Razvoj web aplikacije za upravljanje procesom prijave grešaka

Gazić, Dino

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Dino Gazić

RAZVOJ WEB APLIKACIJE ZA UPRAVLJANJE PROCESOM PRIJAVE GREŠAKA

Diplomski rad

Pula, rujan 2023.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Dino Gazić

RAZVOJ WEB APLIKACIJE ZA UPRAVLJANJE PROCESOM PRIJAVE GREŠAKA

Diplomski rad

JMBAG: 0303076099 6, redovni student

Studijski smjer: Informatika

Predmet: Napredne strukture i algoritmi

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv.prof.dr.sc. Tihomir Orehovački

Pula, rujan 2023.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Dino Gazić, kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student
Dino Gazić

U Puli, rujan 2023. godine

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, Dino Gazić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom **Razvoj web aplikacije za upravljanje procesom prijave grešaka**

koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, rujan 2023. godine

Potpis

Dino Gazić

IN2 društvo s ograničenom odgovornošću
za informatički inženjering i usluge

Marohnićeva 1/1
10000 Zagreb, HR

T +385 1 6386 800
F +385 1 6386 801

OIB 68195665956
MBS 080071082

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli
Rovinjska 14,
52100 Pula

Zagreb, 14.04.2023

PREDMET: Suglasnost za korištenje podataka u svrhu pisanja diplomskog rada studenta Dine Gazića

Poštovani,

Temeljem zaprimljene zamolbe, koju nam je uputio student Dino Gazić, o korištenju podataka i resursa poduzeća IN2 d.o.o. u svrhu pisanja i izrade diplomskog rada pod nazivom "Razvoj web aplikacije za upravljanje procesom prijave grešaka" (u daljnjem tekstu Diplomski rad), ovim putem potvrđujemo i dajemo našu suglasnost.

Suglasni smo da student Dino Gazić koristi sve potrebne podatke i resurse našeg poduzeća u svrhu pisanja i izrade Diplomskog rada isključivo uz poštivanje sljedećih uvjeta:

1. Svi podaci i informacije koje mu IN2 pruži za potrebe Diplomskog rada tretiraju se kao vrlo povjerljivi. Student Dino Gazić će osigurati i odgovoran je da se takvi podaci neće dijeliti ili otkrivati trećim stranama bez izričitog pisanog odobrenja poduzeća IN2.
2. Podaci tvrtke IN2 koje student Dino Gazić koristi u Diplomskom radu bit će korišteni isključivo u svrhu izrade diplomskog rada na temu "Razvoj web aplikacije za upravljanje procesom prijave grešaka".
3. IN2 ni u kojem slučaju nije odgovoran i odriče se svake odgovornosti za bilo kakvu interpretaciju podataka koje je student Dino Gazić napravio u sklopu Diplomskog rada. Student je isključivo odgovoran za sve aspekte rada, uključujući zakonitost i valjanost korištenih podataka.

Ova suglasnost se izdaje i vrijedi isključivo tijekom pisanja i izrade Diplomskog rada na temu "Razvoj web aplikacije za upravljanje procesom prijave grešaka" i danom obrane prestaje važiti, a svaka druga uporaba podataka poduzeća IN2 d.o.o. za bilo koji drugi projekt ili svrhu zahtijeva odvojeni pristanak poduzeća IN2 d.o.o. Zagreb.

Srdačan pozdrav,

Danijel Budaković

Izvršni direktor, Sektor zdravstva i javne uprave
Voditelj IN2 ureda Pula

Sadržaj

1. UVOD

U današnjem digitalnom dobu, organizacije sve više prepoznaju važnost efikasnog upravljanja procesima i resursima kako bi osigurale glatko funkcioniranje svojih aktivnosti. U tom kontekstu, web aplikacije su postale ključni alat za olakšavanje različitih poslovnih funkcija, uključujući i procese upravljanja i praćenja grešaka ili problema unutar organizacije. Ova tema je od suštinskog značaja za organizacije koje žele poboljšati svoj interni i eksterni tok komunikacije, smanjiti vrijeme rješavanja problema i optimizirati resurse.

U današnjem globalnom poslovnom okruženju, gdje su brze promjene i dinamičnost ključni faktori uspjeha, sposobnost brze reakcije na tehničke greške ili nedostatke predstavlja kritičnu komponentu za očuvanje zadovoljstva korisnika, očuvanje ugleda kompanije te optimizaciju ukupnih poslovnih performansi. Stoga, kako bi postigle viši nivo konkurentske prednosti i ostvarile održivi rast, organizacije se sve više okreću prema inovativnim tehnološkim rješenjima. Ovaj diplomski rad temelji se na istraživanju i izradi praktičnog dijela upravo ove bitne problematike koja oblikuje današnje poslovno okruženje.

Ovaj diplomski rad ima za cilj duboko istražiti ključne aspekte razvoja web aplikacija usmjerenih na upravljanje procesom prijave grešaka. Fokusirajući se na metode analize, dizajna i implementacije, detaljno ćemo razmotriti kako integracija tehnoloških resursa može efikasno podržati ovu ključnu funkciju unutar organizacije. Pored toga, temeljito ćemo istražiti i analizirati aspekte sigurnosti podataka i korisničkog sučelja kako bismo osigurali visoku razinu pouzdanosti i jednostavnu upotrebljivost razvijene web aplikacije. Ovaj rad će također obratiti pažnju na praktične primjene i praktične izazove koji se javljaju u procesu razvoja ovakvih aplikacija kako bi pružio sveobuhvatno razumijevanje teme.

U nastavku rada, detaljno ćemo analizirati koncepte upravljanja greškama, istražiti potrebe i zahtjeve korisnika prema aplikaciji te pregledati relevantne tehnološke alate i resurse ključne za razvoj ovakve web aplikacije. Kroz kombinaciju teorijskih saznanja i praktične primjene, ovaj diplomski rad ima za cilj pružiti dublje razumijevanje procesa upravljanja prijavama grešaka putem web aplikacije. Time će doprinijeti širem shvaćanju važnosti tehnološki podržanog upravljanja greškama u suvremenim poslovnim okruženjima.

2. TEHNOLOGIJE I ALATI KORIŠTENE PRI IZRADI APLIKACIJE

Za ovu web aplikaciju tehnologije koje su korištene su: Flutter/Dart, .NET Core 6.0 te SQL server. Flutter programski okvir napisan u programskom jeziku Dart je zaslužan za korisničko sučelje web aplikacije, .NET Core 6.0 tehnologija je bila zadužena za web API, a za bazu podataka korištena je tehnologija SQL Server. Aplikacija je u komunikaciji sa ticketing sustavom od kompanije gdje se vode zabilješke i kontrolira se status zadatka. Ticketing sustav koji firma ima je MantisBT.

2.1. AKTUALNE FLUTTER APLIKACIJE

Ovaj inovativni okvir za razvoj koristeći jedan kod za više platformi privukao je pažnju i omogućio stvaranje aplikacija visoke kvalitete. Neki od tih istaknutih primjera jesu [1]:

1. **Google Ads**: Flutter je poslužio kao alat moći za Google, omogućujući im da unaprijede segment svoje Google Ads aplikacije. Tako se jasno potvrđuje izvrsnost i pouzdanost Fluttera, s obzirom na to da se Google oslanja na njega čak i za interne potrebe.
2. **Alibaba**: Kineski e-trgovinski div, Alibaba, koristio je Flutter kako bi podigao razinu jednog dijela svoje aplikacije. Ova odluka ističe se kao izvanredna, s obzirom na veličinu i složenost aplikacija koje se povezuju s Alibabom.
3. **Reflectly**: Putem privlačnog korisničkog sučelja i besprijekornog iskustva, aplikacija Reflectly, koja služi kao osobni dnevnik i mjesto refleksije, postigla je značajnu popularnost - a sve to zahvaljujući Flutteru.
4. **Hamilton**: Službena aplikacija za poznati mjuzikl "Hamilton" omogućila je obožavateljima da zaroni u raznolike sadržaje i informacije vezane uz ovaj mjuzikl, pružajući im jedinstveno iskustvo koje im povezuje s umjetničkim djelom.
5. **Nubank**: Fintech tvrtka Nubank iz Brazila prepoznala je prednosti Fluttera i iskoristila ga za stvaranje ključnih dijelova svoje mobilne aplikacije. To je snažan dokaz njegove funkcionalnosti i sposobnosti, čak i u zahtjevnim sektorima.

Ovaj navedeni popis vjerojatno predstavlja tek vrh ledene sante kada je riječ o popularnim aplikacijama razvijenim u Flutteru. S razvojem tehnologije i sveobuhvatnosti ekosustava Fluttera, možemo očekivati da će se pojaviti još značajnih primjera koji će dodatno osvijetliti široki potencijal ovog okvira.

2.2. USPOREDBA S POSTOJEĆIM TEHNOLOGIJAMA

Programski okvir Flutter, ističući svojom sposobnošću za stvaranje atraktivnih, brzih i istovremenih aplikacija za više platformi, predstavlja rješenje za razvoj korisničkog sučelja. Međutim, važno je istaknuti kako postoje i drugi programski okviri koji se koriste u svrhu razvoja korisničkog sučelja, a koji konkuriraju Flutteru.

U nastavku su navedeni primjeri nekoliko takvih okvira [2]:

1. **React Native:** Ovaj okvir, razvijen od strane Facebooka, omogućava razvoj mobilnih aplikacija za platforme iOS i Android putem korištenja jezika JavaScript i koncepta React. React Native primjenjuje prilagodljive komponente te pruža opciju "Hot Reload" kako bi se postigla brza iteracija razvoja. Iako se oslanja na različite jezike i pristupe u odnosu na Flutter, također se izdvaja kao popularan izbor za razvoj mobilnih aplikacija.
2. **Xamarin:** Microsoftov okvir Xamarin omogućava razvoj mobilnih aplikacija za platforme iOS i Android putem programskog jezika C# i .NET platforme. Njegova važnost leži u mogućnosti integracije s Microsoftovim ekosustavom te u pružanju rješenja za aplikacije koje zahtijevaju takvu integraciju.
3. **Ionic:** Okvir Ionic omogućava razvoj hibridnih mobilnih aplikacija pomoću web tehnologija kao što su HTML, CSS i JavaScript. Aplikacije izgrađene putem Ionic okvira često se izvode unutar "Webview"-a u web preglednicima mobilnih uređaja.
4. **NativeScript:** NativeScript je okvir koji omogućava razvoj mobilnih aplikacija za platforme iOS i Android kroz upotrebu programskih jezika JavaScript, TypeScript ili Angular. Aplikacije stvorene putem NativeScript okvira koriste autentične komponente platformi putem bridževa.

5. **SwiftUI:** Okvir SwiftUI, razvijen od strane Applea, namijenjen je izradi aplikacija za iOS, macOS, watchOS i tvOS putem programskog jezika Swift. Njegova značajnost leži u pristupu deklarativnom razvoju korisničkog sučelja.

Svaki od ovih programskih okvira posjeduje svoje prednosti i ograničenja koja je važno razmotriti pri odabiru najprikladnijeg za konkretni projekt ili razvojni tim. Odabir između ovih okvira ovisi o širokom spektru faktora, uključujući tehničku infrastrukturu, vještine tima, specifične zahtjeve aplikacije i postavljene ciljeve projekta.

2.3. TEHNOLOGIJE KORIŠTENE ZA IZRADU APLIKACIJE

Tehnologije korištene za izradu ove web aplikacije jesu: programski okvir *Flutter*, *.NET Core 6.0* kao *web API* te *SQL Server* alat od kompanije Microsoft.

2.3.1 FLUTTER/DART

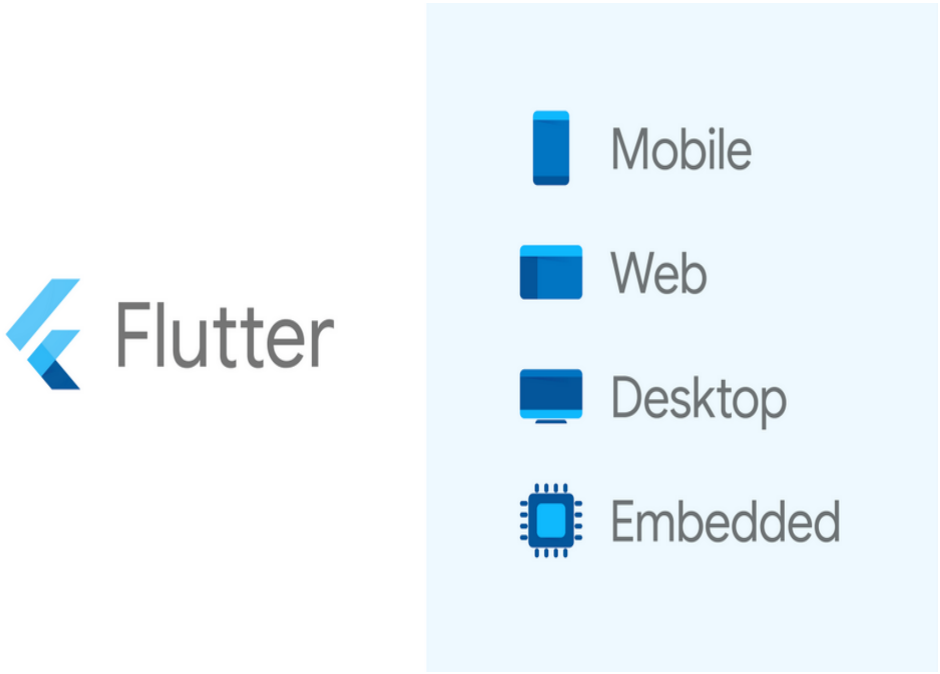
Otvoreni okvir (eng. framework) za razvoj korisničkog sučelja (eng. UI) koji je razvila tvrtka Google je Flutter. Flutter omogućava izradu visokokvalitetnih, brzih i atraktivnih mobilnih, web i desktop aplikacija. Ključna karakteristika Fluttera je njegova upotreba programskog jezika Dart za razvoj aplikacija. Moderan programski jezik koji je također razvijen od strane Googlea je Dart. On kombinira sintaksu sličnu drugim popularnim jezicima kao što su JavaScript, Java i C#, te dodatne značajke koje ga čine pogodnim za razvoj aplikacija koje imaju visoke performanse [3].

Evo nekoliko značajki koje Dart nudi [3]:

1. Brzina izvođenja: Dart koristi optimizirani JIT (eng. Just-In-Time) i AOT (eng. Ahead-of- Time) kompajler za brzo izvođenje koda.
2. Sučelje: Dart je dizajniran kako bi olakšao izradu bogatih korisničkih sučelja, što ga čini prikladnim za razvoj aplikacija kao što su Flutter aplikacije.
3. Objektno orijentirani jezik: Dart je objektno orijentiran jezik, gdje se sve u Dartu smatra objektom, olakšavajući organizaciju koda.
4. Razumljiva sintaksa: Sintaksa Darta je čista i lako čitljiva, slična sintaksi mnogih drugih popularnih programskih jezika.
5. Tipizacija: Dart podržava statičku tipizaciju, što pomaže u otkrivanju grešaka tijekom razvoja.

6. Asinkrono programiranje: Dart podržava asinkrono programiranje putem Future i Stream API-ja, što je ključno za brze i odzivne aplikacije.
7. Razvoj za različite platforme: Dart i Flutter omogućavaju razvoj aplikacija koje se mogu izvoditi na različitim platformama (mobilnim uređajima, webu i desktopu) koristeći isti kod, olakšavajući održavanje i širenje aplikacija.

Zahvaljujući kombinaciji Flutter okvira i Dart programskog jezika, programeri mogu brzo razvijati visokokvalitetne aplikacije s bogatim korisničkim sučeljem koje su brze, izdržljive i prilagodljive različitim platformama. Na slici 1. prikazan je programski okvir Flutter koji može biti izvršavan na više platformi.



Slika 1. Flutter cross platoforma [Izvor: https://medium.com/swlh/flutter-2020-state-of-cross-platform-814f1d8ff16]

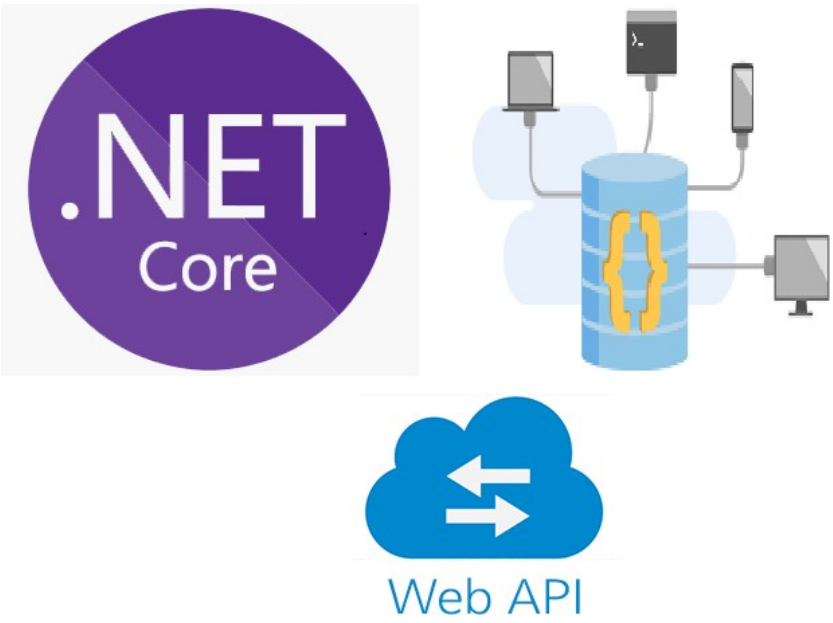
2.3.2. .NET CORE 6.0

.NET Core 6.0 je moderni okvir (eng. framework) za razvoj aplikacija, posebno web aplikacija i servisa. Razvijen od strane Microsofta, .NET Core omogućava programerima izgradnju visokokvalitetnih i skalabilnih aplikacija koje su kompatibilne s različitim platformama. .NET Core 6.0 se često koristi za razvoj web API-ja, a osigurava iznimnu brzinu izvođenja i performanse [4].

Glavne karakteristike .NET Core 6.0 uključuju [4]:

1. **Višestruke platforme:** .NET Core je prilagodljiv za različite platforme kao što su Windows, Linux i macOS. To omogućava razvoj aplikacija koje se mogu izvoditi na različitim operativnim sustavima.
2. **Open Source:** .NET Core je open-source projekt, što znači da je dostupan svima za pregled, doprinos i korištenje. To potiče zajednički razvoj i inovacije.
3. **Brzina i performanse:** .NET Core je optimiziran za brzo izvođenje i visoke performanse. To je posebno važno za web API-je koji moraju odgovarati na zahtjeve velikog broja korisnika.
4. **Skalabilnost:** .NET Core omogućava izgradnju skalabilnih aplikacija koje se mogu nositi s povećanim opterećenjem i rastućim brojem korisnika.
5. **Suvremeni jezik i alati:** .NET Core podržava različite programerske jezike kao što su C#, F# i VB.NET, uz pružanje moćnih alata za razvoj i debagiranje.
6. **Web API razvoj:** .NET Core se često koristi za izgradnju web API-ja. Ovo omogućava kreiranje servisa koji omogućavaju komunikaciju između različitih komponenata aplikacije ili s klijentskim aplikacijama.
7. **Kontejnerizacija:** .NET Core se lako može kontejnerizirati pomoću tehnologija poput Docker-a, što olakšava distribuciju aplikacija i upravljanje okruženjima.

