management Studio (SSMS) je integrirano okruženje za upravljanje bilo kojom SQL infrastrukturom. Koristi se za pristup, konfiguraciju, upravljanje, administraciju i razvoj svih komponenti SQL Servera, Azure SQL baze podataka, Azure SQL Managed Instance i ostalog. SSMS nudi bogat set alata uključujući mogućnost izrade i upravljanja dijagramima baza podataka, skriptiranje objekata baze podataka kao što su tablice, pohranjene procedure, funkcije, mogućnost izrade, uređivanja i izvršavanja SQL upita i skripti. Također, nudi i integraciju s alatima za

upravljanje verzijama kao što je Git za kontrolu izvornog koda i mogućnost rada s više instanci SQL Servera i Azure SQL servisa iz jednog sučelja. Za potrebe izrade web sustava, korišten je SSMS 20 kako bi se olakšalo upravljanje bazom podataka i razvojne operacije vezane uz podatke. [13]

4.1.4 Git i GitHub

Git je distribuirani sustav za kontrolu verzija (VCS) koji se koristi za upravljanje izvorima koda. On se razlikuje od ostalih VSC-ova jer ne pohranjuje promjene na razini datoteka, već kao snimke ili *snapshotove* cijelog projekta u određenom trenutku. Svaki *commit* u Git-u predstavlja točnu kopiju svih datoteka u tom trenutku, što olakšava praćenje povijesti promjena i povratka na prethodne verzije.

Jedna od ključnih prednosti Git-a je lokalnost operacija. Većina operacija, poput pregleda povijesti projekta ili usporedbe verzija datoteka, izvršava se lokalno bez potrebe za mrežnom vezom prema centralnom serveru. To znači da su operacije brze i gotovo trenutne, što posebno dolazi do izražaja kod velikih projekata s kompleksnom poviješću.

Git se često koristi uz GitHub, web-platformu koja omogućuje pohranu Git repozitorija i pruža mnoge dodatne funkcionalnosti. GitHub olakšava suradnju među timovima omogućujući pisanje dokumentacije, stvaranje i praćenje problema te upravljanje zadacima. Osim toga, GitHub nudi alate poput GitHub Actions za automatizaciju raznih zadataka, čime se povećava učinkovitost i organizacija u razvojnim projektima. [14]

4.1.5 Swagger

Swagger je alat koji se koristi za dizajn, izradu dokumentacije i testiranje RESTful API-ja. Omogućava programerima da opišu strukturu svojih API-ja pomoću specijaliziranog jezika nazvanog *OpenAPI Specification* (OAS). Swagger pomaže u stvaranju interaktivne dokumentacije koja olakšava razumijevanje i korištenje API-ja te omogućava testiranje funkcionalnosti API-ja izravno iz preglednika. [15]

4.2 Implementiranje baze

ASP.NET koristi NuGet paket Microsoft EntityFramework Core, koji pojednostavljuje proces kreiranja baze podataka i tablica unutar nje. Zahvaljujući EntityFramework Core paketu, za kreiranje baze podataka i potrebnih tablica nije bilo nužno koristiti SQL naredbe izravno u bazi, već je to bilo moguće obaviti direktno iz Visual Studija. Entiteti su kreirani pomoću klasa. Primjer implementacija tablice **Users** prikazana je u nastavku:

```
public class Users : BaseEntity
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    [JsonIgnore] public string Password { get; set; }
    public ICollection<Space> Spaces { get; set; }
    public ICollection<WorkItem> WorkItems { get; set; }

    public ICollection<UserSpace> UserSpaces { get; set; }
}
```

Primjer implementacije entiteta *Users* prikazuje osnovnu strukturu korisnika u sustavu. Klasa *Users*, koja nasljeđuje *BaseEntity*, uključuje atribute kao što su ime, prezime i email, dok je lozinka označena s *JsonIgnore*, pa se ne prikazuje prilikom dohvaćanja korisnika radi sigurnosti. Osim toga, entitet sadrži kolekcije *Spaces* i *WorkItems* za povezane prostore i zadatke, te *UserSpaces* za upravljanje vezama između korisnika i prostora.

Nakon što su svi entiteti definirani, sljedeći korak je uspostavljanje veza između njih. To je ostvareno korištenjem klase *ApplicationDbContext*, koja nasljeđuje *DbContext* kao svoju osnovnu klasu. Unutar ove klase, entiteti su predstavljeni kao *DbSet* svojstva. Klasa također uključuje metodu *OnModelCreating*, koja uzima *ModelBuilder* kao argument i koristi se za definiranje specifičnih veza između entiteta.

Kako bi se ove veze definirale, korišten je *ICollection*, koja omogućava rad s grupama entiteta na način sličan listama ili nizovima. U nastavku je prikazan način kako se veza između entiteta *Space* i *Users* uspostavlja. U ovom kodu, entitet *Space* ima referencu na entitet *Users*, što znači da svaki *Space* pripada određenom korisniku. U *OnModelCreating* metodi, konfigurira se veza između *Space* i *Users*, pri čemu se specificira da jedan *Users* može imati više *Space* objekata, dok svaki *Space* pripada jednom *Users* entitetu. Također je postavljeno pravilo brisanja s *Cascade* ponašanjem, što znači da će se svi povezani *Space* objekti automatski obrisati ako se obriše odgovarajući *Users* entitet.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<Space>()
        .HasOne(space => space.Users)
        .WithMany(users => users.Spaces)
        .HasForeignKey(space => space.UsersId)
        .OnDelete(DeleteBehavior.Cascade);

}
```

Kako bi se uspješno spojili s bazom podataka, bilo je potrebno u datoteci *appsettings.json* definirati osnovne podatke o bazi. Datoteka *appsettings.json* sadrži konfiguracije koje uključuju logiranje, dopuštene *hostove* i *stringove* za povezivanje. Na primjer, pod *ConnectionStrings* definiran je *string* koji detaljno opisuje kako se aplikacija povezuje s lokalnom bazom podataka, uključujući server, ime baze, i opcije povezivanja. Kako bi sustav pravilno funkcionirao bilo je potrebno nadodati *Trusted_Connection* koji omogućava korištenje *Windows* autentikacije za povezivanje s bazom podataka i *TrustServerCertificate* koji omogućava prihvaćanje SSL/TLS certifikata servera bez provjere njegove valjanosti. Ovdje je naveden primjer kako izgleda sadržaj *appsettings.json*:

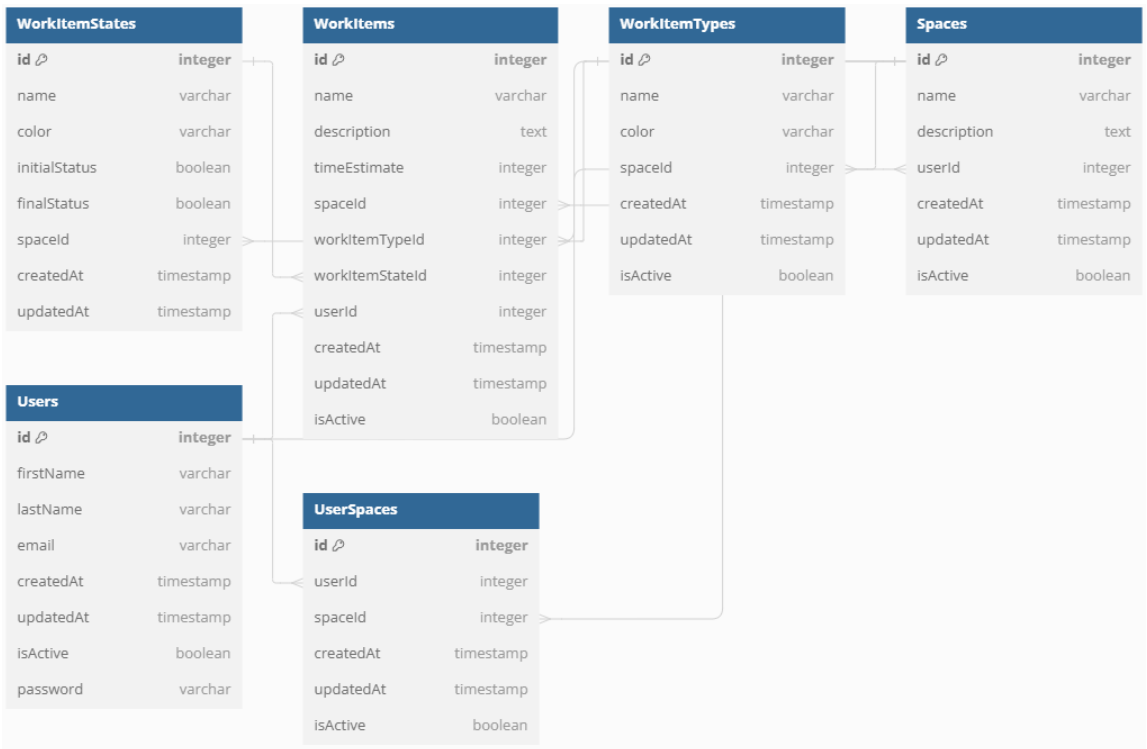
```
{
```