Na slici 2. je prikazan .NET Core gdje se koristi kao web API.



Slika 2. .NET web API [Izvor: https://dev.to/renukapatil/create-web-api-with-aspnet-core-60-46l4]

2.3.3. SQL SERVER (baza podataka)

SQL Server je relacijski upravljač sustava bazama podataka (DBMS) razvijen od strane Microsofta. Ova tehnologija omogućava organizacijama i programerima pohranjivanje, upravljanje i pristupanje podacima na efikasan i pouzdan način. SQL Server je često korišten za izgradnju baza podataka koje podržavaju različite vrste aplikacija, od web aplikacija do poslovnih sustava [5].

Ključne karakteristike SQL Servera uključuju [5]:

1. **Relacijska struktura podataka:** SQL Server koristi relacijski model podataka, gdje su podaci organizirani u tablicama s redovima (zapisisma) i stupcima (atributima). Ova struktura omogućava jednostavno organiziranje i upravljanje podacima.
2. **SQL (Structured Query Language):** SQL Server koristi SQL, standardni jezik za upravljanje i upite baza podataka. SQL omogućava definiranje, manipulaciju i dohvaćanje podataka.
3. **Visoka pouzdanost:** SQL Server je poznat po svojoj pouzdanosti i zaštiti podataka. Pruža mehanizme za osiguravanje integriteta podataka, sigurnosnih prava pristupa i automatskog oporavka od kvarova.

4. **Visoke performanse:** SQL Server je optimiziran za brze upite i transakcije.

Ima napredne algoritme za upravljanje upitima i indeksiranjem podataka kako bi se osigurala visoka performansa.

5. **Skalabilnost:** SQL Server omogućava horizontalno i vertikalno skaliranje, što znači da se baze podataka mogu proširiti kako bi se nosile s rastućim zahtjevima i opterećenjem.
6. **Raznoliki tipovi podataka:** SQL Server podržava različite tipove podataka, uključujući tekstualne, numeričke, vremenske, binarne i mnoge druge.
7. **Upravljanje transakcijama:** SQL Server omogućava izvođenje sigurnih i dosljednih transakcija, što je ključno za održavanje integriteta podataka.
8. **Poslovna inteligencija:** SQL Server uključuje alate za poslovnu inteligenciju poput upita za analizu i generiranja izvještaja, čime omogućava dublje razumijevanje podataka i donošenje informiranih odluka.

SQL Server omogućava organizacijama da učinkovito upravljaju i iskorištavaju svoje podatke, podržavajući širok raspon aplikacija i poslovnih potreba. S njegovim mogućnostima pouzdane pohrane, brzog pristupa i analize podataka, SQL Server igra ključnu ulogu u podržavanju tehnološkog okruženja organizacija diljem svijeta. Na slici 3. je prikazan logo SQL Servera.

Slika 3. SQL Server logo [Izvor: https://www.sqlservertutorial.net]

2.4. OSTALE TEHNOLOGIJE I ALATI

Od ostalih tehnologija i alata korišten je *MantisBT* koji je integriran sa samom aplikacijom i vrši interakciju s njim, *postman* koji se koristio za testiranje, izgradnju i praćenje API poziva, te IIS (eng. *Internet information services*) za postavljanje aplikacije na poslužitelja.

2.4.1. MANTISBT – TICKETING SUSTAV

MantisBT (eng. Mantis Bug Tracker) je open-source sustav za praćenje grešaka (bugova) i upravljanje zadacima (ticketing) koji se koristi za praćenje razvoja softvera, održavanje i podršku [6]. Evo nekoliko ključnih informacija o MantisBT sustavu:

MantisBT je besplatan i open-source softver, što znači da izvorni kôd sustava možete preuzeti, prilagoditi ga svojim potrebama i koristiti ga besplatno. Glavna svrha mu je omogućiti timovima da prate i upravljaju greškama u softveru, kao i drugim zadacima i zahtjevima. Korisnici mogu prijavljivati greške, sugerirati nove značajke ili postavljati općenite zadatke, a zatim pratiti njihov napredak. MantisBT se koristi putem web-baziranog sučelja, što znači da korisnici pristupaju sustavu putem web preglednika bez potrebe za instalacijom dodatnog softvera. Timovi mogu koristiti MantisBT za praćenje cijelog ciklusa razvoja softvera, uključujući praćenje grešaka od trenutka kada su prijavljene do trenutka kada su ispravljene i zatvorene [6].

Svaka prijava greške ili zadatka može biti označena prioritetom i dodatnim atributima, što pomaže timovima da rangiraju zadatke prema važnosti i hitnosti. Korisnici mogu dodavati komentare i priloge uz svaku prijavu, olakšavajući komunikaciju među članovima tima i omogućavajući dodatne informacije o problemima. Također, automatski bilježi promjene koje su se dogodile s određenom prijavom tijekom vremena, omogućujući uvid u to kako se greška razvijala i kako je bila riješena. Sustav podržava višekorisnički pristup s različitim razinama ovlasti. Administratori, razvojni inženjeri, QA timovi i ostali korisnici mogu imati različite razine pristupa i mogućnosti [6].

MantisBT je prilagodljiv, što znači da ga možete prilagoditi svojim potrebama. Možete definirati vlastite atribute, prioritetne razine, tokove rada itd. Često podržava integraciju s drugim alatima i servisima kao što su sustavi za verzioniranje, e-pošta, upravljanje projektima i sl. Popularan alat među razvojnim timovima i organizacijama

koje traže efikasno rješenje za praćenje grešaka i upravljanje zadacima. Osim toga, zajednica korisnika i razvijatelja često doprinosi daljnjem razvoju i poboljšanju sustava [6].

2.4.2. POSTMAN

Postman se ističe kao iznimno cijenjen alat među programerima za potrebe testiranja i razvoja API-ja (eng. Application Programming Interface). Ova platforma omogućuje programerima da na vrlo učinkovit način izvode HTTP zahtjeve prema API-ju, pružajući im kontrolu nad interakcijom s njim. Kroz Postman, programeri su u mogućnosti kreirati, uređivati i izvoditi različite scenarije, simulirajući stvarne uvjete rada API-ja te prateći reakcije i odgovore koje API vraća [7].

Jedna od ključnih prednosti Postmana leži u njegovom intuitivnom i korisnički prijateljskom grafičkom sučelju. Ovo sučelje omogućuje korisnicima da lako definiraju zahtjeve dodajući različite parametre, zaglavlja te tijelo zahtjeva. Osim toga, Postman podržava automatsko generiranje programskog koda za različite programske jezike, što omogućuje programerima da brže i jednostavnije integriraju svoje aplikacije s API-jem [7].

Postman ne nudi samo osnovne funkcionalnosti testiranja, već i niz naprednih opcija koje dodatno obogaćuju njegovu primjenu. Automatizacija testova omogućuje izvođenje ponovljivih testova, čime se osigurava dosljedna kvaliteta API-ja. Organizacija zahtjeva u kolekcije olakšava upravljanje većim brojem zahtjeva, dok mogućnost dijeljenja projekata i suradnje s kolegama omogućuje timsko radno okruženje. Također, Postman omogućuje praćenje povijesti zahtjeva, što pomaže u razumijevanju promjena u API-ju tijekom vremena [7].

U konačnici, sve ove značajke čine *Postman* nezamjenjivim alatom za sve koji se bave razvojem i testiranjem API-ja. Njegova sposobnost olakšavanja procesa testiranja, podrška za različite jezike i mogućnost timskog rada čine ga ključnim alatom za osiguranje stabilnih i visokokvalitetnih API usluga, dok istovremeno povećava produktivnost programera i omogućuje im da brže i efikasnije izvode razvojne zadatke [7].

2.4.3. INTERNET INFORMATION SERVICES (IIS)

Nakon što je aplikacija napravljena, njezino postavljanje na server je odrađeno putem *Microsoft* alata IIS (eng. *Internet information services*). Zapravo, IIS je popularan softver za posluživanje web. Koristi se za hostanje web stranica, web aplikacija i drugih web usluga na Windows operativnim sustavima. IIS pruža infrastrukturu za upravljanje HTTP zahtjevima, obradu ASP.NET i drugih web tehnologija te osigurava siguran i pouzdan način distribucije web sadržaja [8].

3. MOTIVACIJA

Motivacija je pronađena u potrebama kompanije za efikasnim i transparentnim procesom upravlja greškama. Brzo otkrivanje, praćenje i rješavanje problema igra ključnu ulogu u održavanju visokog nivoa zadovoljstva korisnika i reputacije organizacije. Uočeno je dosta nedostataka u postojećem načinu upravljanja greškama koji je često bio spor, neorganiziran i nije pružao dovoljno detaljnih informacija o prijavljenim problemima. Prema gore navedenim nedostatcima stvorila se ideja za stvaranje alata koji će omogućiti precizno praćenje svakog koraka u procesu rješavanja grešaka, time će korisnicima biti pružena bolja vidljivost i bolja mogućnost suradnje. Za izradu web aplikacije će biti potrebno intenzivno učenje, istraživanje i tehničke vještine kako bi se postigao željeni rezultat. No, kada finalni proizvod bude završen, u obliku funkcionalne i korisne web aplikacije, koja ima pozitivan utjecaj na organizaciju, tim i korisnike bit će sigurno odlično riješenje za gore navedeni problem.

U konačnici, ideja o stvaranju alata koji može poboljšati način na koji se suočavamo s problemima i greškama, kako unutar tima tako i s našim korisnicima, predstavlja glavni motivacijski faktor za razvoj ove web aplikacije.

3.1. SWOT ANALIZA

Ova SWOT analiza pruža pregled unutarnjih snaga i slabosti aplikacije te vanjskih prilika i prijetnji s kojima se aplikacija može suočiti. To može pomoći u planiranju strategija za daljnji razvoj, implementaciju i održavanje web aplikacije za upravljanje procesom prijave grešaka.

Snage:

1. **Efikasnost upravljanja:** Aplikacija omogućava brzu i centraliziranu prijavu grešaka, što povećava učinkovitost procesa otkrivanja i rješavanja problema.
2. **Transparentnost:** Korisnici imaju pristup svim fazama procesa rješavanja grešaka, pomažući u transparentnosti.
3. **Smanjenje vremena rješavanja:** Bolja organizacija i praćenje problema omogućava brže reagiranje i rješavanje grešaka, što može povećati zadovoljstvo korisnika.

Slabosti:

1. **Tehnička zahtjevnost:** Implementacija web aplikacije zahtijeva tehničko znanje i resurse za razvoj.
2. **Učenje korisnika:** Korisnicima može biti potrebno vrijeme za prilagodbu novoj platformi, posebno ako su navikli na postojeće metode.

Mogućnosti:

1. **Poboljšana korisnička podrška:** Aplikacija omogućava bolju komunikaciju između korisnika i timova podrške, poboljšavajući iskustvo korisnika.
2. **Skalabilnost:** Ako aplikacija postane uspješna, moguće je proširiti funkcionalnosti i dodati nove značajke kako bi se bolje zadovoljile potrebe korisnika.

Prijetnje:

1. **Konkurencija:** Ako već postoje rješenja za upravljanje greškama, tehnologija mora ponuditi dodatne prednosti kako bi se istakla.
2. **Sigurnost podataka:** S obzirom na osjetljivu prirodu podataka, sigurnost podataka je ključna; eventualni sigurnosni propusti mogu imati ozbiljne posljedice.
3. **Promjena zahtjeva:** Brze promjene u tehnologiji ili poslovnim zahtjevima mogu zahtijevati redovito ažuriranje aplikacije kako bi se ostala relevantna.

U zaključku SWOT analize za web aplikaciju koja se bavi procesom prijave grešaka, ističu se ključni faktori koji će oblikovati njen uspjeh i izazove.

Aplikacija ima neospornih snaga, uključujući efikasnost upravljanja i centraliziranu prijavu grešaka. Ovo omogućava brzu detekciju i rješavanje problema, što povećava produktivnost i korisničko iskustvo. Transparentnost koju pruža, omogućavajući korisnicima uvid u fazama rješavanja grešaka, dodatno podiže nivo povjerenja.

Međutim, postoji nekoliko slabosti koje treba uzeti u obzir. Tehnička zahtjevnost implementacije može zahtijevati značajne resurse, kako financijske tako i ljudske. Također, korisnicima će možda trebati vrijeme da se prilagode novoj platformi, što može usporiti prihvaćanje aplikacije.

Analiza također ukazuje na mnoge mogućnosti koje su pred aplikacijom. Poboljšana korisnička podrška kroz bolju komunikaciju i brže rješavanje problema može povećati zadovoljstvo korisnika. Skalabilnost aplikacije, ukoliko postane uspješna, otvara put ka dodavanju novih značajki i proširenju funkcionalnosti.

No, uz mogućnosti dolaze i prijetnje koje treba pažljivo upravljati. Konkurencija u industriji upravljanja greškama može zahtijevati inovativne prednosti kako bi aplikacija privukla korisnike. Sigurnost podataka je ključna briga, jer osjetljivi podaci mogu biti meta potencijalnih napada. Brze promjene tehnologije ili poslovnih zahtjeva mogu zahtijevati redovita ažuriranja kako bi aplikacija ostala relevantna.

4. RAZRADA FUNKCIONALNOSTI

U poglavlju razrada funkcionalnosti će biti opisana dva dijagrama. Prvi dijagram koji će biti opisan je dijagram slučajeva korištenja, drugi je sekvencijalni dijagram, dok je treći klasni dijagram. U uvodu ovog poglavlja će dijagrami biti opisani.

Dijagram slučajeva korištenja (eng. Use Case Diagram) je vrsta dijagrama u UML notaciji koji se koristi u softverskom inženjeringu i analizi poslovnih procesa. Ovaj dijagram se često koristi za modeliranje funkcionalnosti ili ponašanja sistema iz perspektive korisnika ili aktera sistema [10].

Sekvencijalni dijagram je grafički prikaz procesa ili sustava pomoću različitih oblika i povezanih strelica koje označavaju korake, odluke, akcije i tok informacija unutar tog procesa. Dijagrami toka su alat koji se često koristi za vizualizaciju složenih procesa ili procedura kako bi se olakšalo razumijevanje, analiza, planiranje i komunikacija [11].

Dok klasni dijagram prikazuje grafički prikaz u objektno orijentiranom programiranju koji ilustrira strukturu razreda (klasa) u nekom softverskom sustavu ili aplikaciji. Ovaj dijagram koristi se za vizualizaciju odnosa među razredima, njihovim atributima i metodama te nasljeđivanju između njih [12].

4.1. DIJAGRAM SLUČAJEVA KORIŠTENJA (USE CASE)

Pogledom na dijagram slučajeva korištenja se vidi da aplikacija ima šest osnovnih prikazanih mogućnosti iz kojih proizlaze još neke mogućnosti koje smo primorani koristiti <<*include*>> i neke koje su opcionalne za korištenje <<*extend*>>. Dijagram opisuje isključivo krajnjeg korisnika koji iterira sa aplikacijom.

Prvi korak pri korištenju same aplikacije je korak prijave u aplikaciju, odnosno ukoliko korisnik nije registriran obavezan je napraviti registraciju jer bez iste neće biti u mogućnosti pristupiti samoj aplikaciji, to je ujedno i razlog zašto je registracija u sustav povezana <<*include*>> vezom sa prijavom u sustav.

Kada je korisnik prijavljen unutar aplikacije, ima slijedeće mogućnosti: prijavljivanje zadataka, pregled zadataka, odabir pojedinačnog zadatka, administracija korisnika i dodavanje novog korisnika. Svaka od nabrojanih mogućnosti ima još dodatne opcije korištenja akcije kako je navedeno u odlomku iznad. Dijagram slučajeva korištenja se može vidjeti na slici br. 4.

Slika 4. Prikaz dijagrama slučaja korištenja

4.2. SEKVENCIJALNI DIJAGRAM

Sekvencijalni dijagram sadržava objekt korisnika koji iterira sa aplikacijom, drugi skup objekata je web aplikacija, dok je treći objekt MantisBT koji je izdvojeni dio sustava. U dijagramu toka imamo jednog sudionika, on je korisnik aplikacije. Također postoje tri objekta: jedan od njih je korisničko sučelje aplikacije, drugi je serverska strana aplikacije. Ta dva objekta zajedno čine cjelinu web aplikacije. Treći objekt je vanjski sustav odnosno MantisBT. Za svaki od navedenih objekata imamo liniju života, linija života nam prikazuje koliko se dugo prikazani objekt koristi kroz aplikaciju.

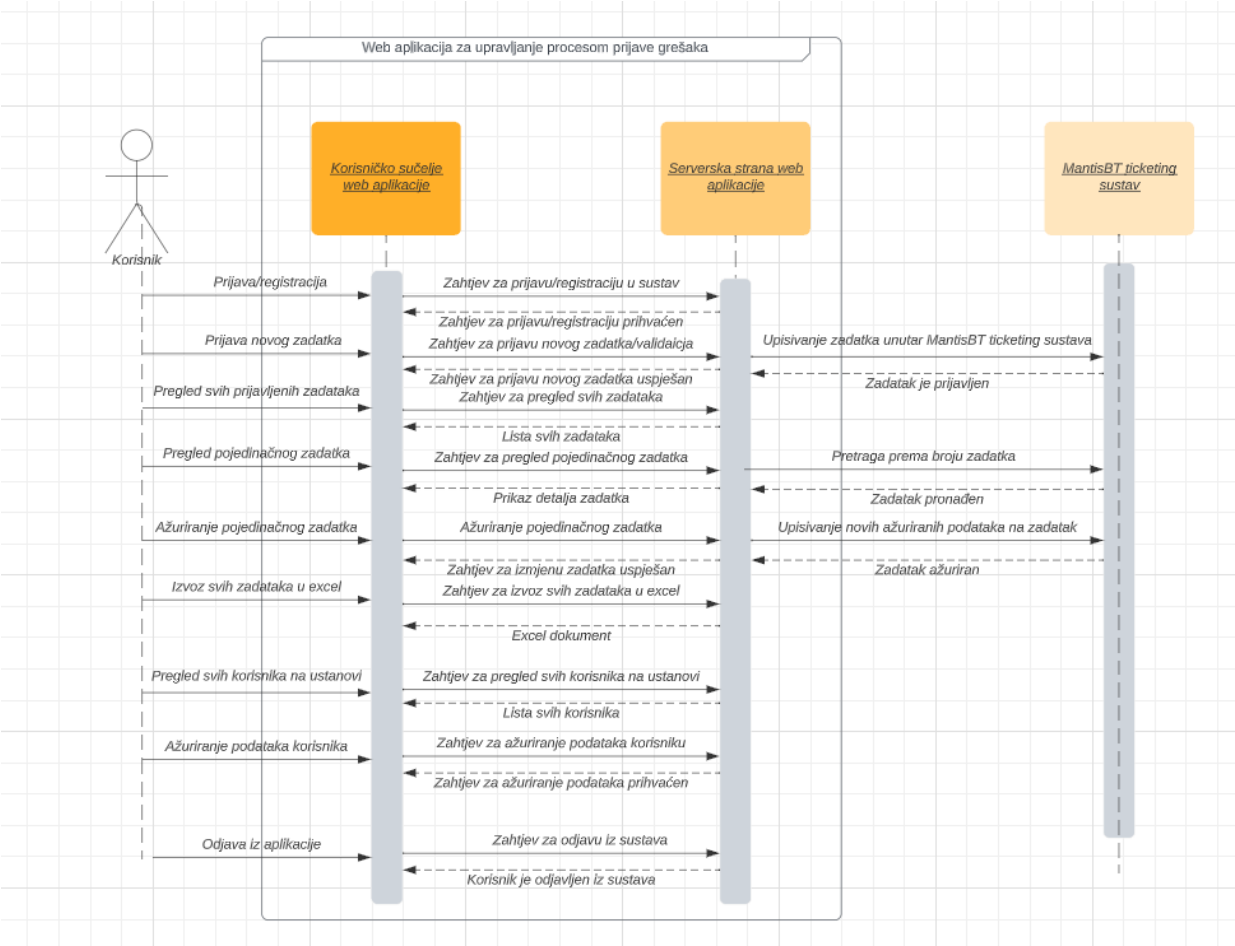
Korisnik šalje zahtjev za prijavu unutar sustava, sustav s druge strane mu odobrava ili ga odbija u tom zahtjevu, sve ovisi o upisanim podacima za prijavu.