```
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "local": "Server=DESKTOP-ORT3GHN\\SQLEXPRESS;
    Database=WorkManagerSystem;
    Trusted_Connection=true;
    TrustServerCertificate=true;"
  }
}
```

Zadnji korak za uspostavljanje veze s bazom podataka je izvršenje migracije. Migracija se provodi pomoću *Package Manager Console*, koristeći naredbe *Add-Migration* i *Update-Database*. Ove naredbe dodaju i ažuriraju bazu podataka, stvarajući novonastale tablice vidljivima u SQL Server Management Studio (SSMS).

Baza web sustava se sastoji od šest entiteta: korisnici, prostori, korisnici prostora, zadaci, tipovi zadataka i stanje zadataka.

Na slici 4.1 prikazana je baza podataka sustava u koju se prijavljuje korisnik. Svaki korisnik može imati više prostora, a svaki prostor može sadržavati više korisnika. Veza između korisnika i prostora je više-na-više, zbog čega se dodaje dodatni entitet korisnici prostora koji povezuje korisnike s pripadajućim prostorima. Svaki prostor može imati više zadataka, ali svaki zadatak može pripadati samo jednom prostoru, čime se uspostavlja veza više-na-jedan između prostora i zadataka. Svaki zadatak ima jedan tip zadatka koji je vezan za prostor. Također, svaki zadatak ima samo jedno stanje.



Slika 4.1 Prikaz baze podataka

4.3 Entiteti

A.1 Bazni entit (BaseEntity)

Klasa *BaseEntity* stvorena je s namjerom da bude temelj za sve druge entitete. Uključuje identifikacijsku oznaku ID, koji funkcionira kao primarni ključ. Također sadrži atribute *CreatedAt* i *UpdatedAt*, koji označavaju vrijeme stvaranja i vrijeme zadnjeg ažuriranja. Dodatno, uveden je atribut *isActive* koji provjerava je li entitet aktivan.

```
public abstract class BaseEntity
{
    [Key]
    public long Id { get; set; }
```

```
        public DateTime CreatedAt { get; set; } = DateTime.Now;
        public DateTime UpdatedAt { get; set; } = DateTime.Now;
        public bool isActive { get; set; } = true;
    }
```

A.2 Korisnici (Users)

Entitet *Users* sadrži informacije o korisnicima sustava. Sastoji se od atributa *FirstName* i *LastName*, koji predstavljaju ime i prezime korisnika. Nadalje korisnik se prijavljuje korisničkim računom, koji je jedinstvena e-mail adresa korisnika. Posljednje, korisnik sadrži *Password*, koji pohranjuje lozinku korisnika. Entitet *Users* je povezan s entitetima *Space* i *UserSpaces*.

A.3 Prostori (Spaces)

Entitet *Spaces* predstavlja prostore koji sadrže zadatke. Atributi ovog entiteta uključuju *Name* kao naziv prostora, *Description* za opis *Space-a*, *UserId* koji je strani ključ povezan s entiteom *Users*. *Spaces* su povezani s entitetima *WorkItemTypes*, *WorkItemStates* i *WorkItems*.

A.4 Korisnici Prostora (UserSpaces)

Kao posljedica veze više-na-više između entiteta *Users* i *Spaces*, formiran je entitet *UserSpaces*. Sastoji se od atributa od dva strana ključa, *UserId* koji je povezan s *Users* i *SpaceId*, koji je povezan sa *Spaces*. Ovaj entitet omogućava da jedan korisnik bude povezan s više prostora i da jedan prostor ima više korisnika.

A.5 Zadaci (WorkItems)

Pojedinačni zadaci unutar sustava predstavljeni su kroz entitet *WorkItems*. Ovaj entitet uključuje različite atribute kao što su *Name* za naziv zadatka, *Description* za opis zadatka, *TimeEstimate* za procjenu vremena potrebnog za izvršenje, te strane ključeve *SpaceId* koji je povezan sa *Spaces*, *WorkItemTypeId* povezan sa *WorkItemTypes*, *WorkItemStateId* povezan sa *WorkItemStates*, i

UserId povezan sa *Users*. *WorkItems* je ključni dio sustava jer omogućavaju praćenje i organizaciju radnih aktivnosti.

A.6 Tipovi zadataka (WorkItemTypes)

Svaki zadatak uključuje svoj tip, koji je definiran entitetom *WorkItemTypes*. Sastoji se od atributa *Name* za naziv tipa zadatka, *Color* za boju koja se koristi za vizualno razlikovanje tipova zadataka, *SpaceId* kao strani ključ povezan sa *Spaces*. Ovaj entitet omogućava organizaciju zadataka prema njihovim tipovima, olakšavajući upravljanje i filtriranje zadataka.

A.7 Stanje zadataka (WorkItemStates)

Entitet *WorkItemStates* predstavlja različite statuse koje zadatak može imati. Atributi uključuju *Name* za naziv statusa, *Color* za boju povezanu sa statusom, *InitialStatus* koji označava početni status, *FinalStatus* koji označava završni status, *SpaceId* kao strani ključ povezan sa *Spaces*. Stanje zadataka omogućava praćenje napretka zadataka kroz različite faze od početka do završetka.

4.4 Kontroleri

Kako bi se moglo upravljati, dodavati i modificirati podatke unutar baze podataka, za svaki entitet bilo je potrebno dodati njegov kontroler. Kontroler je klasa koja nasljeđuje superklasu *ControllerBase*, a cilj mu je manipuliranje podacima u bazi podataka. Svaki kontroler se sastoji od putanje (eng. path), konteksta i *mapper-a*, te funkcionalnosti za manipulaciju podataka, primjerice dodavanje (HttpPost).

Mapper služi za pretvaranje (mapiranje) objekata iz jednog oblika u drugi. Na primjer, može se koristiti za pretvaranje DTO (Data Transfer Object) objekata u entitete baze podataka i obrnuto. U nastavku je prikazan primjer DTO-a:

```
public class SpaceCreateDto
{
    public string Name { get; set; }
```

```
    public string Description { get; set; }
    public long UsersId { get; set; }
}
```

Kontekst (ApplicationDbContext) predstavlja sesiju s bazom podataka koja omogućava upite i spremanje podataka. Kontekst također sadrži *DbSetove* koji predstavljaju kolekcije entiteta. Svaki kontroler sastoji se od četiri dijela: dodavanje podataka, dohvaćanje podataka, brisanje podataka i ažuriranje podataka.

Dodavanje podataka

Dodavanje podataka funkcionira na način da se instacira novi objekt te se dohvaćaju uneseni podaci iz DTO-a, koji se zatim mapiraju pomoću metode *CreateMap*. Ova metoda prima argumente DTO-a i klase entiteta koju želimo stvoriti. Pomoću konteksta podaci se dodaju u bazu podataka, te se nakraju spremaju.

```
[HttpPost]
[Route("Create")]
public async Task<ActionResult<int>> CreateSpace
    ([FromBody] SpaceCreateDto dto)
{
    Space newSpace = _mapper.Map<Space>(dto);
    await _context.Spaces.AddAsync(newSpace);
    await _context.SaveChangesAsync();

    return Ok(newSpace.Id);
}
```

Dohvaćanje podataka

Podaci se dohvaćaju iz baze pomoću konteksta, a zatim se mapiraju u odgovarajući DTO objekt za vraćanje korisniku. Ova metoda koristi kontekst za dohvaćanje svih zapisa iz baze podataka, uključujući povezane entitete, a zatim koristi mapper za pretvaranje tih podataka u DTO format.