Ukoliko korisnik uspije dobiti uspješnu potvrdu za prijavu u aplikaciju otvaraju mu se druge mogućnosti unutar iste. Prilikom prijavljivanja novog zadatka korisnik traži zahtjev za prijavu, nakon toga serverska strana obrađuje sve potrebne parametre te ako je sve validno upisuje zadatak unutar baze podataka i šalje taj isti zadatak na MantisBT.

Nakon što se nekolicina zadataka prijavila, korisnik ima mogućnost zatražiti listu svih prijavljenih zadataka, te će mu serverska strana vratiti listu zadataka u određenom periodu. Kada se na sučelje vrati lista zadataka, korisnik ima mogućnost putem jednog klika na zadatak otvoriti i pogledati detalje zadatka. Također, uz pogled detalja mogu se napraviti i promjene prioriteta zadatka ili zatvoriti cijeli zadatak ukoliko se desila neka pogreška.

Za nekakve dodatne analitike može se koristiti opcija izvoza u *Microsoft-ov* alat excel. Nakon što su objašnjene iteracije s zadacima, dolazi na red i administracija korisnika. Sama riječ administracija korisnika označava njihovo vođenje. Kad je riječ o administraciji korisnik ima opciju pregleda svih korisnika na ustanovi na koju je korisnik prijavljen. Kada dobije sa serverske strane listu svih korisnika, tada ima mogućnost ažuriranja podataka za svakog korisnika.

Posljednji proces u dijagramu toka je odjava iz aplikacije gdje korisnik pritiskom na gumb koji se nalazi u donjem desnom kutu početnog sučelja aplikacije šalje zahtjev prema serverskoj strani za odjavu korisnika. Kao rezultat tog procesa dolazimo na stranicu za prijavu te korisnik nije u mogućnosti pristupiti aplikaciji dok se ponovo ne prijavi u istu. Sekvencijalni dijagram možemo vidjeti na slici 5.



Slika 5. Sequence diagram (sekvencijalni dijagram)

4.3. KLASNI DIJAGRAM

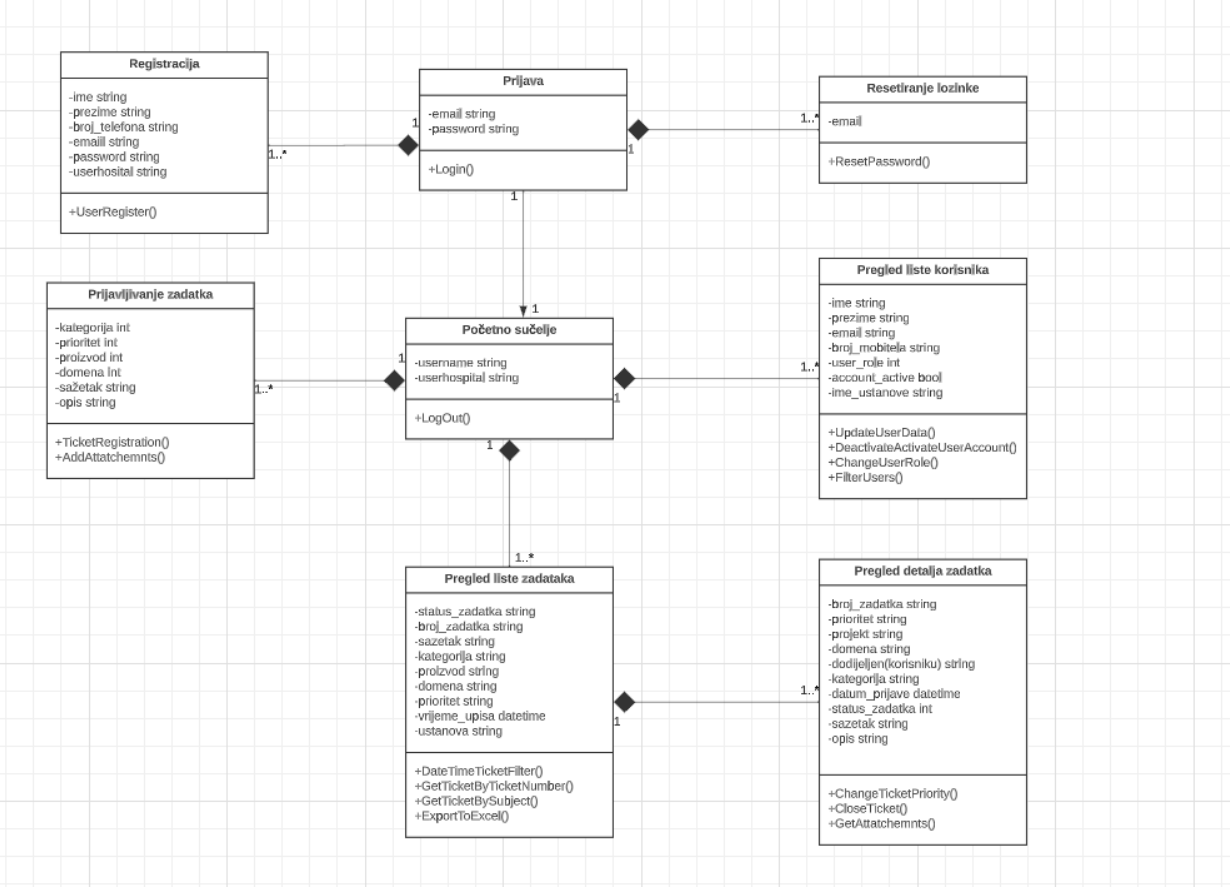
Kao što se može vidjeti, klasnom dijagramu postoji 5 klasa, svaka od tih klasa ima svoje atribute te operacije koje se mogu izvršavati unutar te klase. Također se može vidjeti da je svaka klasa povezana nekom vezom s drugom klasom, te veze zapravo prikazuju odnose između pojedinih klasa.

Prva klasa je klasa "Prijava" koja je povezana s klasom "Registracija" kompozicijskom vezom. To implicira da je klasa "Prijava" zavisna od klase "Registracija", što znači da je pristup sustavu moguć samo ako je prethodno izvršena registracija. Osim toga, klasa "Prijava" je povezana asocijacijskom vezom s klasom "Početno sučelje". Iz ove veze može se vidjeti da se prijava u sustav može obaviti jednom ili više puta, dok istovremeno u sustavu može biti prijavljen samo jedan korisnik s jedinstvenom email

adresom. Također, može se primijetiti da se isti korisnik može više puta registrirati u aplikaciju, ali isti email se može koristiti samo jednom za registraciju.

Također, klasu „Resetiranje lozinke“ može koristiti samo registrirani korisnik koji već ima unesenu elektroničku poštu u bazu. Postoje još i klase do kojih se može doći preko klase „Početno sučelje“, a to su: klasa „Prijavljivanje zadataka“ koja u sebi ima metode određene svrsi prijavljivanju zadataka, klasa „Pregled liste zadataka“ koja služi da nam projicira listu zadataka. Pod klasa klase „Pregled liste zadataka“ je klasa „Pregled detalja zadatka“ gdje se vidi detaljni prikaz pojedinačnog zadatka s njegovim trenutnim statusom.

Dalje dolazimo do klase „Pregled liste korisnika“ prvo će biti opisana veza ove klase s klasom „Početno sučelje“. Spomenute klase su povezane vezom gdje se vidi da postoji mogućnost pregledavanja n broja korisnika, dakle određenog broja korisnika koji je promjenjiv. Aplikativna struktura klasnog dijagrama se može vidjeti na slici 6.



Slika 6. Class diagram (klasni dijagram)

5. IMPLEMENTACIJA

Projekt je napravljen u tri dijela kako je gore spomenuto: *Frontend*[1]*, backend*[2] *i baza podataka*. *Flutter* programski okvir je zadužen za *frontend*, dok je *backend* napravljen u *Microsoftovom* alatu .NET Core 6.0 koji je služio kao web API. Baza podataka je napravljena isto u *Microsoftovom* alatu SQL Server.

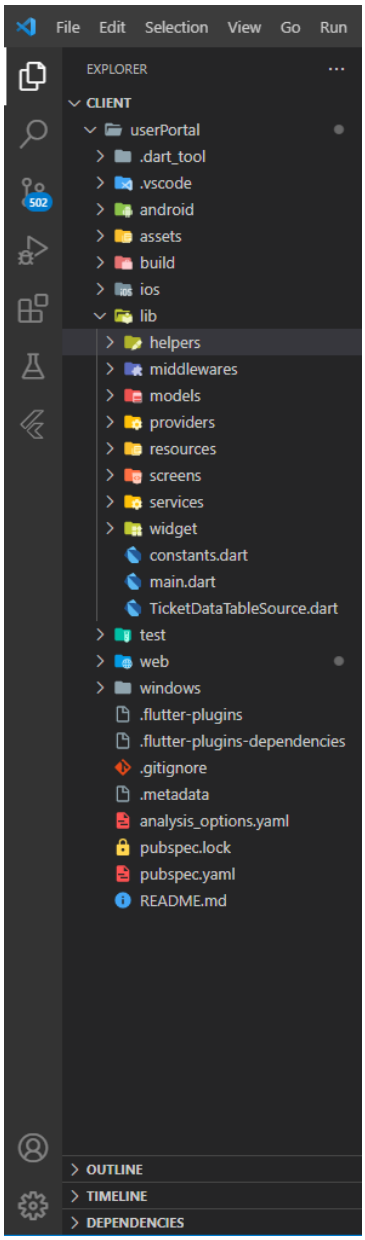
5.1. IMPLEMENTACIJA *FRONTENDA*

U ovom poglavlju će biti opisano samo postavljanje razvojnog okruženja za *frontend* te arhitektura samog pisanja koda i hijerarhija složenosti datoteka kako bi se što lakše snašli u projektu. Također, bit će prikazano i samo dizajniranje komponenti (*widget-a*[3]). Zadnji dio implementacije je povezivanje korisničkog sučelja sa serverskom stranom.

5.1.1. POSTAVLJANJE RAZVOJNOG OKRUŽENJA

Korisničko sučelje aplikacije izraženo je u programskom okviru *Flutter.* Prvobitno je potrebno instalirati *Flutter* na računalo, te preuzeti *Flutter SDK* koji je osnovni alat za razvoj *Flutter* aplikacija, te *Dart SDK* jer programski okvir *Flutter* koristi *Dart* kao programski jezik. Vrlo bitna stavka je i postavljanje okruženja gdje se dodaju putanje do *Flutter i Dart SDK-a* u varijablu okruženja kako bi omogućili pristup ovim alatima iz bilo koje lokacije u terminalu. Jednostavnom komandom u terminalu *„flutter doctor“* provjeravamo jesu li svi alati ispravno instalirani i konfigurirani. Također, za uređivač programskog koda je korišten koristio *Visual Studio Code*. Sam projekt je napravljen preko naredbe *„flutter create user_portal“* preko kojeg se stvara projekt sa hijerarhijom koja je prikazana na slici br. 7. Hijerarhija datoteka se nalazi s lijeve strane ali glavni direktorij je direktorij s nazivom *„lib“* u kojemu se nalaze komponente za daljnji razvoj aplikacije.

[1] Sučelje web aplikacije
[2] Dio aplikacije koji upravlja različitim funkcionalnostima i logikom aplikacije
[3] Osnovna građevna jedinica korisničkog sučelja u flutteru-u



Slika 7. Prikaz hijerarhije flutter aplikacije

Nadalje, u direktoriju *„lib“* se nalaze direktoriji *widget, services, screens, resources, providers, middlewares* te *helpers*. S desne strane slike se nalazi datoteka *„main.dart“* koja je inicijalna datoteka i ona se prva pokreće kada se otvara aplikacija.

5.1.2. OPIS HIJERARHIJE DATOTEKA

Kao je gore i napomenuto bitan direktorij za razvoj aplikacije je *„lib“*. Ostali direktoriji koji se nalaze u hijerarhiji služe da ovisno o platformi na kojoj se pokreće aplikacija izvršimo određene promjene. Kao što su npr. „zatraži pristup kameri“ ili „zatraži pristup mikrofonu“ itd. Svaki direktorij ima svrhu postojanja, tako se u direktoriju *widget* nalaze zasebne komponente koje se nalaze u aplikaciji. U direktoriju *screens* se nalaze svi ekrani koji su u aplikaciji. Što se tiče direktorija *models, providers, middlewares te services* oni služe isključivo za *mapiranje* podataka sa *frontend-a* i slanje prema *backend* strani.

5.1.3. DIZAJNIRANJE SUČELJA

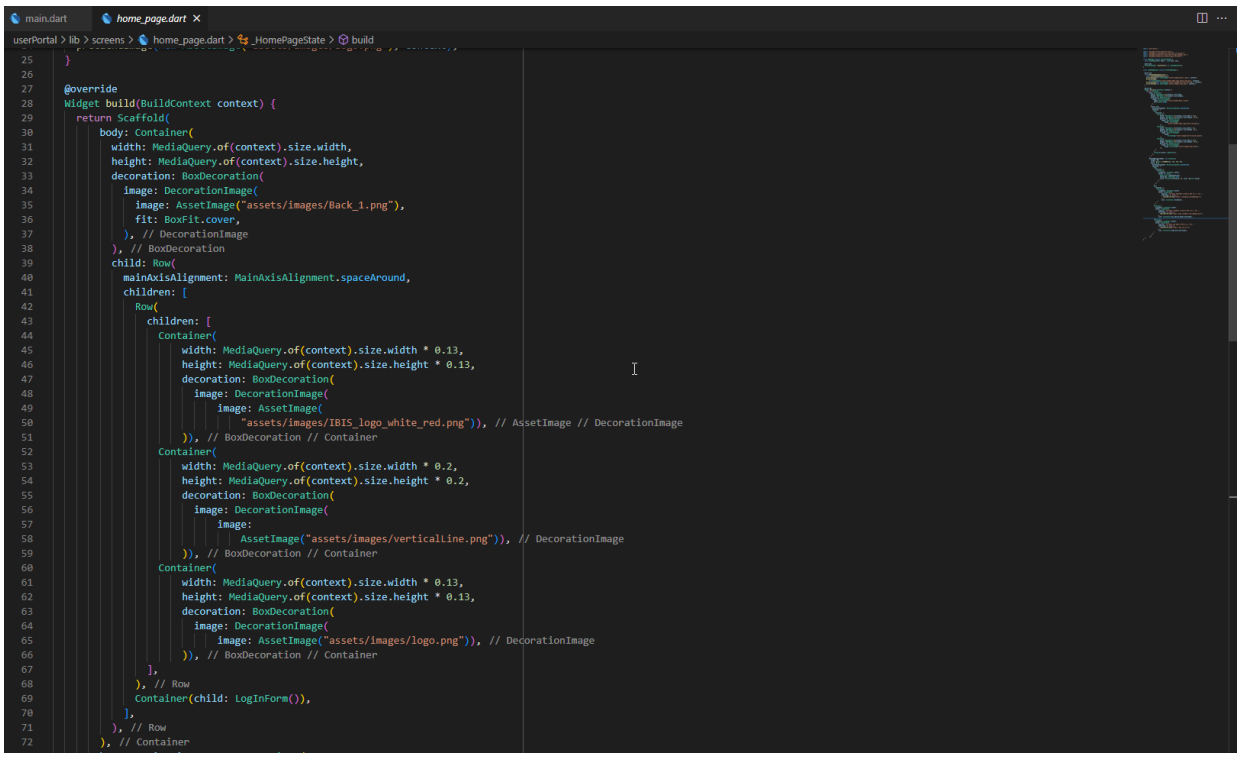
Pri dizajniranju sučelja za aplikaciju važno je istaknuti ključne aspekte koji će osigurati optimalno korisničko iskustvo i funkcionalnost aplikacije. Korištenje *Flutter* okvira omogućava izgradnju sučelja koje kombinira vizualnu privlačnost s intuitivnim interakcijama, doprinoseći uspješnom vođenju procesa prijave grešaka. Osnovni elementi, poput *Container*, *Row*, *Column*, *Text* i *Image widgeta*, omogućuju organizaciju i prikaz informacija na učinkovit način. Pravilno korištenje *layout widgeta* poput *Stack*, *Expanded* i *Flexible* osigurava dosljedan raspored elemenata na različitim veličinama zaslona. Jednako važna je paleta boja i tipografija. Odabirom pažljivo usklađenih boja i prilagodbom fontova putem *TextStyle* atributa, postiže se dosljedan vizualni identitet aplikacije. Navigacija i interakcije olakšavaju se upotrebom interaktivnih elemenata poput gumbova i padajućih izbornika.

Implementacija animacija dodaje dinamiku korisničkom iskustvu, čime se korisnicima pruža estetski ugodan i privlačan doživljaj tijekom korištenja aplikacije. Također, razmatranje agilnosti pojedinih elemenata osigurava pravilno prikazivanje sučelja na različitim uređajima. Bitan aspekt je i pristup materijalnom dizajnu *(eng. Material Design)* ili *Cupertino* dizajnu, ovisno o platformi na kojoj će aplikacija biti dostupna. Integracija vektorskih ikona i grafika dodaje vizualnu jasnoću i prepoznatljivost. Paketni menadžer i dijeljeni resursi pružaju učinkovitu organizaciju slika, fontova i drugih resursa, dok se optimizacija performansi postiže pažljivim upravljanjem veličinom slika i efikasnim prikazivanjem istih. Dizajn početne stranice je prikazan na slici 8.