```
[HttpGet]
[Route("Get")]
public async Task<ActionResult<IEnumerable<SpaceGetDto>>> GetSpaces()
{
    var spaces = await _context
        .Spaces
        .Include(space => space.Users)
        .ToListAsync();
    var convertedSpaces = _mapper
        .Map<IEnumerable<SpaceGetDto>>(spaces);
    return Ok(convertedSpaces);
}
```

Brisanje podataka

Brisanje podataka se provodi tako da se prvo dohvaća objekt koji želimo obrisati pomoću njegove identifikacijske oznake ID iz baze podataka, a zatim se uklanja iz konteksta, te se promjene spremaju.

```
[HttpDelete("{spaceId}")]
public async Task<IActionResult> DeleteSpaceById(long spaceId)
{
    try
    {
        var spaceToDelete = await _context.Spaces.FindAsync(spaceId);

        if (spaceToDelete == null)
        {
            return NotFound($"Space with ID {spaceId} not found.");
        }

        _context.Spaces.Remove(spaceToDelete);
        await _context.SaveChangesAsync();
```

```
        return Ok($"Space with ID {spaceId} deleted successfully.");
    }
    catch (Exception ex)
    {
        return BadRequest($"Failed to delete space: {ex.Message}");
    }
}
```

Ažuriranje podataka

Ažuriranje podataka u bazi se provodi tako da se prvo dohvaća objekt koji želimo ažurirati pomoću njegove identifikacijske oznake ID, zatim se primjenjuju promjene pomoću mapper-a i na kraju se promjene spremaju u bazu.

```
[HttpPut]
[Route("UpdateEmail")]
public async Task<IActionResult> UpdateUserEmail
    ([FromBody] UpdateUserEmailDto dto)
{
    var user = await _context.Users.FindAsync(dto.Id);
    if (user == null)
    {
        return NotFound("User not found");
    }

    // Ažuriranje korisnika koristeći AutoMapper
    _mapper.Map(dto, user);

    await _context.SaveChangesAsync();

    return Ok(new { message = "User email updated successfully" });
}
```

Autentifikacija korisnika

Registracija korisnika je realizirana na način da se na početku kreira novi korisnik. Nadalje, pomoću paketa BCrypt, šifra se kriptira radi sigurnosti. Također, generira se token za autentifikaciju korisnika.

Autentifikacija korisnika provodi se putem JWT tokena. JWT (JSON Web Token) je kompaktan, URL-siguran način za prijenos tvrdnji između dvije strane. JWT tokeni se koriste za autentifikaciju i autorizaciju korisnika, omogućujući sigurnu razmjenu informacija. Metoda za generiranje JWT tokena radi na sljedeći način: prvo se kreira sigurnosni ključ koristeći tajni ključ (secureKey) koji je kodiran u UTF-8 formatu. Zatim se stvaraju vjerodajnice za potpisivanje tokena koristeći HMAC-SHA256 algoritam. Kreira se zaglavlje JWT tokena koristeći prethodno definirane vjerodajnice,a zatim se kreira korisni teret (eng. payload) tokena koji sadrži identifikacijsku oznaku korisnika kao jedinstvenu identifikaciju, kao i vrijeme isteka tokena (1 dan od današnjeg dana). Nakon toga se kreira sigurnosni token koji kombinira zaglavlje i korisni teret. Na kraju, metoda *WriteToken* iz *JwtSecurityTokenHandler* klase koristi serijalizaciju sigurnosnog tokena u obliku JWT *stringa* koji se može koristiti za autentifikaciju korisnika.

```
public string Generate(long id)
{
    var symmetricSecurityKey = new SymmetricSecurityKey
        (Encoding.UTF8.GetBytes(secureKey));
    var credentials = new SigningCredentials(
        symmetricSecurityKey,
        SecurityAlgorithms.
        HmacSha256Signature);
    var header = new JwtHeader(credentials);

    var payload = new JwtPayload(
        id.ToString(),
        null,
        null,
```

```
        null,
        DateTime.Today.AddDays(1));

    var securityToken = new
    JwtSecurityToken(header, payload);

    return new JwtSecurityTokenHandler()
    .WriteToken(securityToken);
}
```

Nakon što se token generira, pomoću kolačića dodajemo JWT token i na kraju šaljemo poruku da je korisnik uspješno kreiran.

```
Response.Cookies.Append("jwt", jwt, new CookieOptions {
    HttpOnly = true,
    SameSite = SameSiteMode.None,
    Secure = true
});

return Ok(new { message = "User Created Successfully" });
```

Navedeni dio koda postavlja JWT token u kolačić s opcijama *HttpOnly*, označavajući da kolačić nije dostupan putem JavaScript-a. *SameSite* je postavljen na *None* omogučavajući slanje kolačića preko domena,a *Secure* je postavljen na *True* osiguravajući da se kolačić šalje samo preko HTTPS-a. Nakon toga, vraća se odgovor s porukom "User Created Successfully".

Prijava funkcionira slično kao i registracija. Prvo se pretražuje korisnik na temelju unesenog e-maila koji se nalazi u DTO-u. Nakon što je korisnik pronađen, provjerava se ispravnost lozinke, te se, ukoliko je lozinka netočna, vraća pogrešku. Ako je lozinke točna, kao i kod registracije, pomoću kolačića dodajemo JWT token.

```
[HttpPost]
[Route("Login")]
```

```
public async Task<IActionResult> LoginUser
    ([FromBody] UserLoginDto dto)
{
    var user = GetUserByEmail(dto.Email);

    if (user == null)
    {
        return BadRequest("Invalid email");
    }

    if (!BCrypt.Net.BCrypt.Verify(dto.Password,user.Password))
    {
        return BadRequest("Invalid password");
    }

    var jwt = _jwtService.Generate(user.Id);

    Response.Cookies
    .Append("jwt", jwt,
    new CookieOptions
    {
        HttpOnly = true,
        SameSite = SameSiteMode.None,
        Secure = true
    });