Slika 8. Dizajn početne stranice

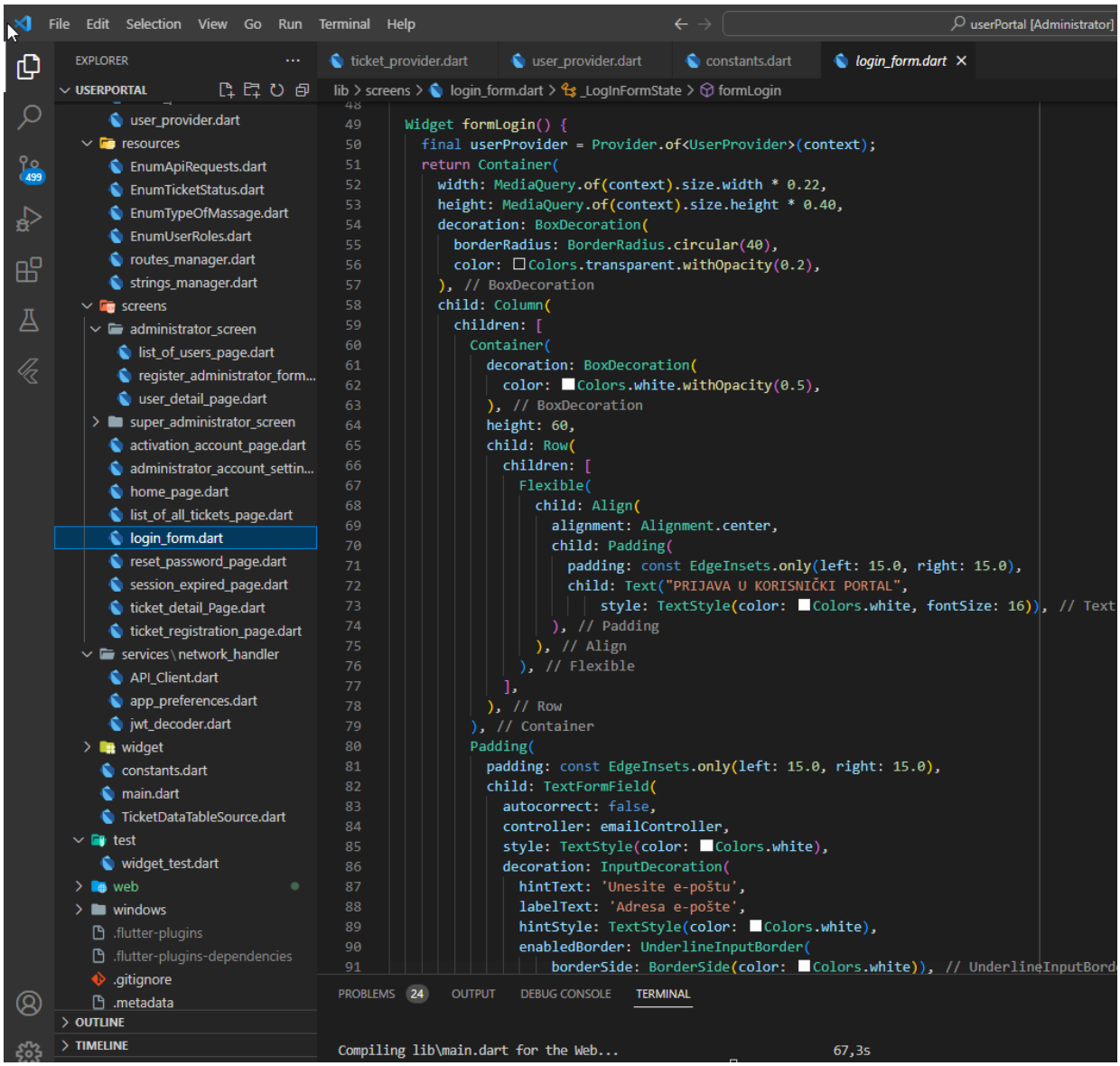
Na slici 9. je prikazan djelomičan programski kod od početnog sučelja. Glavna metoda koja generira grafički prikaz određenog UI elementa zove se *Widget build(BuildContext context)*. Metoda prima objekt *BuildContext* koji pruža informacije o kontekstu izgradnje *widgeta*, kao što su teme, veličine ekrana itd. Nadalje, postoji i osnovni *widget Scaffold* koji pruža kostur za prikaz UI elemenata kao što su *appBar*, *body* i *bottomNavigationBar*. Unutar *Scaffold-a* definirano je tijelo ekrana, a zove se *body* i u njemu se nalaze svi *widgeti* koji služe za iscrtavanje nekog objekta. Unutar *body-a* vidimo definirana svojstva kao što su i visina i širina ekrana, različite dekoracije

pojedinog objekta te na kraju pozivanje novog *widget-a* odnosno pozivanje forme za prijavu.

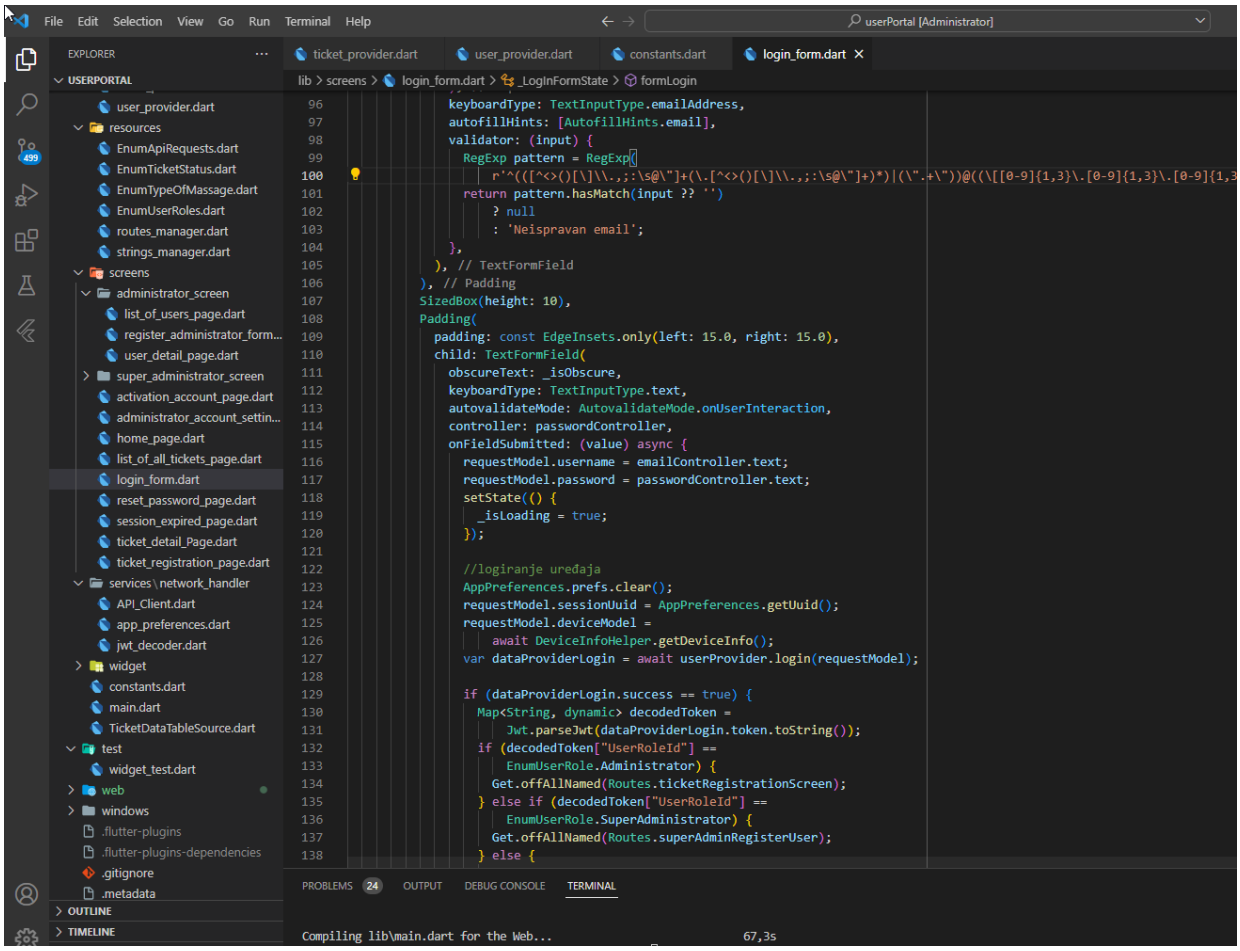


Slika 9. Programski kod početne stranice

Obrazac za prijavu je prikazan na slici br. 10 i slici br. 11 gdje se nalaze isto specificirani grafički elementi sa dužinom, širinom te dekoracijama. Unutar ovakve forme dolazi i do postavljanja *widget-a* koji se zovu TextFormField, a to su *widgeti* koji služe za unos podataka prema *provider-u* o kojem će biti napisano nešto više kasnije. TextFormField kao *widget* ima posebna svojstva za unos podataka, a to su: svojstvo automatskog ispravljanja, različite dekoracije izgleda takvog *widget-a, te validacije koje su potrebne za određenu vrstu unosa.*



Slika 10. Prikaz widgeta obrasca za prijavu 1

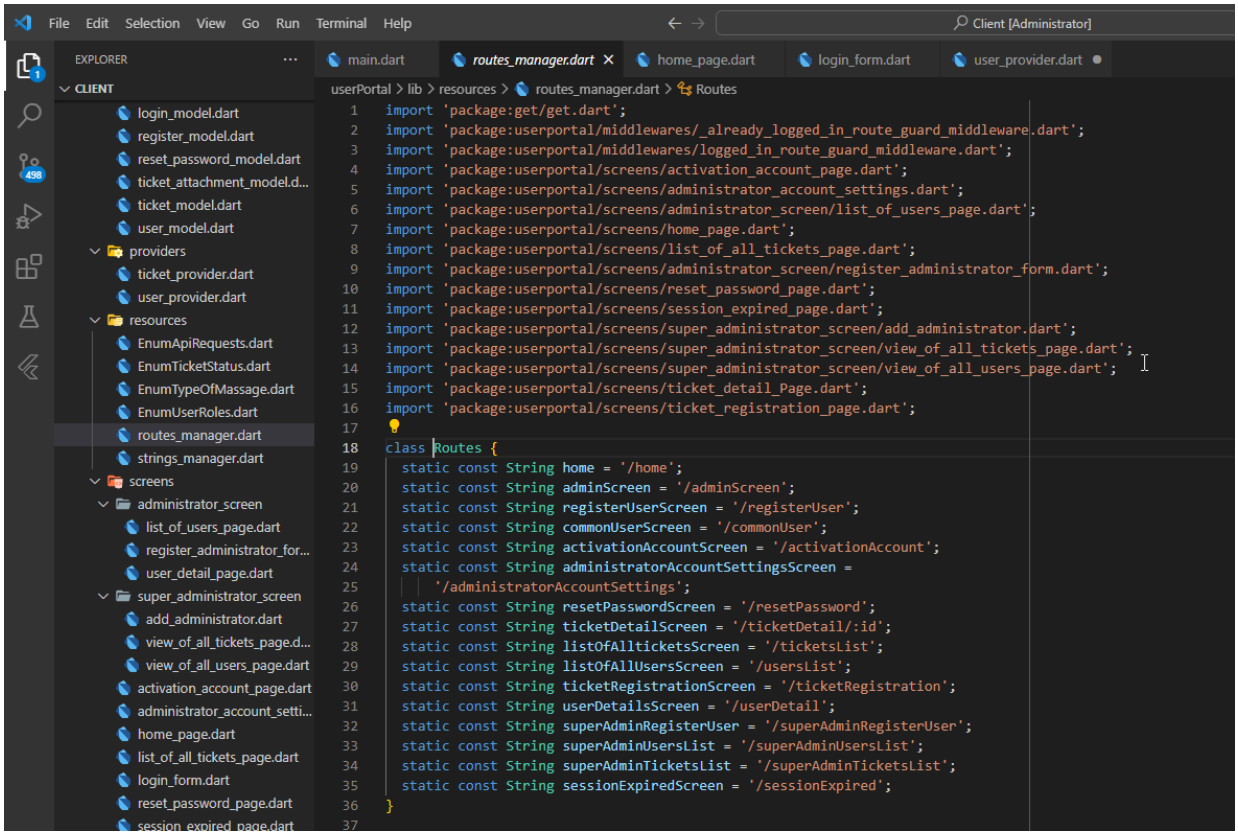


Slika 11. Prikaz widgeta obrasca za prijavu 2

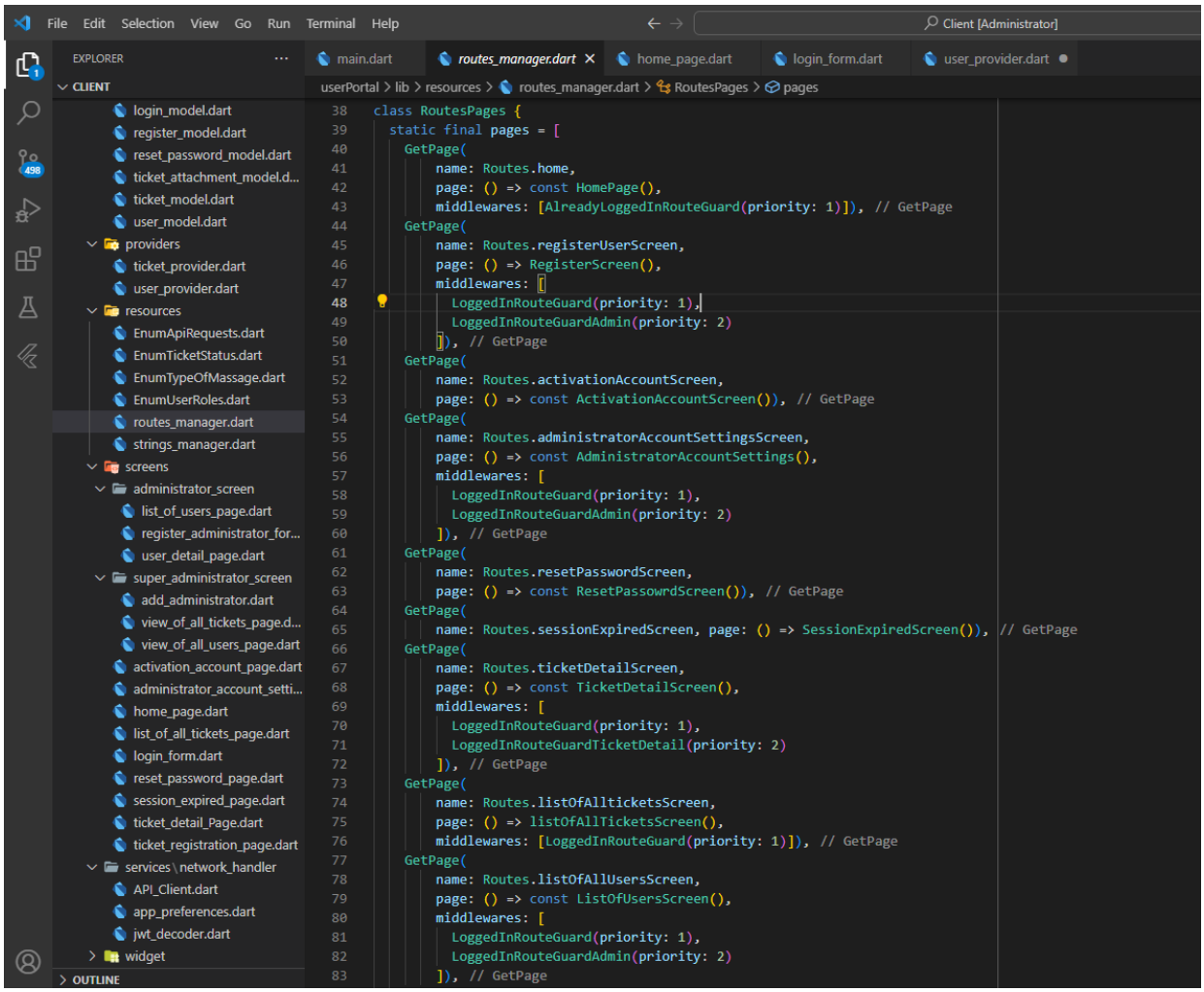
5.1.4. FLUTTER „ROUTING“

Kada se govori o flutter „routing“ pojmu u ovom projektu je korišten GetX popularan paket koji pruža različite alate i funkcionalnosti za razvoj, uključujući upravljanje stanjem, navigaciju i rutiranje. Unutar datoteke „routes_manager“ postoje dvije klase: u prvoj klasi „Routes“ na slici br. 12. se vide statičke konstantne *string* varijable s kojima definiramo imena url adresa, dok u drugoj klasi, prikaz je na slici br. 13, definirana iz *GetX* biblioteke *GetPage()* funkcija koja govori koje je ime određenoj ruti, na kojoj putanji mu je točno datoteka te *middlewares* u kojem postoji mehanizam koji se zove *„route guarding“* koji na jedan način štiti aplikaciju od trećih strana i neovlaštenih ulazaka, npr. ukoliko JWT token koji je došao sa servera, istekao,

mehanizam „route guard“ će ga automatski preusmjeriti na početnu stranicu gdje ne treba nikakva autorizacija.

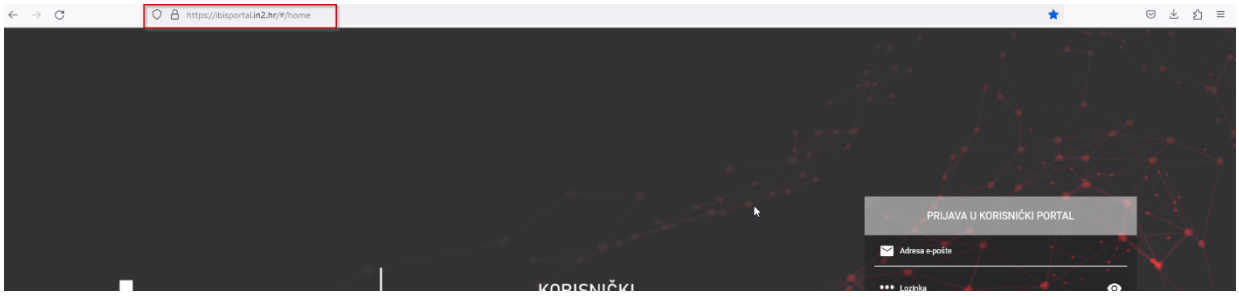


Slika 12. Prikaz klase „Routes“



Slika 13. Prikaz klase "RoutesPage"

Konačni prikaz i finalni proizvod „routing-a“ prikazan je na slici br. 14. u gornjem lijevom kutu.



Slika 14. Prikaz routing-a unutar web preglednika

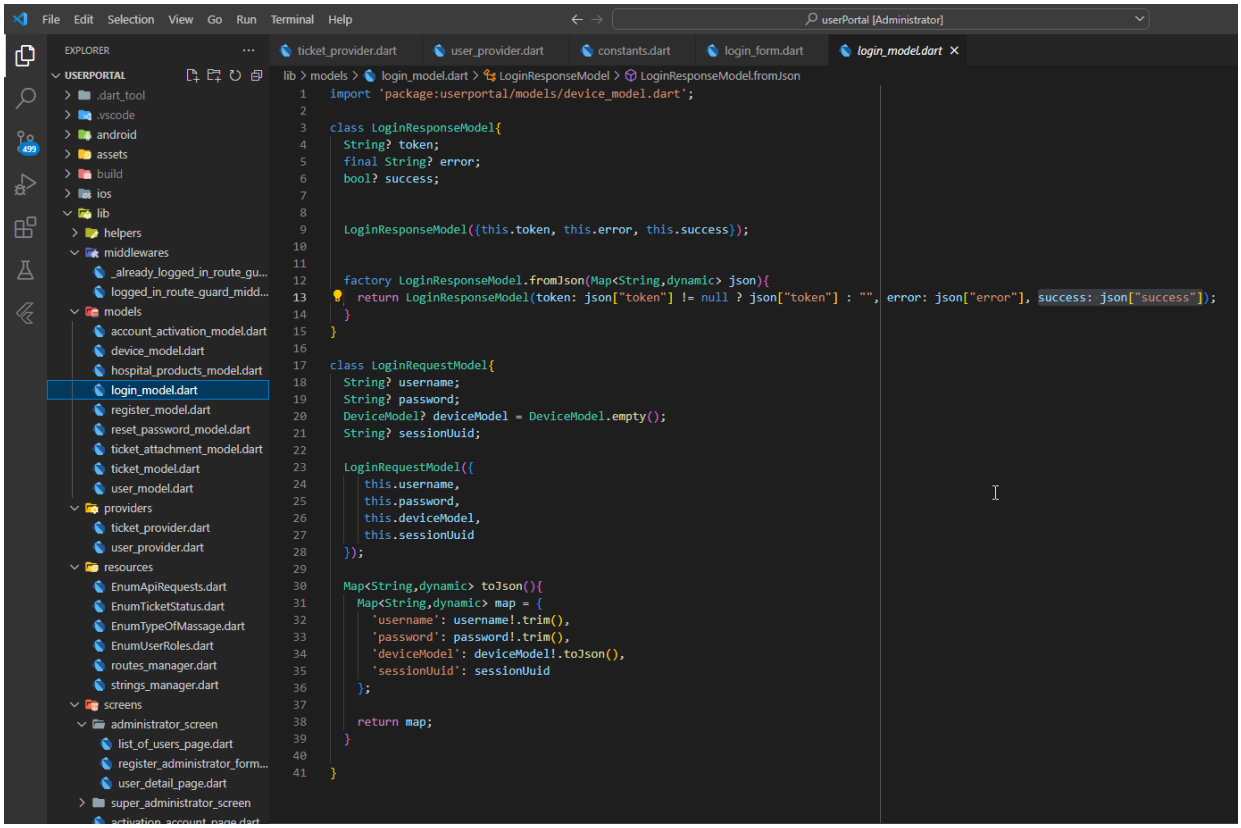
5.1.5. PRIPREMA PODATAKA I SLANJE NA BACKEND DIO

Ključan segment slanja podataka na *backend* dio obuhvaća temeljitu pripremu podataka za uspješnu komunikaciju s *backend* serverom. Ovaj proces revolucionira način na koji se razmjenjuju informacije između korisničkog sučelja i *backend* infrastrukture, omogućavajući efikasno i sigurno upravljanje prijavama grešaka. Prvi korak u tom procesu uključuje definiranje modela podataka. Precizno i dosljedno definiranje struktura podataka koje se šalju prema *backend-u*, kao i očekivanog odgovora, osigurava konzistentnost i preciznost komunikacije. Ovi modeli služe kao most između korisničkog sučelja i *backend-a*, osiguravajući da se podaci pravilno interpretiraju na obje strane. Upotreba *Dart* objekata omogućava lako upravljanje podacima koje želimo slati. Kreiranje instanci definiranih modela omogućava čitljiv i organiziran pristup podacima, osiguravajući da se informacije dosljedno prikupljaju i pravilno pripremaju za slanje na *backend*. Nadalje, ključna komponenta je proces serijalizacije podataka. Ovdje, alati poput *json_serializable* biblioteke omogućuju konverziju *Dart* objekata u format JSON koji je lako prenosiv putem HTTP zahtjeva. Ovaj proces čini podatke prikladnima za prijenos putem mreže, omogućavajući *backend* serveru da ih razumije i interpretira.

Korištenje HTTP zahtjeva putem paketa poput http omogućava aplikaciji da uspostavi komunikaciju s *backend-om*. Odabir odgovarajuće metode zahtjeva, postavljanje zaglavlja i pravilno postavljanje tijela zahtjeva s serijaliziranim podacima osigurava da podaci budu sigurno preneseni i pravilno interpretirani na *backend-u*. Važan korak je obrada odgovora od backend servera. Ovaj dio uključuje pretvaranje JSON odgovora u *Dart* objekte, analizu statusnih kodova kako bi se razumjelo je li zahtjev uspješan, te rukovanje greškama i drugim informacijama koje se vraćaju.

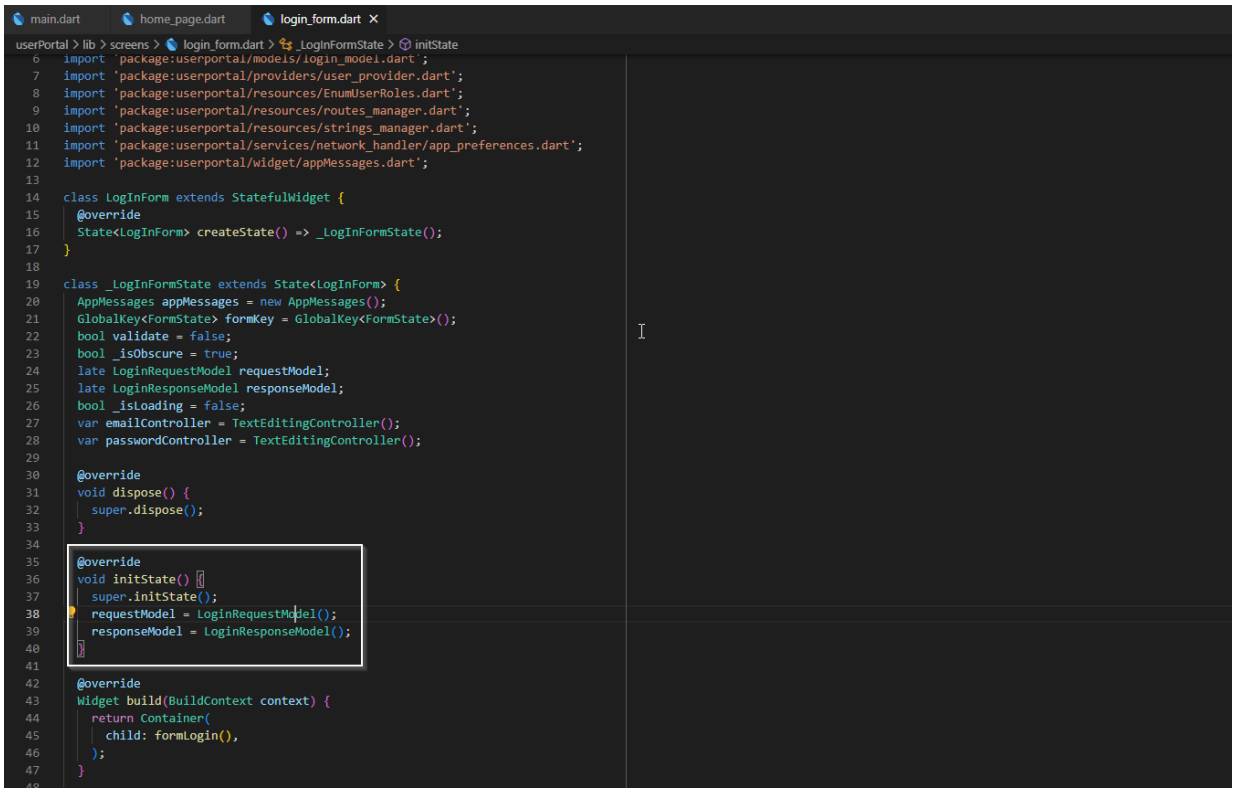
Asinkrono programiranje, omogućeno async i await konstrukcijama, ključno je za osiguravanje nesmetane izvedbe aplikacije tijekom procesa slanja i primanja podataka. Naposljetku, proces primanja odgovora, koji može sadržavati različite informacije kao što su relevantni podaci, statusni kodovi i zaglavlja, omogućava aplikaciji da reagira sukladno primljenim informacijama, pružajući korisnicima točne povratne informacije. Kroz ovaj detaljan proces, aplikacija postiže visoku razinu interakcije s *backend-om*, osiguravajući pouzdanost, sigurnost i dosljednost u komunikaciji.

Na slijedećem primjeru u aplikaciji će biti prikazan proces pripreme podataka i slanje na *backend* u procesu prijave korisnika. Nakon što se napravilo korisničko sučelje za prijavu u sustav. Potrebno je definirati *request* model i *response* model odnosno model koji se šalje i model koji se prima natrag sa *backend* strane. Na slici br. 15. je programski kod *„login_model“* datoteke u kojoj je definiran *LoginRequestModel* odnosno model koji će se s nekakvim parametrima slati prema *backend* strani, te *LoginResponseModel* u koji će se spremiti podaci koji se vrate sa *backend* strane.



Slika 15. Prijava request i response model

Na slici br. 16 je prikazano stvaranje *request* i *response* objekta kada se govori o procesu prijave u sustav. Također su na slici ispod definirana i dva *controller-a* koji se nalaze u nekakvom *widget-u* i takvi *controller-i* sadrže podatke koje je korisnik upisao u ulazna polja u formi.



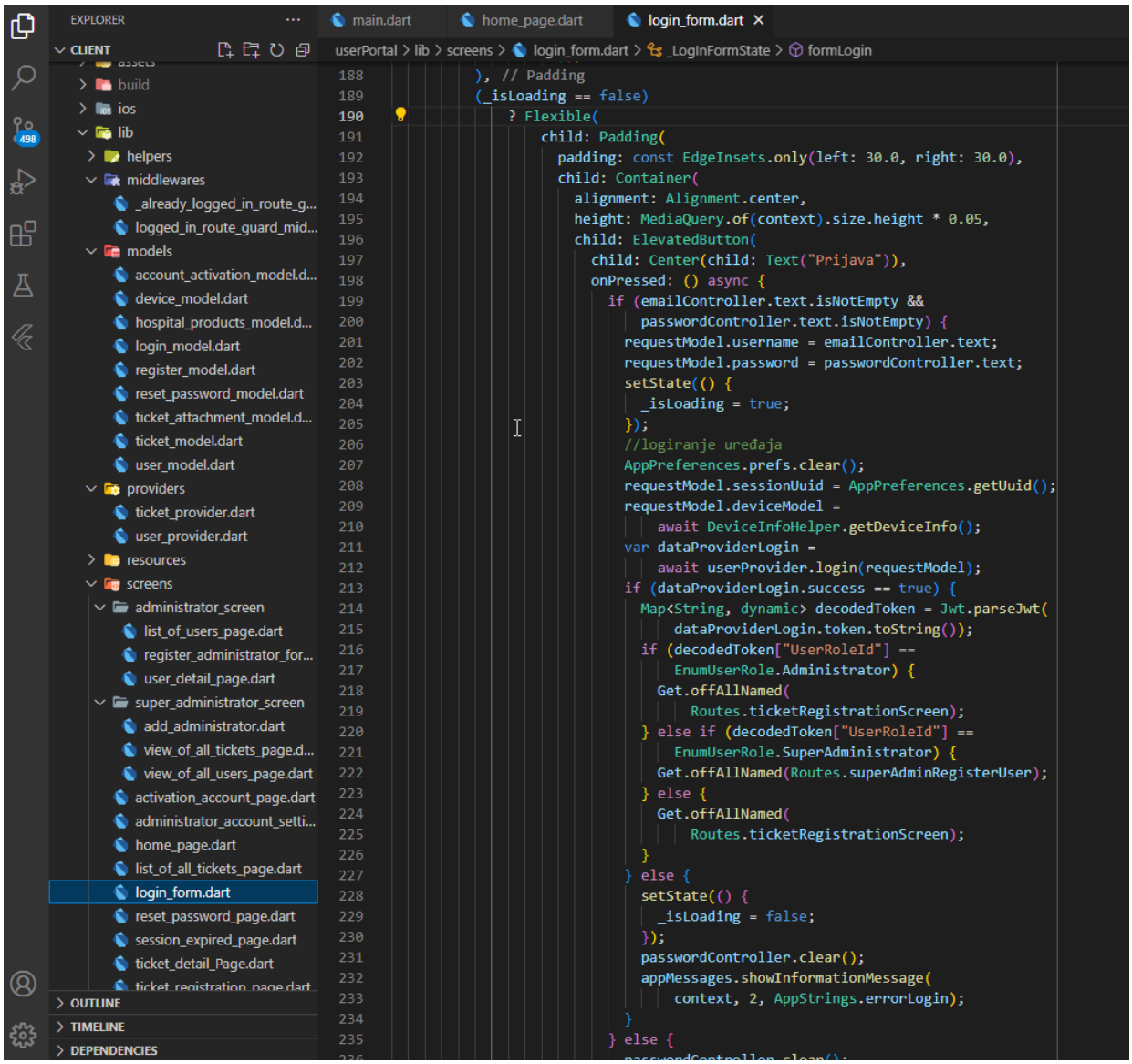
Slika 16. Instanciranje request i response modela unutar InitState() metode

Na slici br. 17. se prikazuje korisničko sučelje. Kada korisnik upiše svoj e-mail tada se prvo taj e-mail provjerava prema pravilima koja su napisana na slici br. 17. Ako je sve u redu taj tekst koji je upisan se sprema u varijablu koja je gore definirana a zove se *emailController.* Isti postupak se ponavlja i za lozinku, samo nema proces validacije kao kod e-maila nego ide drugačijom logikom.

```
80          Padding(
81            padding: const EdgeInsets.only(left: 15.0, right: 15.0),
82            child: TextFormField(
83              autocorrect: false,
84              controller: emailController,
85              style: TextStyle(color: Colors.white),
86              decoration: InputDecoration(
87                hintText: 'Unesite e-poštu',
88                labelText: 'Adresa e-pošte',
89                hintStyle: TextStyle(color: Colors.white),
90                enabledBorder: UnderlineInputBorder(
91                    borderSide: BorderSide(color: Colors.white)), // UnderlineInputBorder
92                prefixIcon: Icon(Icons.mail, color: Colors.white),
93                filled: true,
94                labelStyle: TextStyle(fontSize: 12, color: Colors.white),
95              ), // InputDecora
96              keyboardType: Text                    Type: List<String>  ss,
97              autofillHints: [AutofillHints.email],
98              validator: (input) {
99                RegExp pattern = RegExp(
100                   r'^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-
101               return pattern.hasMatch(input ?? '')
102                   ? null
103                   : 'Neispravan email';
104             },
105           ), // TextFormField
106         ), // Padding
```

Slika 17. Komponenta za unos elektroničke pošte u procesu prijave

Na slici br. 18. se prikazuje proces kada pritiskom na gumb pod imenom *„Prijava“* počinje asinkrona radnja gdje se podaci iz dva *controller-a* spremaju u gore definirani *request* model koji će se proslijediti unutar *„login“* metode koja se nalazi unutar *„userProvider“* klase. Čeka se povratna informacija s *backend* strane da može odlučiti koju akciju će poslati korisniku, da ide na jedan od tri korisnička sučelja ovisno o tome što dobije sa *backend* strane ili da mu pošalje povratnu informaciju da je netočna lozinka ili da e-mail adresa koja je upisana ne postoji. Takvu sekvencu možemo vidjeti od 211 linije na već gore spomenutoj slici br. 18 gdje se iščitavaju podaci koji su dobiveni kroz JWT token te se dekodiraju i nakon toga je programu jasno gdje na temelju korisnikove uloge treba korisnik završiti ukoliko su mu pristupni podaci odgovarajući.



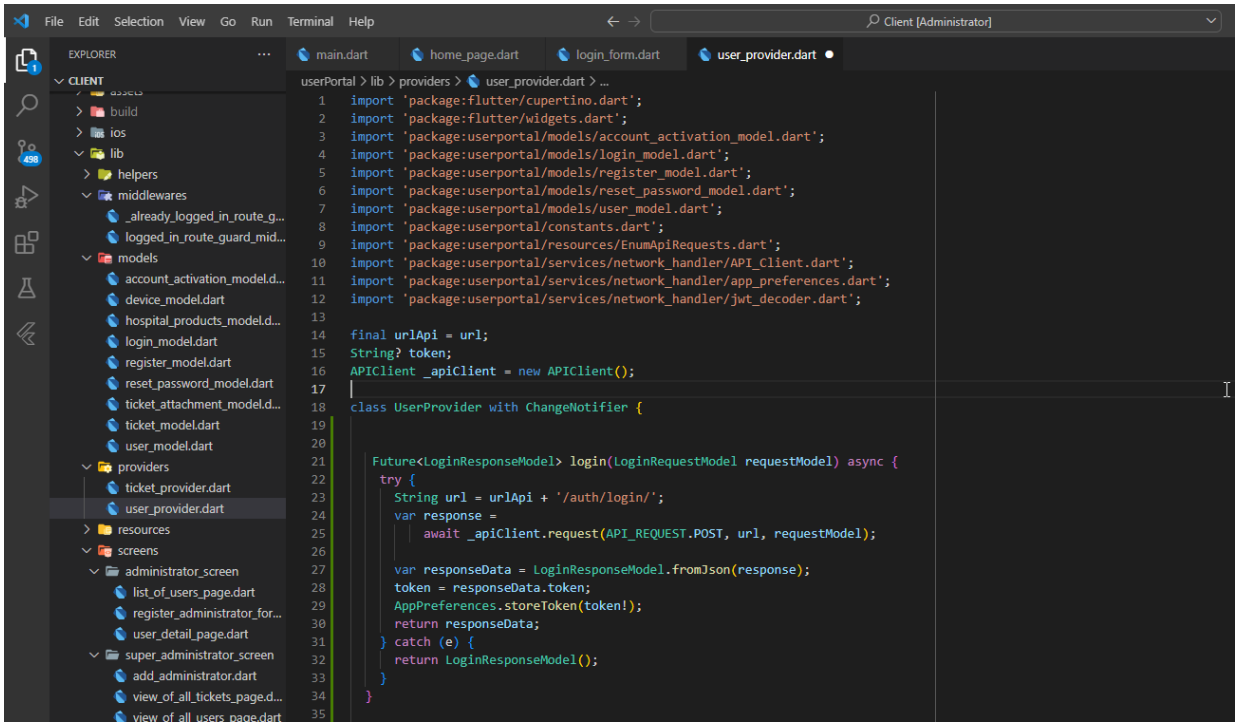
Slika 18. Logika na gumbu kada se pritisne akcija prijava

Daljnjim tijekom program asinkrono odlazi u klasu *„userProvider“* u metodu „login“ čija definicija se može vidjeti na slici br. 19. U klasi postoji funkcija *„login“* koja prima „LoginRequestModel“ kao parametar iz prethodnog obrasca za prijavu. Također, u funkciji je definirana *url* adresa *backend* dijela na koju će se poslati prethodno napisani parametri unutar objekta. U ovoj klasi definirana je instanca klase naziva *„APIClient“* koja je napravljena generički da svi *request-ovi* idu kroz nju.