    return Ok(new
    {
        message = "success"
    });
}
```

Ukoliko se korisnik želi odjaviti sa svog računa, to se ostvaruje na način da se sadržaj kolačića izbriše.

5 Rezultati

Ovo poglavlje detaljnije predstavlja ključne rezultate projekta, obuhvaćajući implementirane funkcionalnosti, provedenja testiranja te evaluaciju performansi sustava. Na početku se detaljno opisuju funkcionalnosti razvijenog grafičkog sučelja, koje korisnicima omogućava intuitivnu interakciju s podacima pohranjenim u bazi. Sljedeće, kroz prikaz testiranja, pokazuje se kako je osigurana pouzdanost i ispravnost sustava korištenjem automatskih testova. Na krajuse analiziraju performanse sustava kako bi se procijenila njegova učinkovitost i skalabilnost.

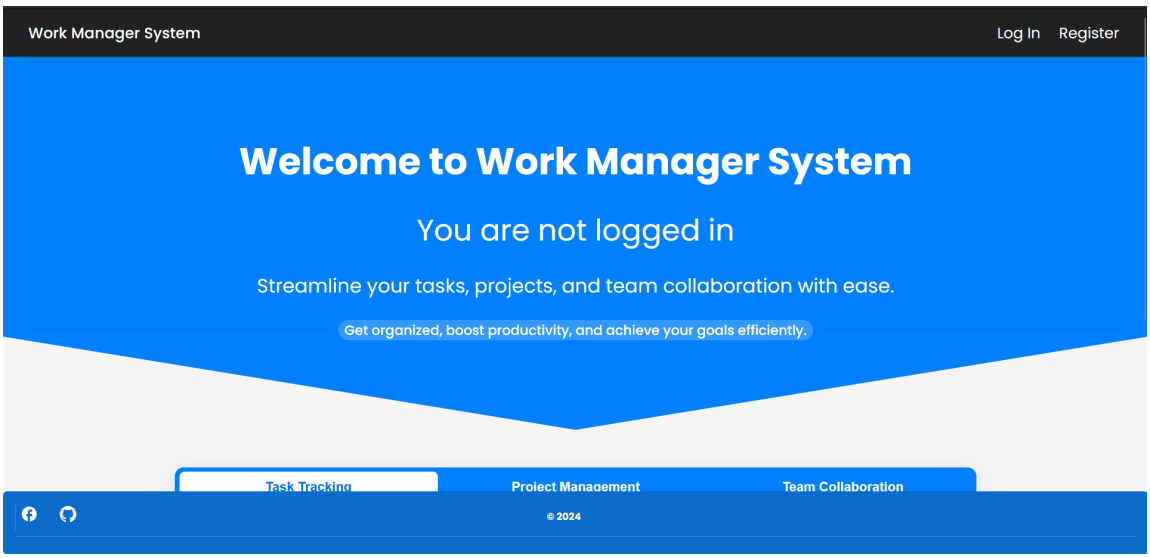
5.1 Funkcionalnosti

Nakon što je uspješno implementirana baza podataka, sljedeći ključni korak u razvoju sustava je implementacija grafičkog sučelja. Ova cjelina fokusira se na kreiranje intuitivnog i korisnički pristupačnog sučelja koje omogućava interakciju s podacima pohranjenim u bazi. Kroz ovu fazu, detaljno je objašnjeno kako su dizajnirane i razvijene različite komponente sučelja. Posebna pažnja posvećuje se funkcionalnostima koje omogućuju korisnicima jednostavno upravljanje zadacima, vizualizaciju informacija te učinkovitu navigaciju kroz sustav.

5.1.1 Početna stranica

Kada korisnik otvori web sustav, prva stranica koja se učita je početna stranica. Početna stranica sastoji se od navigacijske trake, pozdravne poruke i kratkog opisa funkcionalnosti sustava, te podnožja. Na vrhu stranice nalazi se navigacijska traka, koja omogućava pristup početnoj stranici, prijavi i registraciji.

Kao što je prikazano na slici 5.1, u sredini stranice prikazana je pozdravna poruka, te obavijest je li korisnik prijavljen u sustav. Također, ispod te poruke nalazi se kratak opis funkcionalnosti sustava koji naglašava kako sustav pomaže u organiziranju zadataka, projekata i timskoj suradnji.



Slika 5.1 Prikaz početne stranice

5.1.2 Prijava i registracija u sustav

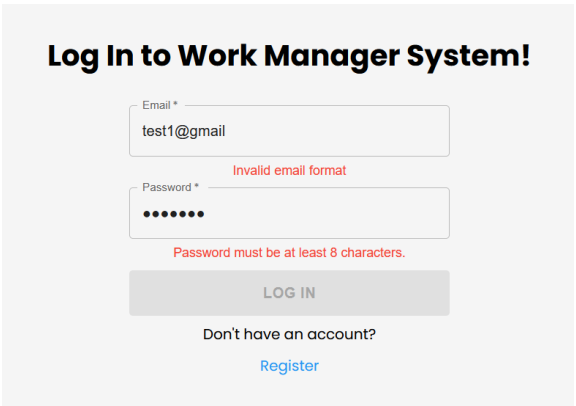
Nakon što korisnik klikne na gumb *Log In*, odnosno gumb za prijavu, na navigacijskoj traci, sustav ga preusmjerava na stranicu za prijavu. Ova stranica prikazuje dvije komponente, komponenta za unos teksta (eng. TextField) i dva gumba: jedan za prijavu i drugi za pristup registraciji, ukoliko korisnik nije registriran.

Prvo polje za unos teksta služi za unos korisničkog računa u formi e-mail adrese, a drugo polje služi za unos lozinke. Prilikom unosa teksta, sustav provjerava ispravnost e-mail adrese, odnosno zadovoljava li pravilan format. Email adresa mora imati tri glavna dijela: lokalni dio koji se nalazi prije znaka "@", domena koja slijedi nakon "@", te završni dio koji sadrži točku i odgovarajuću domenu najviše razine. Na primjer, ispravna email adresa bi bila "test1@gmail.com", gdje "test1" predstavlja lokalni dio, "gmail" predstavlja domenu, a "com" je domena najviše razine. Ukoliko

e-mail adresa ne zadovoljava uvjete, ispod polja za unos teksta se crvenom bojom ispisuje poruka "Invalid email format". Isto tako, provjerava se unesena lozinka koja mora sadržavati najmanje osam znakova, od kojih bar jedan mora biti slovo, a jedan znamenka ili specijalni znak. Ovaj sigurnosni standard poznat je kao politika složene lozinke i definirana je u standardu NIST SP 800-63B (National Institute of Standards and Technology Special Publication 800-63B) [16]. Ako lozinka ne zadovoljava uvjete, ispisuje se odgovarajuća poruka. Na slici 5.2 prikazan je primjer kako sustav reagira kada e-mail adresa ili lozinka ne zadovoljavaju navedene uvjete.

Kada su podaci pravilno uneseni, gumb za prijavu mijenja boju iz sive u plavu, signalizirajući korisniku da je prijava moguća. Kada korisnik klikne na gumb za prijavu, poziva se metoda koja uspostavlja vezu s bazom podataka i provjerava postoji li korisnik u sustavu. Ako korisnik ne postoji, crvenim podebljanim slovima ispisuje se poruka da korisnik s unesenom adresom ne postoji u sustavu. Ako je korisnik pronađen, provjerava se ispravnost lozinke. Ako je lozinka neispravna, ispisuje se greška o neispravnosti lozinke.

Tijekom procesa provjere unesenih podataka, u sredini ekrana pojavljuje se komponenta učitavača (eng. loader), koja pokazuje da je provjera podataka u tijeku.



Slika 5.2 Prikaz nepravilno unesenih podataka za prijavu

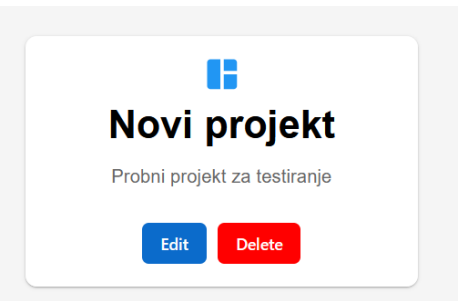
Ako korisnik nije prijavljen u sustav, pristupa stranici za registraciju. Stranica funkcionira slično kao stranica za prijavu, ali se registracija sastoji od četiri polja za unos teksta koje korisnik ispunjava redom: ime, prezime, korisnički račun i lozinka.

Kada korisnik unese sve potrebne podatke, gumb za registraciju poplavi, signalizirajući da je registracija moguća.

Nakon što se registracija ili prijava uspješno izvrši, korisnik se preusmjerava na početnu stranicu, gdje se prikazuje pozdravna poruka i obavijest da je prijava bila uspiješna. Također, kada je korisnik prijavljen, navigacijska traka mijenja svoje elemente. Dodaje se poveznica za stranicu prostora i avatar koji sadrži prvo slovo imena korisnika. Budući da je korisnik već prijavljen, poveznice za prijavu i registraciju više nisu potrebne, te se ne prikazuju. Klikom na avatar otvara se padajući izbornik koji omogućuje pristup upravljanju profilom i odjavi iz sustava.

5.1.3 Prostori

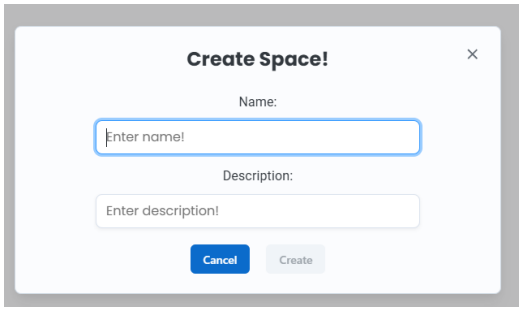
Prijavljeni korisnik pritiskom na poveznicu prostor u navigacijskoj traci pristupa stranici na kojoj se prikazuju svi prostori koje je on kreirao ili u koje je dodan od strane drugih korisnika. Prostori su predstavljeni kao kartice, prikazane na slici 5.3. Kartica se sastoji od naslova prostora, opisa prostora i dva gumba. Prvi gumb služi za uređivanje naslova i opisa, a drugi za brisanje prostora. Ukoliko se korisnik odluči izbrisati prostor, pojavljuje se prozor u kojem korisnik treba potvrditi želi li zaista izbrisati prostor. Klikom na karticu pristupamo odabranom prostoru.



Slika 5.3 Prikaz prostora

U gornjem desnom kutu stranice nalazi se gumb za kreiranje prostora. Kada se klikne na gumb za stvaranje novih prostora, pojavljuje se prozor koji sadrži dva tekstualna polja. Prvo polje služi za upisivanje naziva, a drugo za upisivanje kratkog opisa. Ispod tekstualnih polja nalaze se dva gumba: prvi služi za potvrdu kreiranja,

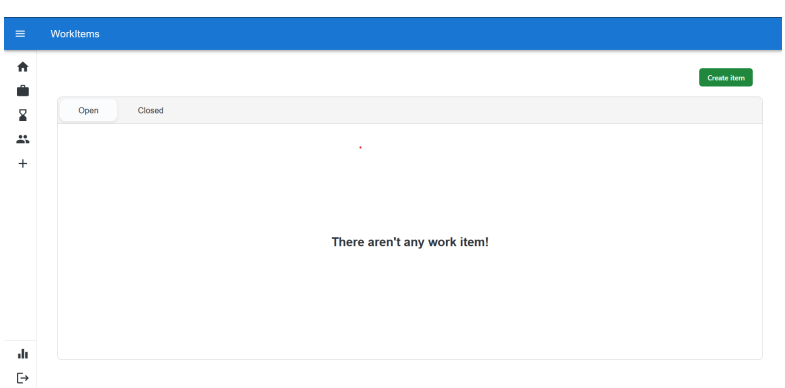
a drugi za odustajanje od kreiranja novog prostora. Također, korisnik može odustati klikom na gumb "X" u gornjem desnom kutu prozora ili jednostavno klikom izvan okvira prozora. Vizualni prikaz prozora za stvaranje novog prostora prikazan je na slici 5.4.



Slika 5.4 Prozor za stvaranje novog prostora

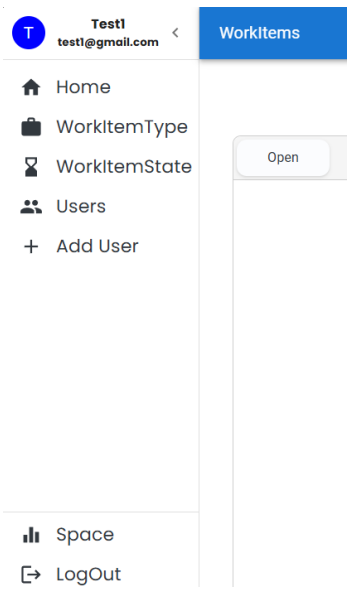
5.1.4 Stranica zadataka

Kada korisnik pristupi odabranom prostora, otvori se nova stranica prikazana na slici 5.5, koja se značajno razlikuje od početne. U središtu stranice nalazi se tablica s karticama zadataka tog prostora, podijeljena na otvorene i zatvorene zadatke. Na lijevoj strani nalazi se bočna traka za navigaciju, koja omogućava pristup ostalim stranicama. Na kraju, u gornjem desnom kutu nalazi se gumb za dodavanje novih zadataka.



Slika 5.5 Prikaz stranice zadataka

Klikom na bočnu navigacijsku traku, traka se proširuje, prikazano na slici 5.6. Traku možemo podjeliti na tri dijela. Prvi dio prikazuje osnovne informacije o korisniku, kao što su ime i prezime te korisnička adresa. Također je prikazan i avatar, na koji korisnik može kliknuti kako bi pristupio stranici za upravljanje korisničkim računom.

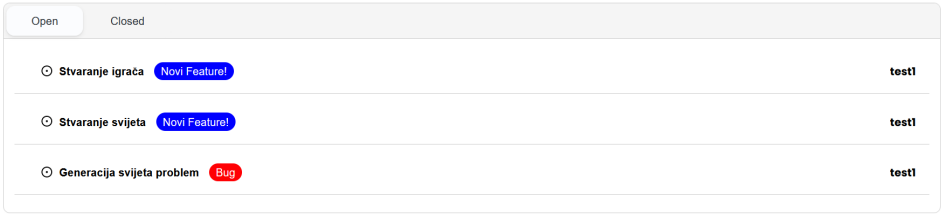


Slika 5.6 Bočna navigacijska traka

Drugi dio navigacijske trake sadrži poredane ikone s pripadajućim tekstom za pristup odgovarajućim stranicama web aplikacije. Ovaj dio uključuje početnu stranicu, stranicu za pristup tipovima i stanjima zadataka, prikaz korisnika koji se nalaze u prostoru i opciju za dodavanje novih korisnika u prostor.

Zadnji dio, smješten na samom dnu trake, služi za pristup stranici za prikaz svih prostora i za odjavu s korisničkog računa.

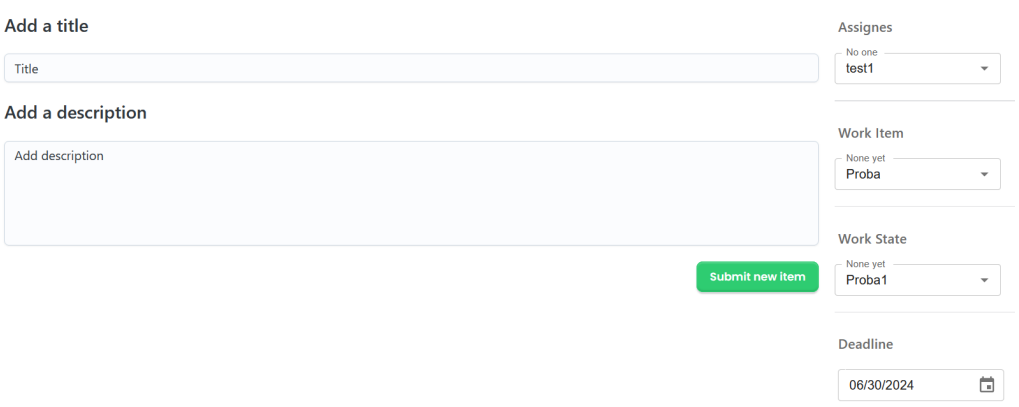
Tablica zadataka ima dva različita načina prikaza: zadatke u tijeku i završene zadatke. Način prikaza može se odabrati pomoću gumba koji se nalaze u zaglavlju tablice. Tablica prikazuje sve zadatke koji se nalaze u prostoru. Za svaki zadatak, tablica prikazuje osnovne podatke kao što su ime zadatka, tip zadatka i odgovornu osobu za izvršavanje tog zadatka. Prikaz tablice zadataka vidljiv je na slici 5.7.



Slika 5.7 Tablica zadataka

5.1.5 Stranica za stvaranje zadataka

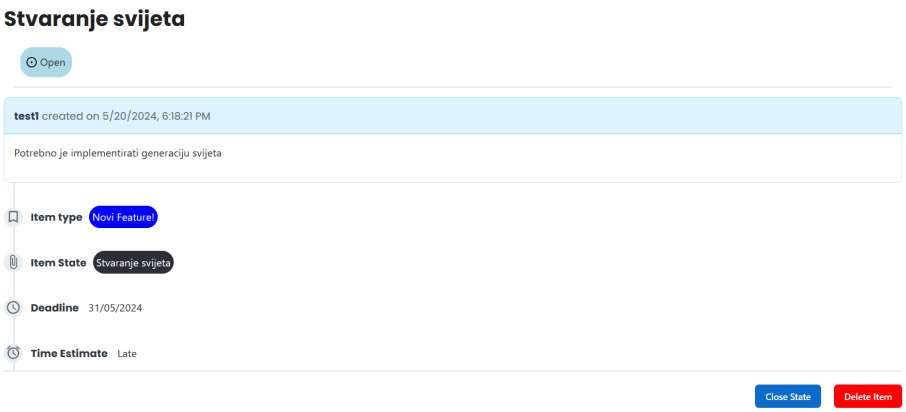
Pritiskom na gumb za stvaranje novih zadataka otvara se nova stranica prikazana na slici 5.8 koja sadrži komponente polja za unos teksta i automatsko dovršavanje (eng. Autocomplete). Komponenta automatskog dovršavanja je vrlo slična polju za unos teksta, u nju možemo upisivati vrijednosti, a ona će na temelju upisane vrijednosti prikazati ponuđene opcije. Za stvaranje novog zadatka potrebno je unijeti ime zadatka, opis zadatka, odgovornu osobu, tip i status zadatka te krajnji rok izvršenja. Kada se svi podaci popune, gumb postaje zelen, signalizirajući da je stvaranje novog zadatka moguće. Pritiskom na gumb za stvaranje, korisnik se preusmjerava na stranicu zadataka.