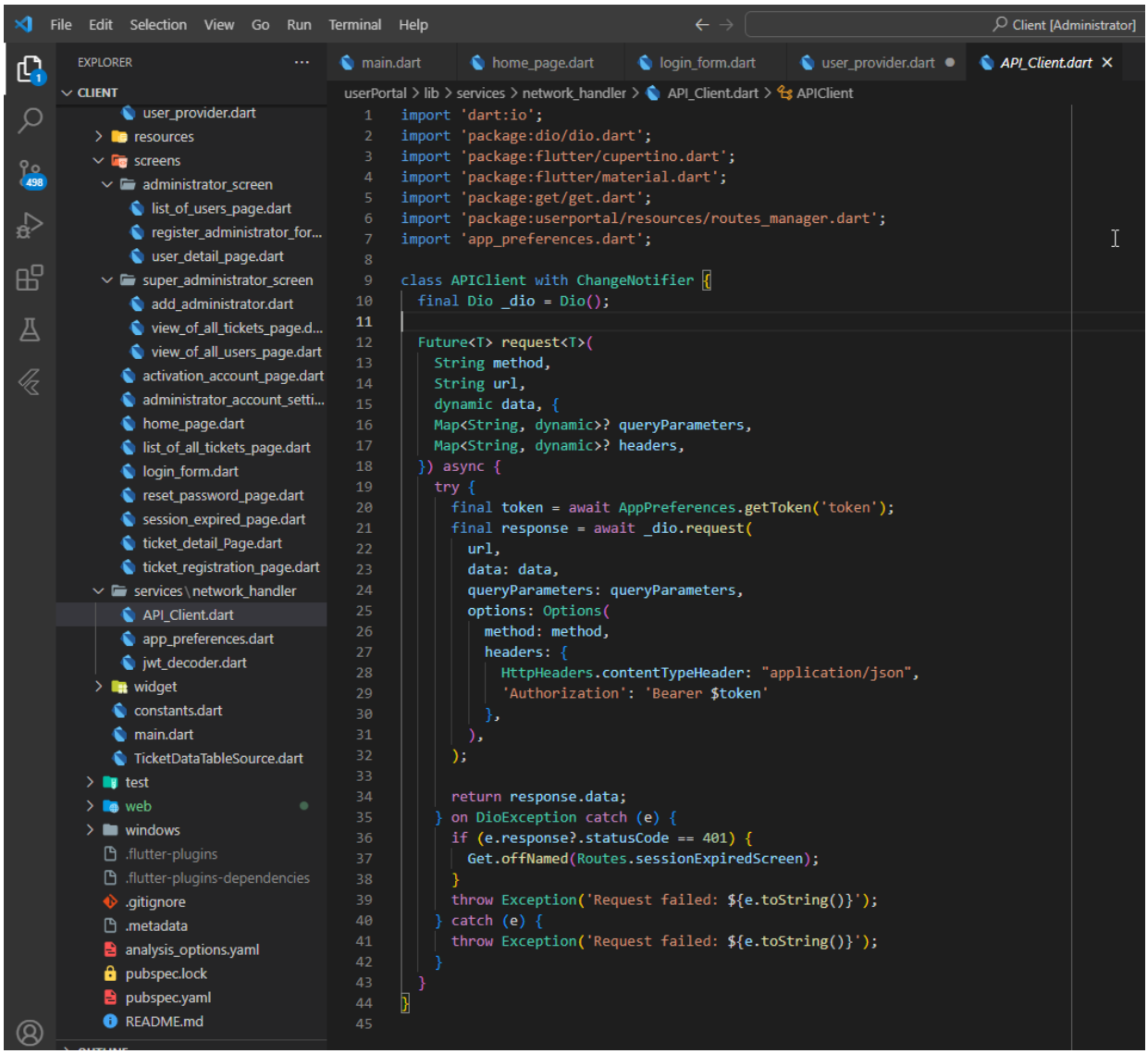
Prikaz klase „APIClient“ se može vidjeti na slici br. 20. koja ima implementirani flutter-ov paket *Dio*[4], te sadržava funkcija koja se zove *„request“* te prima generički model i generičke parametre. Ovo je jako zahvalna funkcija jer svi zahtjevi prema server

[4] Popularna biblioteka za izvođenje HTTP zahtjeva

strani idu preko te metode samo mijenjamo ulazne parametre na mjestu gdje je želimo pozvati. U ovom primjeru kada se šalje zahtjev za prijavu u sustav šalje se POST *request* koji šalje na server upisane podatke sa forme u *body-u*, a u *header-u* definira način slanja podataka prema server strani a to je *„application/json“* format, te ukoliko imamo autorizaciju u obliku JWT token-a tada šaljemo i token s kojim manipuliramo na serverskoj strani.



Slika 19. UserProvider klasa i metoda LogIn



Slika 20. Prikaz generičke klase APIClient

Server kada procesuira podatke vraća set podataka u JSON obliku koji se vraća nazad u klasu *„UserProvider“* te se onda mapiraju podaci unutar *„LoginResponseModel“* klase. *„LoginResponseModel“* kao objekt se koristiti na sučelju kako bi korisničko sučelje dobilo informaciju treba li korisnika propustiti unutar aplikacije ili mu npr. treba ispisati poruku da mu je netočna lozinka ukoliko je upisao krive pristupne podatke.

5.2. IMPLEMENTACIJA *BACKENDA*

U ovom poglavlju će se govoriti o samoj implementaciji drugog dijela aplikacije odnosno serverske strane. Za serversku stranu kako je gore i spominjano odabrana je tehnologija od *Microsofta .NET Core 6.0.* Nadalje, opisan će biti sam dotok podataka do serverske strane aplikacije, njihova obrada kroz različite validacije te pripremanje samih podataka da budu spremni za unos podataka u bazu. Serverska strana je cijela napisana prema DI (eng. *Dependency Injection)* principu koji će biti objašnjen kroz nastavak.

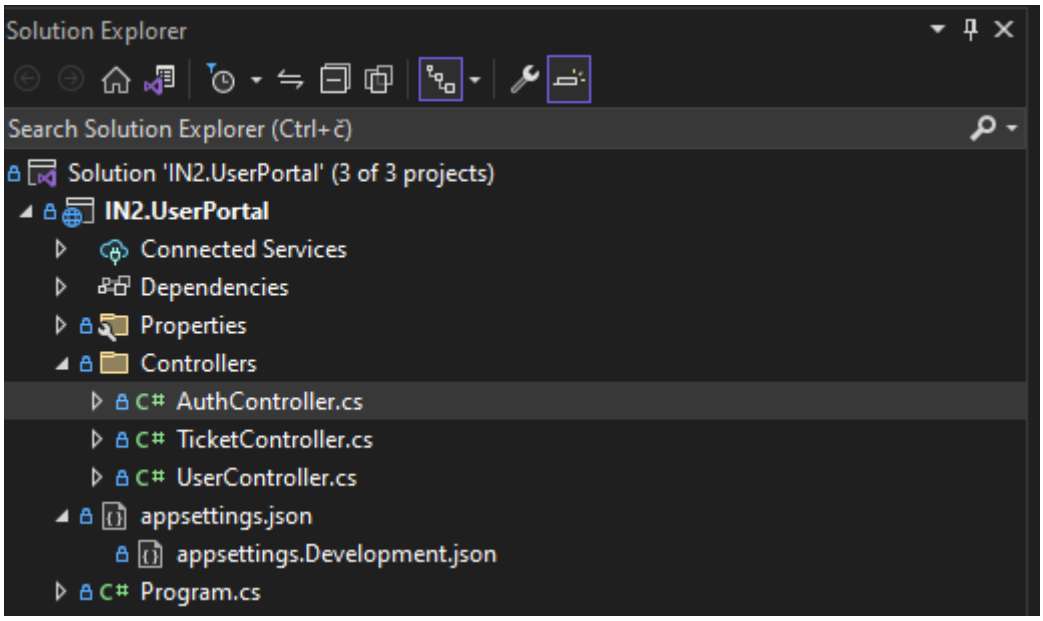
5.2.1. POSTAVLJANJE RAZVOJNOG OKRUŽENJA

U ovom poglavlju, detaljno je opisan proces postavljanja razvojnog okruženja za .NET Core 6.0, koje će omogućiti razvoj, testiranje i implementaciju aplikacija temeljenih na .NET platformi. Postavljanje odgovarajućeg razvojnog okruženja ključno je za uspješno izvođenje projekata, a ovaj dio istraživanja pružit će čitateljima sve potrebne korake i upute za postavljanje okruženja.

Upute za postavljanje okruženja [9]:

1. **Instalacija .NET SDK-a**: Prvi korak u postavljanju razvojnog okruženja za .NET Core 6.0 je instalacija .NET SDK-a. .NET SDK predstavlja Software Development Kit, koji uključuje sve potrebne alate za razvoj .NET aplikacija.
2. **Izbor razvojnog okruženja**: Microsoftova integrirana razvojna okruženja pružaju obilje mogućnosti za razvoj aplikacija. Visual Studio 2022, kao najnovija verzija, pruža bogat skup alata i resursa za razvoj .NET aplikacija.
3. **Kreiranje i pokretanje projekta**: pokrenut je *Microsoft Visual Studio* te je odabrana opcija kreiranja web API projekta.

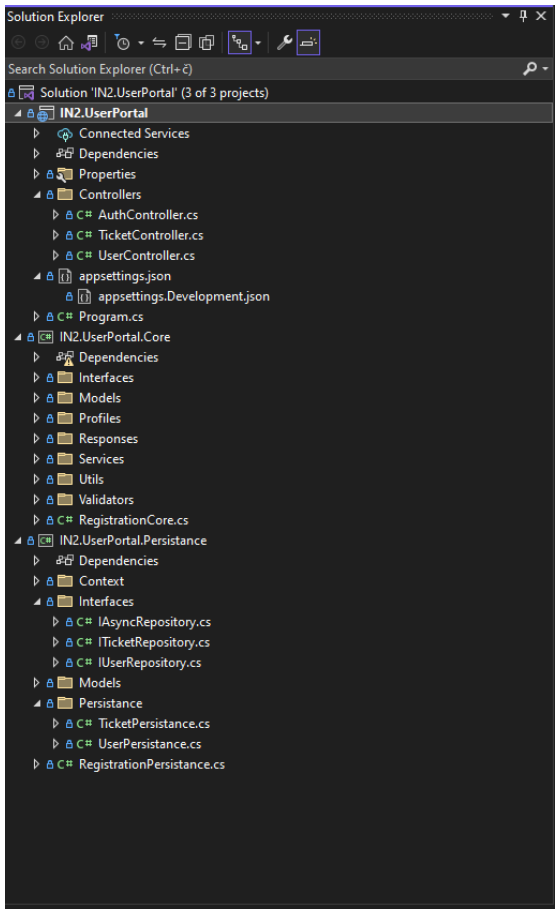
Nakon kreiranja projekta dobije se struktura slična strukturi na slici br. 21. gdje projekt API sadrži direktorij *„Controllers“,* te klasu *„Program.cs“* i još par datoteka. Zapravo je direktorij „Controllers“ vrlo esencijalan u ovoj priči jer na njega dolaze svi zahtjevi iz vanjski aplikacija, dok datoteka *„Program.cs“* predstavlja glavnu ulaznu točku na API. Ova datoteka ima ključnu ulogu u pokretanju web API-ja, važna je i u konfiguraciji hosta i serviranju HTTP zahtjeva.



Slika 21. Prikaz hijerarhije na serverskoj strani

5.2.2. OPIS HIJERARHIJE DATOTEKA

Na slici br. 22. se vidi raspored projekata na serverskoj strani, hijerarhija prikazuje raspored od tri projekta. Prvi projekt pod nazivom *„IN2.UserPortal“* je pristupna točka na web API u kojoj se nalaze pristupni *controller-i* za ulaz na serversku stranu. Slijedeći projekt je pod nazivom *„IN2.UserPortal.Core“* koji služi za različite validacije unesenih podataka kroz *frontend*, te dalje šalje pripremljene podatke u zadnji sloj *backend* dijela. Zadnji sloj *backend* dijela je projekt pod nazivom *„IN2.UserPortal.Persistance“,* najčišći sloj što se tiče podataka, koji je sama veza sa bazom podataka, u njemu se nalaze glavni modeli koji su preslika modelima na bazi podataka.



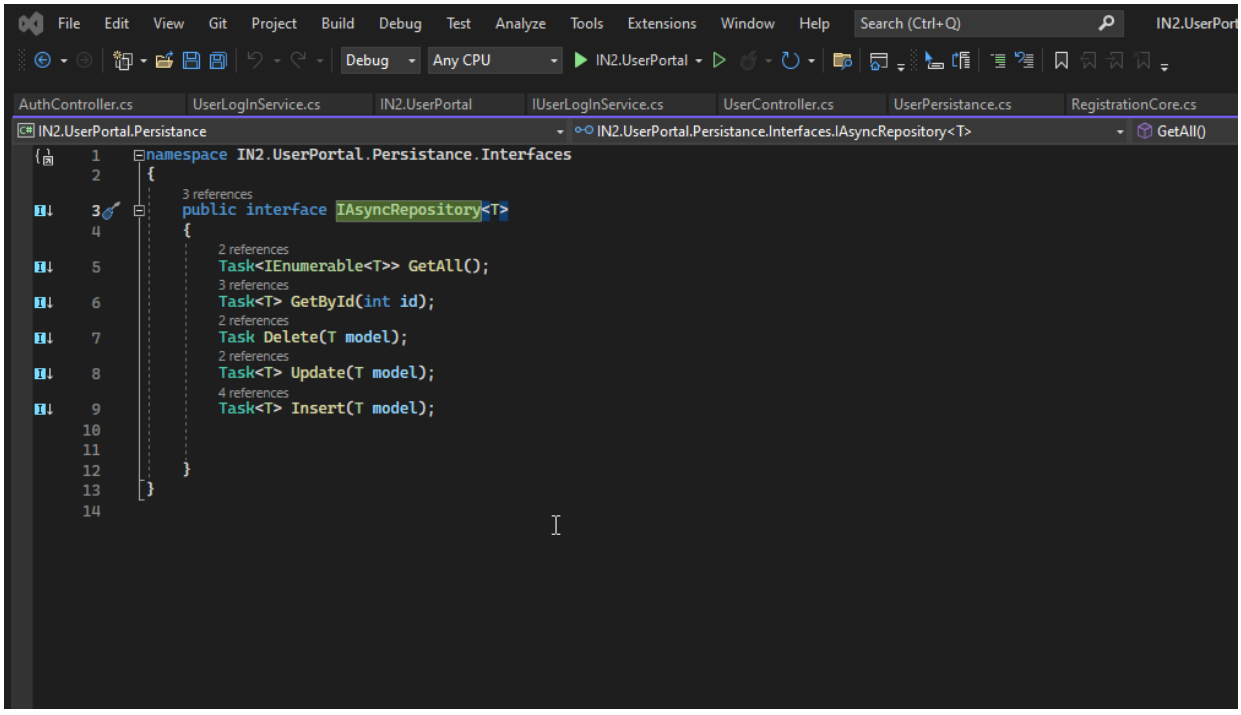
Slika 22. Prikaz hijerarhije datoteka na serverskoj strani

5.2.3. ORGANIZACIJA PISANJA KODA – REPOSITORY PATTERN I DEPENDENCY INJECTION

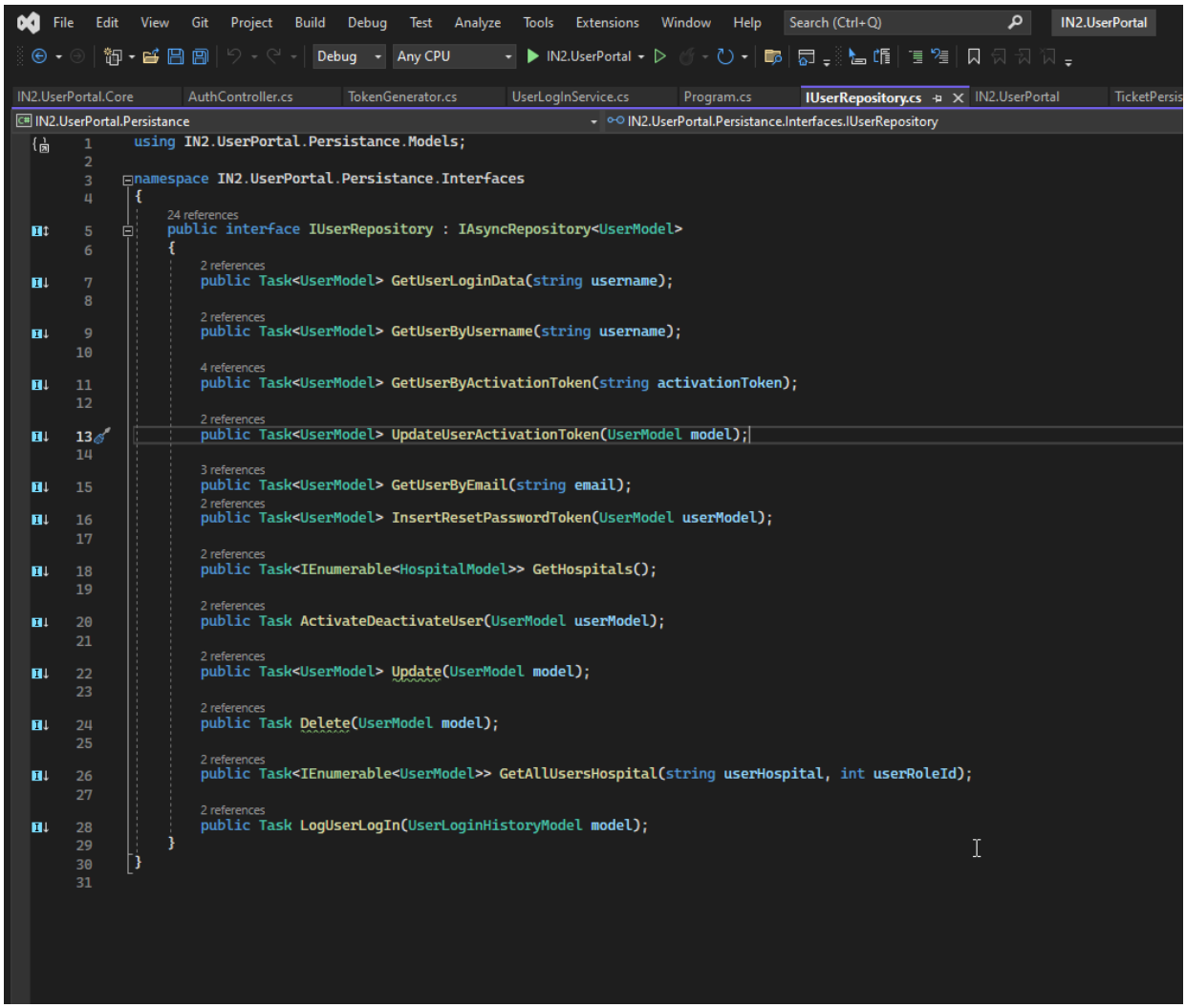
U ovom odjeljku će biti opisana organizacija pisanja programskog koda gdje su korišteni „repository pattern“ i „dependency injection“ principi.