Slika 5.8 Stvaranje zadataka

5.1.6 Detalji zadataka

Kako bi se prikazali detaljni podaci o zadatku, korisnik mora pritisnuti odabrani zadatak koji se nalazi u tablici. Nakon pritiska na zadatak, otvara se nova stranica predočena na slici 5.9. Na vrhu stranice nalazi se ime zadatka, a ispod imena prikazuje se je li zadatak u tijeku ili je završen. U okviru prikazuju se podaci o tome tko je i kada kreirao zadatak, te detaljan opis zadatka. Nadalje, ispod okvira prikazuju se tip i stanje zadatka, krajnji rok za izvršenje zadatka, te koliko je dana preostalo do isteka roka. Ukoliko je rok istekao, ispisuje se poruka "Late". Također, korisnik može označiti zadatak kao dovršen ili ga, ukoliko više nije potreban, izbrisati.

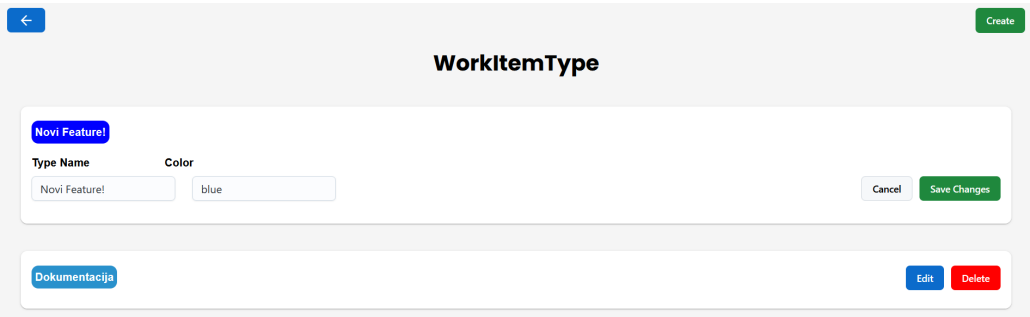


Slika 5.9 Detaljni prikaz zadatka

5.1.7 Tipovi zadataka i stanje zadatka

Prije stvaranja zadataka, korisnik mora definirati tip i stanje zadatka. Pomoću bočne navigacijske trake korisnik može pristupiti stranici za upravljanje tipovima i stanjima zadataka. Tipovi i stanja zadataka prikazani su kao duguljaste kartice poredane jedna ispod druge (Slika 5.10). Svaka kartica sadrži ime i pripadajuću boju, prikazanu s lijeve strane kartice. Na desnoj strani nalaze se dva gumba. Prvi gumb služi za uređivanje, a drugi za brisanje. U desnom gornjem kutu stranice nalazi se gumb za stvaranje novih tipova i stanja, koji otvara prozor kada se pritisne. Prozor sadrži tekstualno polje za unos imena tipa i birač boje, koji omogućava ručni odabir boje

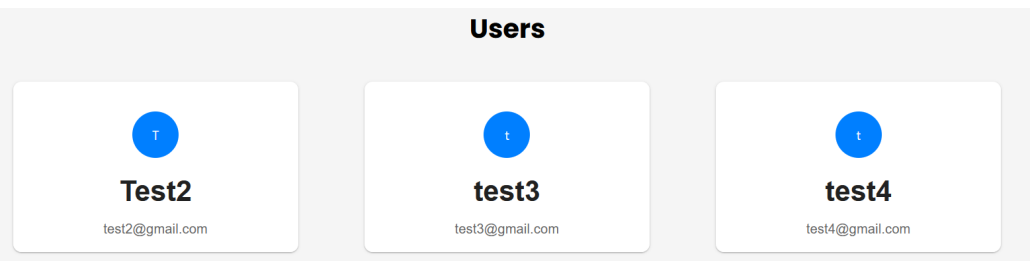
pomoću RGB sustava ili automatski unos željene boje.



Slika 5.10 Tipovi zadatka

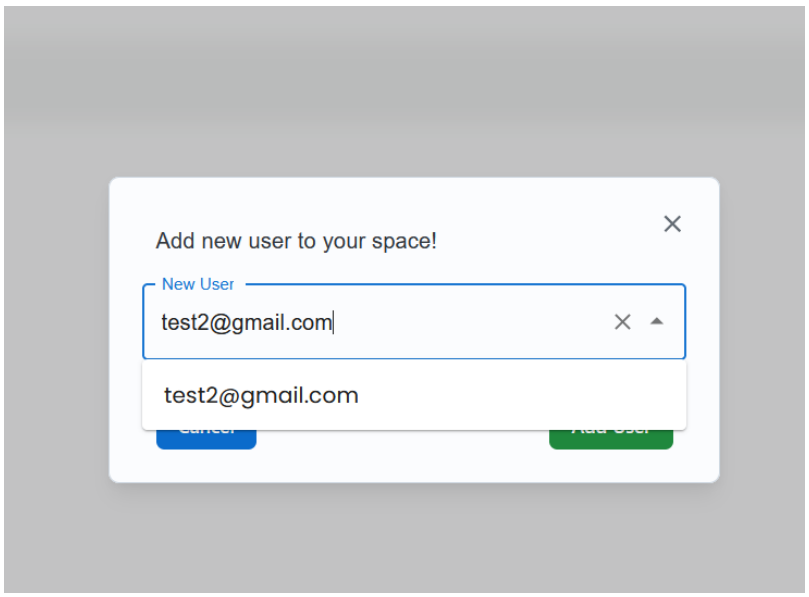
5.1.8 Korisnici prostora

Pomoću stranice korisnika prostora korisnik može upravljati korisnicima unutar prostora. Slika 5.11 prikazuje stranicu koja se sastoji od kartica s avatarima korisnika, imenima i korisničkim adresama. Klikom na karticu otvara se prozor koji prikazuje upozorenje prije brisanja korisnika, gdje korisnik mora potvrditi da želi izbrisati odabranog korisnika iz prostora.



Slika 5.11 Prikaz korisnika

Ukoliko korisnik želi dodati nove korisnike, to može učiniti pomoću bočne navigacijske trake. Klikom na gumb "Add User" otvara se prozor prikazan na slici 5.12, koja sadrži komponentu za automatsko dovršavanje teksta. Kada korisnik pronađe odgovarajuću osobu, može je odabrati i dodati u prostor.

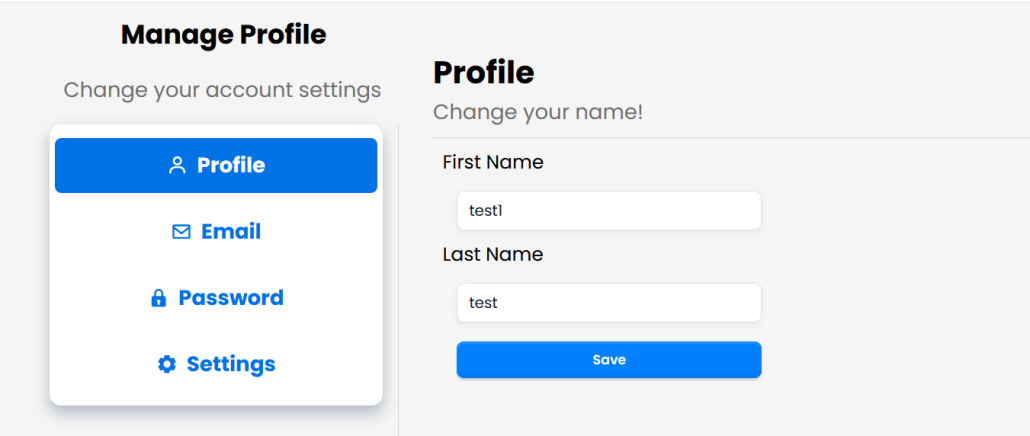


Slika 5.12 Dodavanje novih korisnika

5.1.9 Upravljanje korisničkim računom

Korisnik može izmijeniti svoje podatke u postavkama profila (slika 5.13). Kada korisnik pristupi postavkama, na lijevoj strani prikazuje se navigacija koja omogućava promjenu imena i prezimena, korisničkog računa, lozinke, kao i trajno brisanje računa. U tekstualnim poljima prikazane su trenutne vrijednosti korisnika. Nakon uspješne promjene podataka, prikazuje se kratka poruka o uspjehu. Ukoliko promjena podataka nije bila uspješna, prikazuje se crveni prozor koji signalizira da je došlo do problema.

Ako korisnik želi promijeniti lozinku, prvo je potrebno unijeti trenutnu valjanu lozinku. Za trajno brisanje korisničkog računa, nakon pritiska na gumb za brisanje, otvara se novi prozor u koji je, također, potrebno unijeti lozinku. Tek nakon unosa ispravne lozinke, korisnički račun se trajno briše i korisnik biva preusmjeren na početnu stranicu.



Slika 5.13 Upravljanje korisničkim računom

5.2 Testiranje

Kako bi se provjerila ispravnost sustava, bilo je potrebno provesti testiranja. Pošto su ručna testiranja duga i zamorna, u sustav su, pomoću xUnit.net-a i Moq-a, implementirana automatska testiranja koja provjerava ponaša li se sustav očekivano.

Testiranje je realizirano pomoću testnih klasa. Testnu klasu možemo podijeliti u dva dijela u našem primjeru provjere metode za stvaranje prostora.

Prvi dio koda sastoji se od konstruktora u kojem definiramo varijable za provjeru, inicijaliziramo *InMemory* bazu podataka i konfiguriramo *AutoMapper*. U konstruktoru *SpaceControllerTests*, deklariramo privatne članove klase *ApplicationDbContext*, *IMapper* i *SpaceController*. Inicijalizira se *ApplicationDbContext* s *InMemory* bazom podataka, što omogućava testiranje bez korištenja stvarne baze. Ovo je korisno jer omogućava izolirano i brzo testiranje bez rizika od utjecaja na stvarne podatke. *AutoMapper* konfiguracija je potrebna kako bi se omogućilo automatsko mapiranje između DTO objekata i entiteta baze podataka. Na kraju, se inicijalizira *SpaceController* koristeći kreirane objekte *ApplicationDbContext* i *IMapper*, čime se osigurava da kontroler ima sve potrebne ovisnosti za ispravan rad.

```
public SpaceControllerTests()
{
```

```
        var options = new
            DbContextOptionsBuilder<ApplicationDbContext>()
            .UseInMemoryDatabase(databaseName: "TestDatabase")
            .Options;
        _context = new ApplicationDbContext(options);

        var config = new MapperConfiguration(cfg =>
        {
            cfg.AddProfile<AutoMapperConfigProfile>();
        });

        _mapper = config.CreateMapper();

        _controller = new SpaceController(_context, _mapper);
    }
```

Drugi dio koda sadrži testnu metodu označenu atributom *Fact*, koja definira konkretne korake testiranja. Metoda *CreateSpaceTest* koristi *Arrange-Act-Assert* pristup: prvo se postavljaju ulazni podaci za test (eng. Arrange), zatim se poziva metoda koju testiramo (eng. Act), te se na kraju provjerava rezultat (eng. Assert). U *arrange* dijelu kreira se *SpaceCreateDto* objekt s potrebnim podacima. U *act* dijelu poziva se metoda *CreateSpace* na kontroleru s pripremljenim dto objektom. U *assert* dijelu provjerava se je li rezultat tipa *OkObjectResult*, je li vraćena identifikacijska oznaka ID novog objekta prostora ispravan, te je li novi objekt prostora dodan u bazu podataka s ispravnim vrijednostima (Name, Description, UsersId).

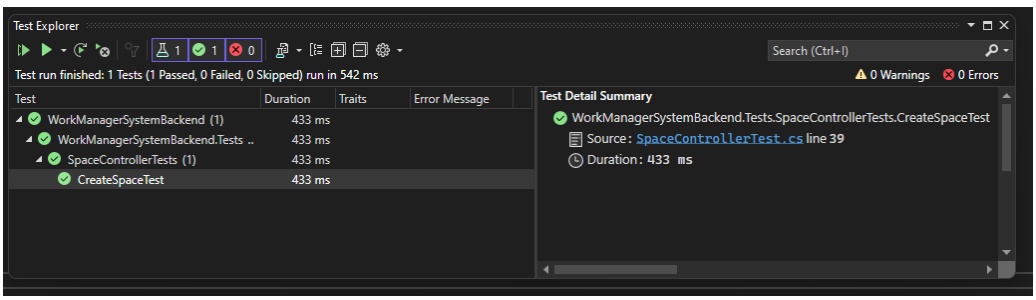
```
[Fact]
public async Task CreateSpaceTest()
{
    // Arrange
    var dto = new SpaceCreateDto
    {
        Name = "New Space",
```

```
            Description = "Description of the new space",
            UsersId = 1
        };

        // Act
        var result = await _controller.CreateSpace(dto);

        // Assert
        var okResult = Assert.IsType<OkObjectResult>(result.Result);
        var newSpaceId = Assert.IsType<long>(okResult.Value);