Za početak što je *„repository pattern“*? *„Repository pattern“* je konceptualni obrazac dizajna softvera koji se koristi za organizaciju pristupa podacima unutar aplikacije. Ovaj obrazac omogućava da se pristup podacima izdvoji u poseban sloj, nazvan *"repository"* (repozitorij), što olakšava upravljanje podacima, poboljšava održavanje i omogućava bolje testiranje.

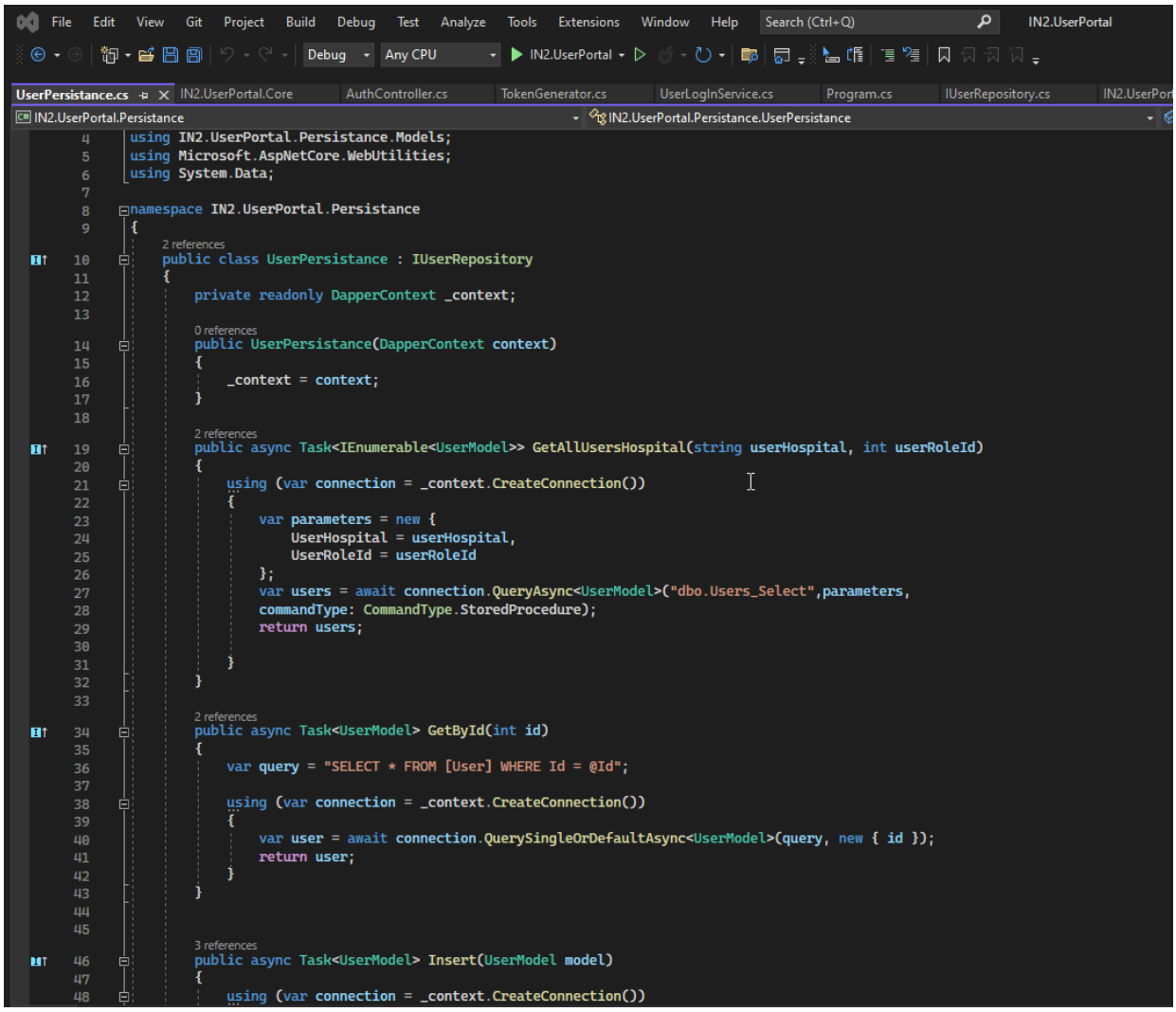
Glavna ideja iza *„repository pattern-a“* je da se pristup podacima izdvoji iz poslovne logike aplikacije. To se postiže tako da se definira sučelje (eng. *interface*) koje opisuje osnovne operacije koje je potrebno izvršiti nad podacima, kao što su dohvaćanje, dodavanje, ažuriranje i brisanje. Nakon toga se implementira to sučelje s konkretnim operacijama za pristup podacima, najčešće kroz komunikaciju s bazom podataka ili drugim izvorima podataka. Prednosti *„repository pattern-a“* odvajanje logike poslovanja od logike pristupa podacima, olakšava se testiranje poslovne logike bez stvarnog pristupa bazi podataka, povećana sigurnost u smislu smanjenja rizika od SQL injekcija i sličnih problema te na kraju je dolazimo do povećane čitljivosti i boljeg razumijevanja koda. Na slici br. 23. 24., i 25. vidimo primjenu *„repository patterna“* gdje imamo glavni *interface* naziva *„IAsyncRepository“* koji sadrži generičke metode koje primaju generičke parametre. Nadalje, imamo *interface „IUserRepository“* koji nasljeđuje metode od glavnog *interface-a* za nekakve generičke radnje dok je *„IUserInterface“* napravljen isključivo za metode koje se odnose na korisnikove interakcije.



Slika 23. Globalni interface - repository pattern



Slika 24. IUserRepository klasa

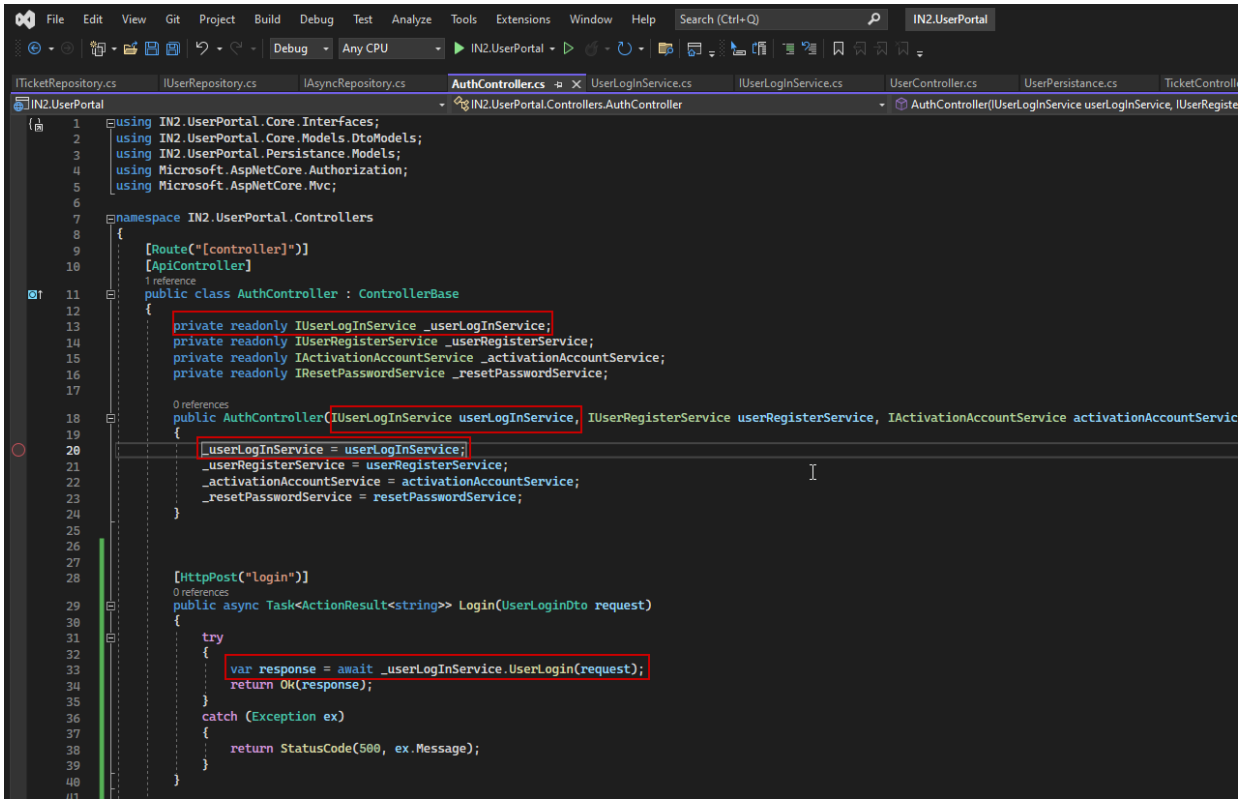


Slika 25. UserPersistance klasa

Nakon *„repository patterna“* bit će objašnjen i drugi princip pisanja programskog koda unutar ovog rada, a to je *Dependency injection.*

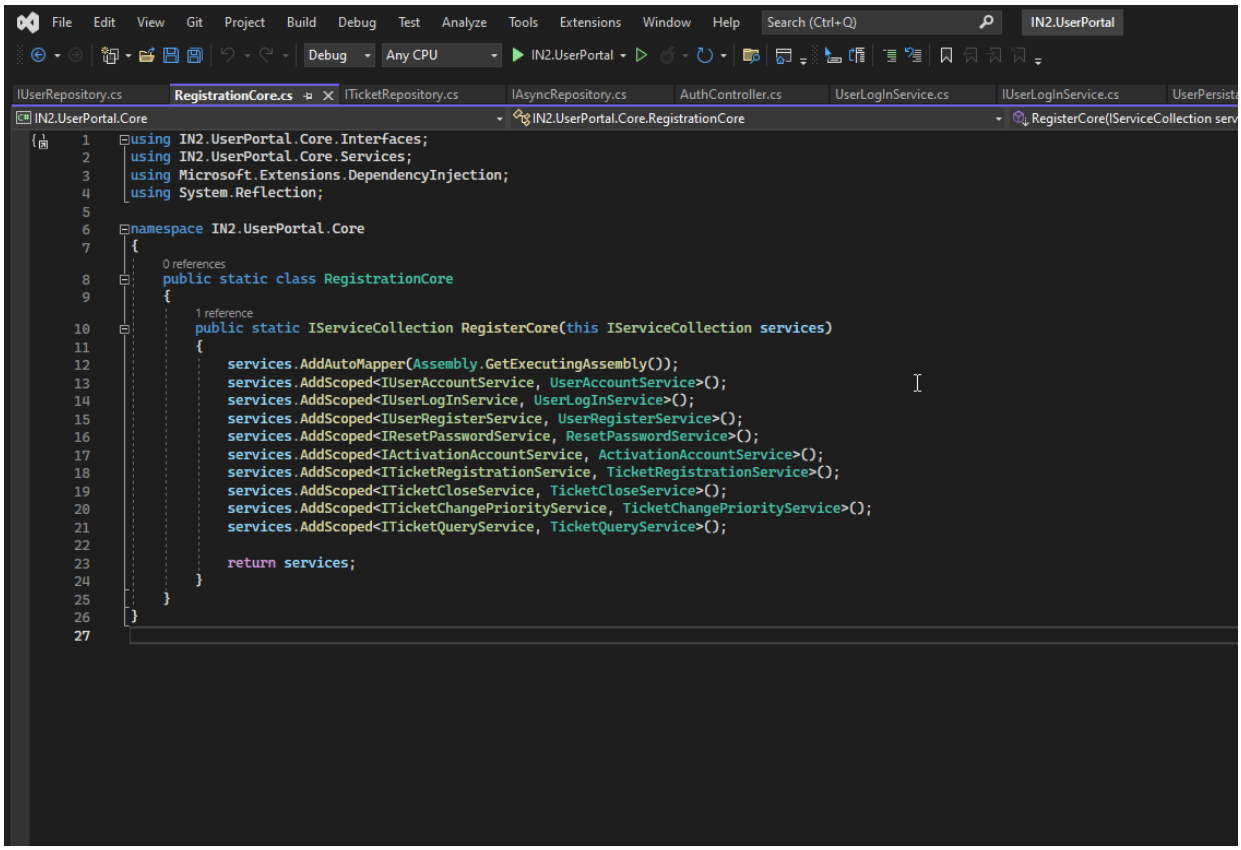
Dependency Injection (DI) je koncept u softverskom inženjeringu koji se koristi za upravljanje ovisnostima između različitih komponenata ili klasa unutar aplikacije. Osnovna ideja DI-a je da se ovisnosti ne stvaraju unutar same klase, već se ubrizgavaju ili predaju izvan klase, obično putem konstruktora, svojstava ili metoda. To olakšava fleksibilno projektiranje aplikacije te omogućava lakše zamjenjivanje komponenata. Klasično, ovisnosti se često stvaraju unutar klase, što može rezultirati čvrsto spojenim komponentama. To otežava testiranje, održavanje i skaliranje aplikacije. *Dependency Injection* rješava taj problem omogućujući da se ovisnosti ubrizgavaju izvana, čime se olakšava ispitivanje i mijenjanje komponenti bez mijenjanja koda koji ih koristi.

Primjer pozivanja *dependency injection* principa iz konstruktora klase *AuthController* je na slici br. 26.



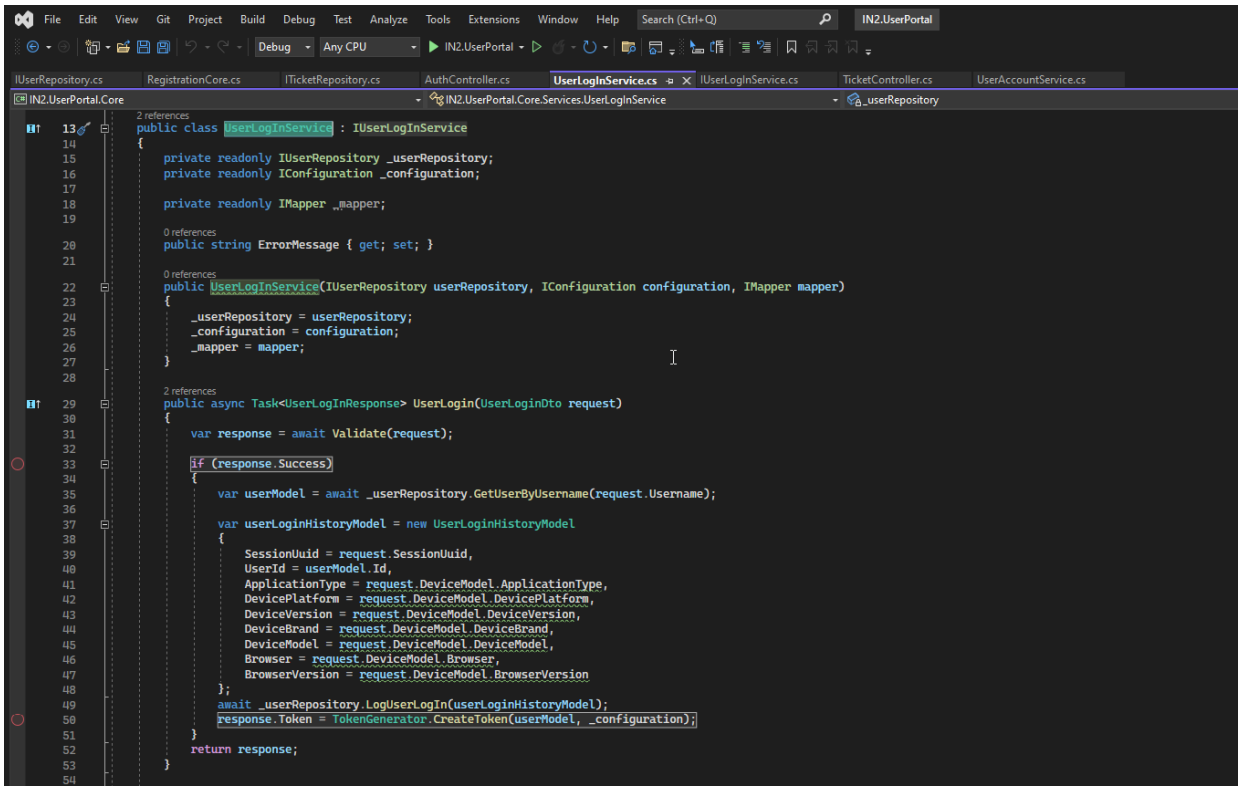
Slika 26. Dependency injection primjer na AuthController klasi

Na slici br. 27., se vidi klasa *„RegistrationCore.cs“* koja služi kao svojevrstan kontejner za *dependency injection* te ga u ovoj klasi konfiguriramo. S metodom „AddScoped<InterfaceServisa, ImeServisa>“ zapravo se dodaje definicija servisa unutar kontejnera.