        var space = await _context.Spaces.FindAsync(newSpaceId);
        Assert.NotNull(space);
        Assert.Equal("New Space", space.Name);
        Assert.Equal("Description of the new space", space.Description);
        Assert.Equal(1, space.UsersId);
}
```



Slika 5.14 Primjer uspješno provedenog testa

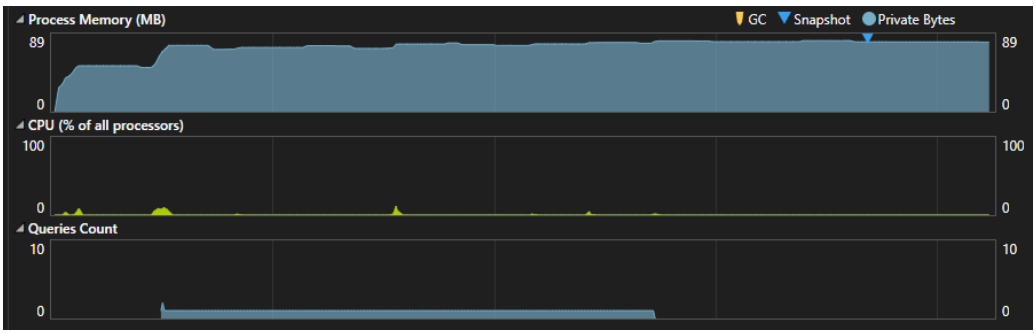
U sklopu testiranja aplikacije, izvršeno je ukupno šest testova. Prvi test je bio usmjeren na stvaranje prostora, dok su ostali testovi obuhvatili dohvaćanje svih prostora, dohvaćanje prostora preko identifikacijske oznake ID, brisanje prostora preko ID-a, prijavu u sustav te promjenu podataka korisnika, kao što je primjerice e-mail adresa. Tokom testiranja, otkrivena je greška kod dohvaćanja prostora gdje se nisu dohvaćali svi potrebni podaci, konkretno identifikacijske oznake korisnika iz prostora.

Ova greška je naknadno ispravljena.

5.3 Performanse

Kako bi se provjerila učinkovitost sustava, bilo je potrebno izvršiti testiranje performansi sustava. Visual Studio prilikom izvršavanja prikazuje osnovne informacije o sustavu pomoću alata *Diagnostic Tools*. *Diagnostic Tools* prikazuju osnovne informacije o korištenju memorije i CPU-a.

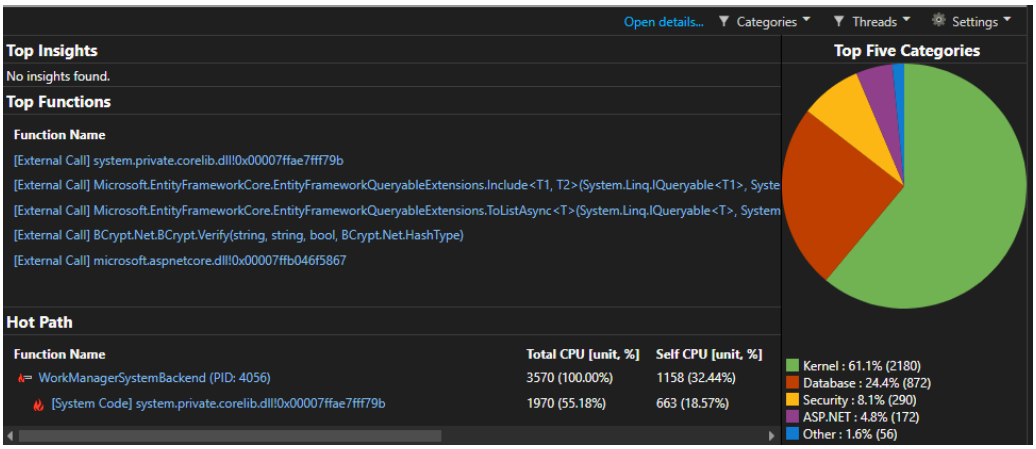
Za detaljniji prikaz performansi sustava, kortisti se funkcionalnost *"Performance Profiler"* koja je ponuđena u postavkama programa. *Performance Profiler* nudi različite alate za mjerenje, kao što su *.NET Async*, *.NET Counters* i drugi. Za potrebe testiranja koristi *CPU Usage*, koji provjerava gdje procesor troši najviše vremena tijekom izvršavanja, *Memory Usage*, koji provjerava koliko se memorije alocira, te *Database*, koji mjeri koliko je puta pozvana SQL naredba i koliko se vremena izvršavala. Nakon izvršavanja profiliranja koda, pomoću dijagrama se prikazuju informacije o sustavu.



Slika 5.15 Dijagram prikaza upotrebe CPU-a, memorije i baze podataka

U ovom primjeru, kao što se vidi na slici 5.15, u početku je relativno niska upotreba memorije i procesora, ali kako raste broj izvršenih SQL naredbi, tako se povećava i korištenje memorije i procesora. Također, ispod dijagrama, prikazanog na slici 5.16, nalaze se detaljne informacije o korištenju procesora. Prvi dio prikazuje funkcije koje su najviše opteretile procesor tijekom izvršavanja, dok drugi dio prikazuje funkcije koje su bile najviše korištene u radnom procesu. Uz to, kružni dijagram

ilustrira pet kategorija koje su najviše koristile procesor.



Slika 5.16 Performanse procesora

6 Zaključak

Ovaj završni rad usmjeren je na razvoj web sustava za upravljanje projektima i zadacima, s ciljem poboljšanja organizacije, produktivnosti i suradnje unutar timova. Sustav je razvijen koristeći moderne tehnologije, uključujući C#, ASP.NET Core, React i TypeScript, čime se osigurava visoka razina funkcionalnosti i prilagodljivosti.

Kroz proces implementacije, detaljno su opisane sve ključne funkcionalnosti sustava, odnosno početna stranica, prijava i registracija korisnika, upravljanje prostorima, zadacima, korisnicima te upravljanje korisničkim računima. Sustav omogućava korisnicima jednostavno kreiranje, praćenje i upravljanje zadacima, čime se poboljšava organizacija i transparentnost rada unutar timova. Korisničko sučelje je intuitivno i responzivno, prilagođeno različitim veličinama ekrana, što omogućava pristup funkcionalnostima s bilo kojeg uređaja.

U fazi implementacije web sustava korišteni su alati Visual Studio za razvoj ASP.NET aplikacije, Visual Studio Code za razvoj *Frontend-a* koristeći React, te Microsoft SQL Server Management Studio za upravljanje bazom podataka.

ASP.NET je okvir koji pojednostavljuje izradu web stranica, ističući se među ostalima svojom brzinom i boljim performansama, omogućavajući brži rad i uštedu resursa. Međutim, jedan od nedostataka ovog okvira je ograničena dokumentacija i nedostatak informacija, što može predstavljati izazov za programere koji ga tek počinju koristiti.

Nadalje, baza podataka je dizajnirana i implementirana korištenjem Entity Framework Core, što omogućava jednostavno definiranje entiteta i njihovih veza, uz automatsko generiranje i upravljanje migracijama. Kontroleri su razvijeni za upravljanje podacima unutar sustava, obuhvaćajući operacije poput dodavanja, dohvaća-

nja, brisanja i ažuriranja podataka, uz podršku za autentifikaciju korisnika putem JWT tokena. Važno je za napomenuti da Entity Framework Core ima i nekoliko nedostataka, primjerice pad performansi kod složenih upita i velikih baza podataka.

React.js i TypeScript koriste se za izradu dinamičkih i responzivnih korisničkih sučelja. React.js omogućava učinkovito upravljanje stanjem aplikacije i ponovnu upotrebu komponenti, dok TypeScript dodaje statičku tipizaciju JavaScript jeziku, pomažući u ranom otkrivanju grešaka olakšavajući održavanje koda. Međutim, React.js može biti kompleksan za učenje, posebno za one koji se prvi put susreću s njim. Koncepti poput virtualnog DOM-a, upravljanja stanjima i životnim ciklusima komponenti mogu biti teški za razumijevanje i pravilnu primjenu bez prethodnog iskustva. Ova kompleksnost može usporiti početni proces učenja i prilagodbe, što može predstavljati prepreku za novije programere ili timove koji nemaju prethodno iskustvo s ovom tehnologijom.

Rezultati testiranja i analize performansi pokazuju da je sustav stabilan, siguran i učinkovit, čime se ispunjavaju svi zadani ciljevi te se korisnicima pruža kvalitetno rješenje za vođenje projekata i upravljanje zadacima.

Kroz ovaj rad uspješno je demonstrirana upotreba tehnologija u razvoju web aplikacija kroz razvoj "Work Management System" kao studije slučaja. Ovaj sustav, iako razvijen kao pokazni primjer, omogućuje uvid u praktičnu primjenu različitih tehnologija i alata u stvarnom okruženju. Cilj rada bio je usporedba različitih tehnologija i alata kako bi se analizirale njihove prednosti i nedostaci u kontekstu razvoja web aplikacija, pružajući tako temelje za odabir najprikladnijih rješenja za specifične potrebe korisnika.

Bibliografija

[1] ASP.NET Core. , s Interneta, https://dotnet.microsoft.com/en-us/apps/aspnet , svibanj 2019 .

[2] What is React.js? Uses, Examples, & More. , s Interneta, https://blog.hubspot.com/website/react-js , studeni 2023.

[3] What Is TypeScript? , s Interneta, https://thenewstack.io/what-is-typescript/ , srpanj 2022.

[4] Anders Hejlsberg, Mads Torgersen, Scott Wiltamuth, Peter Golde, *The C# Programming Language*. Addison-Wesley Professional, listopad 2008.

[5] Joseph Hocking, *Unity in Action, Third Edition*. Manning, veljača 2022. .

[6] What Is C# used for? , s Interneta, https://zerotomastery.io/blog/what-is-c-sharp-used-for/ , studeni 2023.

[7] Introduction to .NET Framework. , s Interneta, https://www.javatpoint.com/vb-net-dot-net-framework-introduction , svibanj 2020 .

[8] Entity Framework? , s Interneta, https://learn.microsoft.com/en-us/aspnet/entity-framework , srpanj 2022.

[9] What is AutoMapper? , s Interneta, https://automapper.org/

[10] How to Secure Passwords with BCrypt.NET. , s Interneta, https://code-maze.com/dotnet-secure-passwords-bcrypt/ , siječanj 2024.

[11] Material UI - Overview. , s Interneta, https://mui.com/material-ui/getting-started/

[12] Visual Studio Code Docs. , s Interneta, https://code.visualstudio.com/docs/?dv=win&build=insiders , lipanj 2024.

[13] What is SQL Server Management Studio (SSMS)? , s Interneta, https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16 , ožujak 2023.

[14] Scott Chacon, Ben Straub, *Pro Git*. Apress, studeni 2014.

[15] What is Swagger? , s Interneta, https://swagger.io/docs/specification/2-0/what-is-swagger/

[16] Paul A. Grassi, James L. Fenton, Elaine M. Newton, Ray A. Perlner, Andrew Regenscheid, *Digital Identity Guidelines, Authentication and Lifecycle Management*. U.S. Department of Commerce, National Institute of Standards and Technology, 2017. .

Sažetak

Ovaj završni rad bavi se razvojem web sustava za vođenje projekata i upravljanje zadacima, s ciljem poboljšanja organizacije, produktivnosti i suradnje u timovima. Sustav je razvijen koristeći moderne tehnologije kao što su C#, ASP.NET Core, React i TypeScript, te pruža korisnicima intuitivno i responzivno korisničko sučelje. Opisane su ključne funkcionalnosti sustava, uključujući prijavu i registraciju korisnika, upravljanje zadacima, prostorima i korisnicima, te sigurnosne mjere poput uporabe Entity Frameworka za upravljanje bazom podataka. Testiranja su pokazala da je sustav stabilan, siguran i učinkovit, te zadovoljava sve zadane ciljeve.

Ključne riječi **— upravljanje zadacima, web sustav, ASP.NET Core, React**

Abstract

This study is focused on the development of a web system for project management and task tracking, with the aim of improving organization, productivity, and collaboration within teams. The system is built using modern technologies such as C#, ASP.NET Core, React, and TypeScript, providing users with an intuitive and responsive user interface. Key functionalities of the system include user registration and login, task, space, and user management, as well as security measures like using Entity Framework for database management. Testing has demonstrated that the system is stable, secure, and efficient, successfully meeting all established goals.

Keywords **— task tracking, web system, ASP.NET Core, React**