Slika 27. Dependency injection kontejner

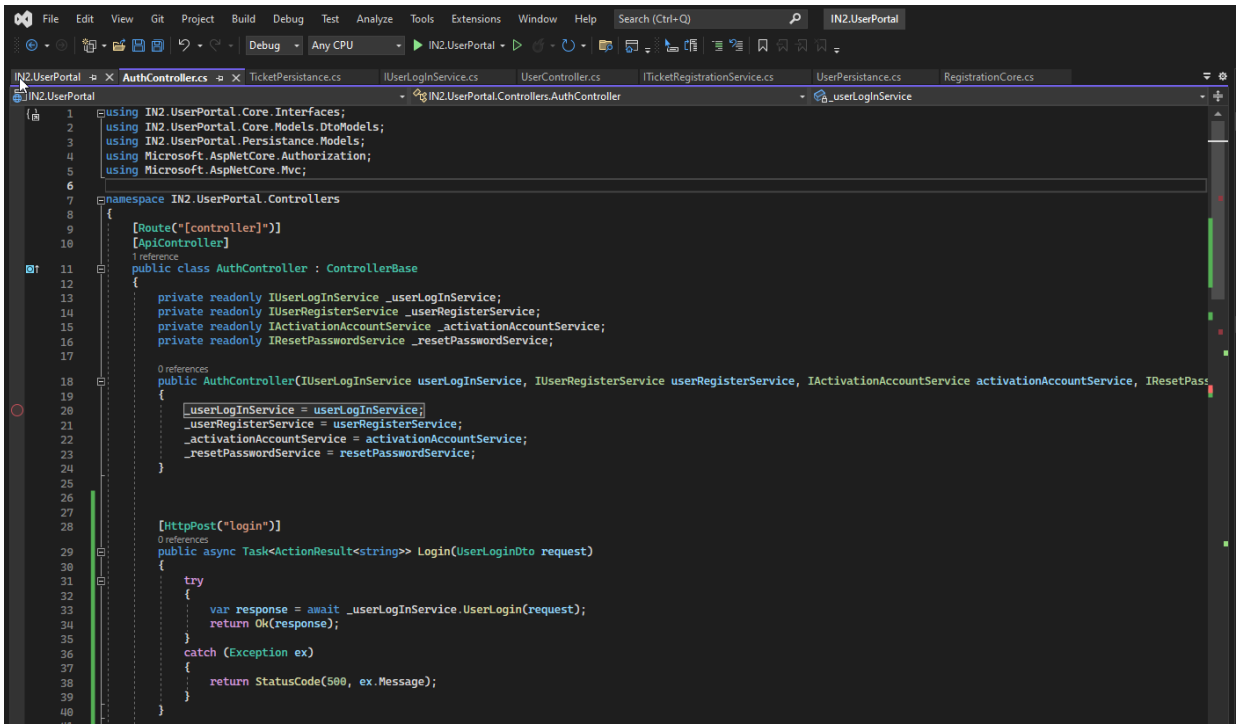
Kada se registriraju servisi *„dependency injection-a“* i pozovu se preko konstruktora kako je gore opisano, pozivaju se metode koje se nalaze unutar registriranog servisa. Klasa „UserLogInService.cs“ u sebi sadržava različite metode za validacije ili nekakve operacije za odrađivanje drugih procesa te poveznicu prema zadnjem dijelu serverske strane, prema „repository-u“. Primjer „UserLogInService.cs“ klase može se vidjeti na slici br. 28.



Slika 28. UserLogInService klasa

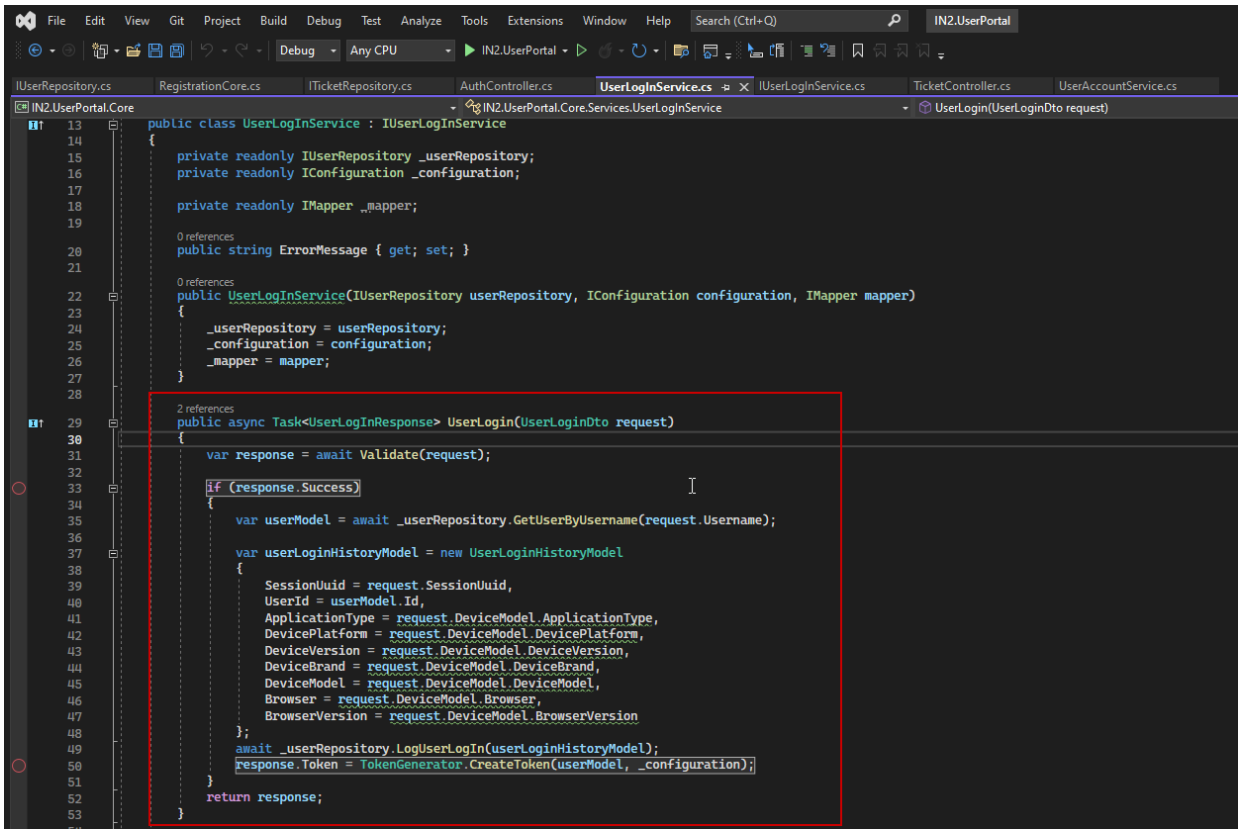
5.2.4. PRIMANJE PODATAKA S FRONTEND-A I SLANJE/ČITANJE PREMA BAZI

Svaka *„Controller“* klasa se nalazi u direktoriju „Controllers“. U klasi *„AuthController“* postoji prikazana metoda *„login“* u koju su poslani podaci sa korisničkog sučelja. Nakon što su poslani podaci sa korisničkog sučelja oni će ući u parametar gore spomenute funkcije a to je *„UserLoginDto“* objekt. Prikaz klase „AuthController“ se može vidjeti na slici br. 29.



Slika 29. Prikaz klase AuthController

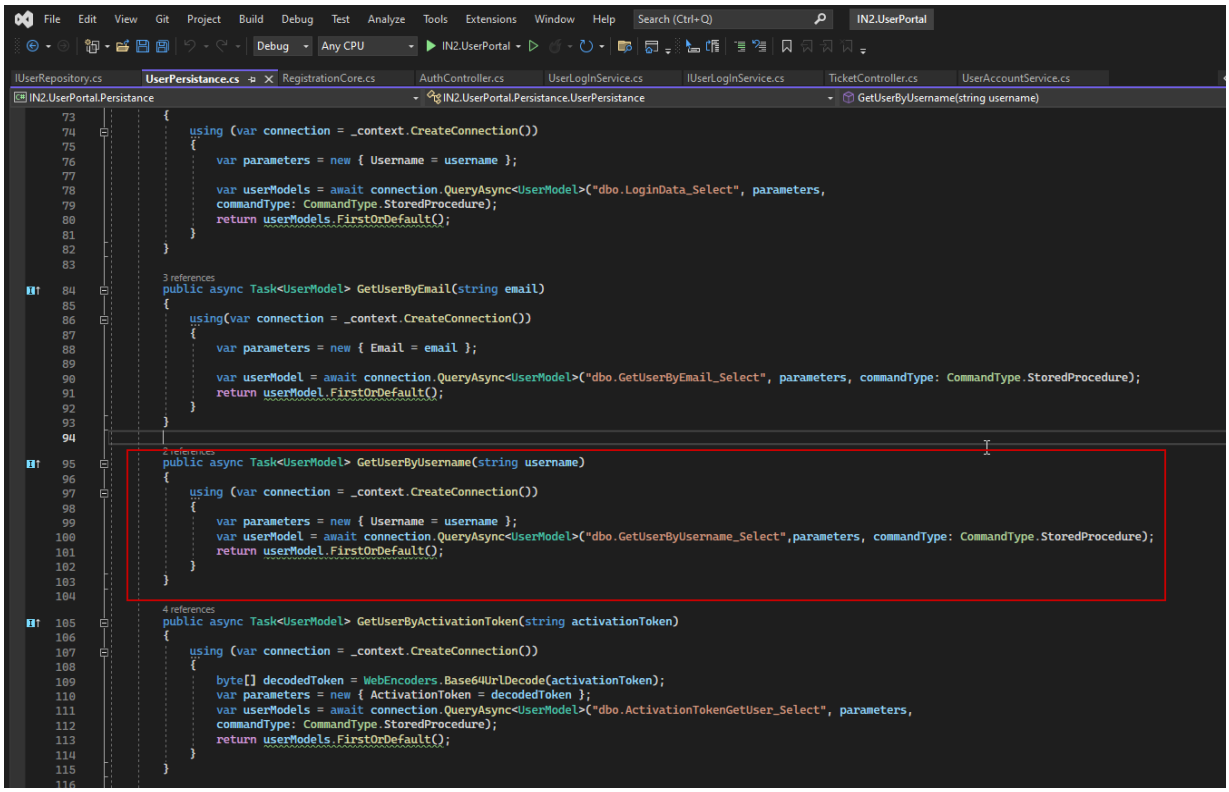
Nakon što su podaci došli u metodu, oni se dalje šalje u servis klasu *„UserLogInService.cs“* na obradu i validaciju, pozivanjem „IUserLoginService“ *interface-a* koji u sebi ima metodu *„UserLogin“*. Opasni koraci se mogu vidjeti na slici br. 30.



Slika 30. UserLogin metoda unutar UserLoginService klase

Metoda *„UserLogin“* će provjeriti postojanost unesenog podatka pozivanjem nove metode *„GetUserByUserName“* koja se nalazi u klasi *„UserPersistance“*. U spomenutu metodu će se predati parametar *request.username* da se preko sql procedure utvrdi ima li zapis u bazi koji ima istu elektroničku poštu kao i ova što je upisana na korisničkom sučelju. Ako redak u tablici ne postoji vratit će se poruka na korisničko sučelje da je unesena elektronička adresa ili lozinka kriva. Ukoliko se elektronička adresa nalazi unutar baze program će dalje ići sekvencijalno te će stvoriti JWT token koji će se vratiti nazad na *cache* memoriju korisnikova uređaja.

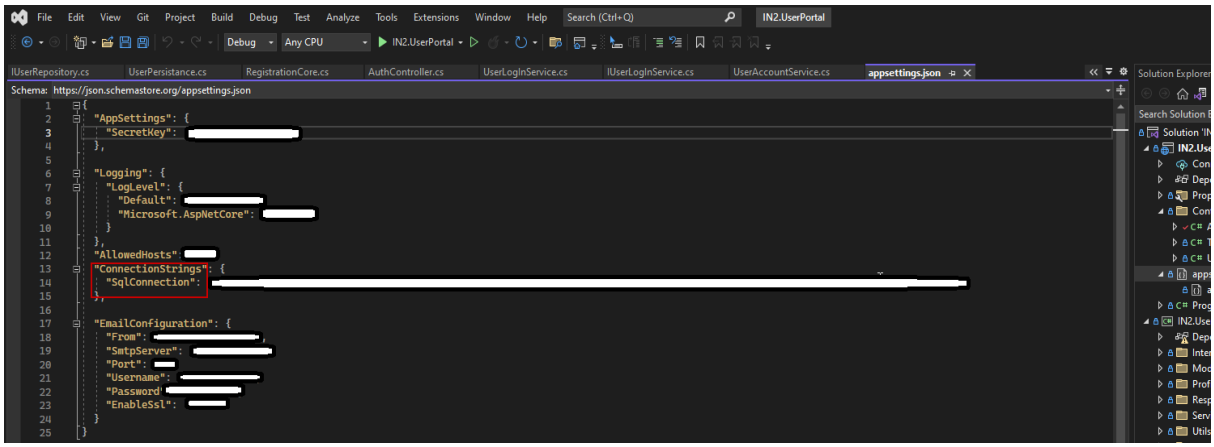
Vrlo bitna stavka je pozivanje spomenute procedure koja će vršiti provjeru na bazi. Procedura ove metode je slijedeća, otvara se komunikacija prema bazi putem ORM biblioteke *Dapper-a*. Nakon toga, predaju se potrebni parametri koje procedura prima, i definira se povratna klasa u koju će se spremati podatkovni set koji će se vratiti iz baze. Opisani tijek rada može se vidjeti na slici br. 31.



```
{
    using (var connection = _context.CreateConnection())
    {
        var parameters = new { Username = username };

        var userModels = await connection.QueryAsync<UserModel>("dbo.LoginData_Select", parameters,
        commandType: CommandType.StoredProcedure);
        return userModels.FirstOrDefault();
    }
}

3 references
public async Task<UserModel> GetUserByEmail(string email)
{
    using(var connection = _context.CreateConnection())
    {
        var parameters = new { Email = email };

        var userModel = await connection.QueryAsync<UserModel>("dbo.GetUserByEmail_Select", parameters, commandType: CommandType.StoredProcedure);
        return userModel.FirstOrDefault();
    }
}

2 references
public async Task<UserModel> GetUserByUsername(string username)
{
    using (var connection = _context.CreateConnection())
    {
        var parameters = new { Username = username };
        var userModel = await connection.QueryAsync<UserModel>("dbo.GetUserByUsername_Select",parameters, commandType: CommandType.StoredProcedure);
        return userModel.FirstOrDefault();
    }
}

4 references
public async Task<UserModel> GetUserByActivationToken(string activationToken)
{
    using (var connection = _context.CreateConnection())
    {
        byte[] decodedToken = WebEncoders.Base64UrlDecode(activationToken);
        var parameters = new { ActivationToken = decodedToken };
        var userModels = await connection.QueryAsync<UserModel>("dbo.ActivationTokenGetUser_Select", parameters,
        commandType: CommandType.StoredProcedure);
        return userModels.FirstOrDefault();
    }
}
```

Slika 31. UserPersistance klasa

Pristupni podaci za bazu podataka prilikom otvaranja potrebne veze iščitavaju se iz datoteke koja se naziva *„AppSettings.json“ u polju „SqlConnection“.* Datoteku možemo vidjeti na slici br. 32.



```
{
  "AppSettings": {
    "SecretKey": [redacted]
  },

  "Logging": {
    "LogLevel": {
      "Default": [redacted]
      "Microsoft.AspNetCore": [redacted]
    }
  },
  "AllowedHosts" [redacted]
  "ConnectionStrings": {
    "SqlConnection": [redacted]
  },

  "EmailConfiguration": {
    "From": [redacted]
    "SmtpServer": [redacted]
    "Port": [redacted]
    "Username": [redacted]
    "Password" [redacted]
    "EnableSsl": [redacted]
  }
}
```

Slika 32. Prikaz datoteke appSettings.json

5.3. IMPLEMENTACIJA BAZE PODATAKA

Baza podataka je treći sloj aplikacijske arhitekture. Kako je gore objašnjeno, tehnologija koja se koristila za bazu podataka je *Microsoft-ov* proizvod SQL Server.

Implementacija baze podataka unutar aplikacije zahtijeva pažljivo planiranje i provedbu kako bi se osigurala pouzdanost, performanse i funkcionalnost sustava. Prvi korak u ovom procesu uključuje instalaciju odgovarajuće verzije SQL Servera, temeljnog sustava za upravljanje bazama podataka, na ciljanom računalu ili poslužitelju. Nakon uspješne instalacije, struktura baze podataka treba biti definirana.

Kreiranje baze podataka postiže se putem SQL naredbi koje se izvršavaju putem alata kao što je SQL Server Management Studio (SSMS). Primarni korak ovog koraka je stvaranje nove baze podataka pomoću SQL naredbe "CREATE DATABASE", gdje se odabire odgovarajuće ime baze podataka. Nakon što je baza podataka stvorena, definiraju se tablice unutar nje. Svaka tablica ima stupce koji specificiraju vrste podataka koje će biti pohranjene i koji određuju strukturu podataka. Ključni elementi tablice uključuju primarni ključ (PRIMARY KEY) za jedinstvenu identifikaciju redaka te indekse za ubrzanje pretraživanja.

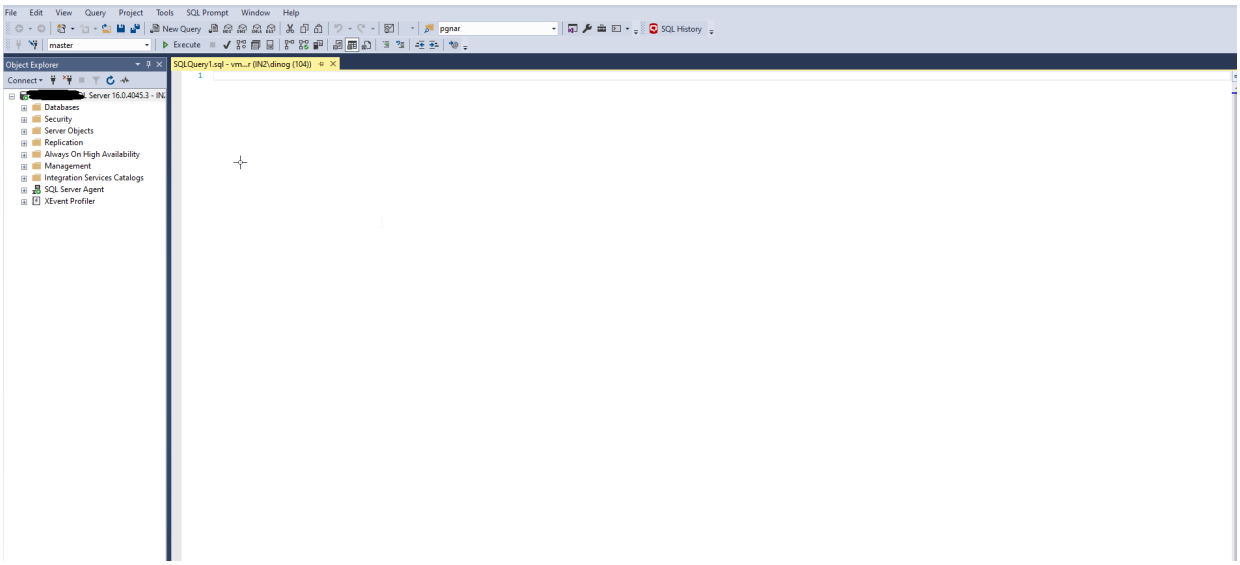
Ako aplikacija zahtijeva povezanost među različitim tablicama, relacije se postižu korištenjem ključeva stranih (FOREIGN KEY) koji referenciraju primarne ključeve drugih tablica. Ovaj korak osigurava dosljednost podataka i integritet baze podataka.

Nakon definiranja strukture baze podataka, razvojni tim može napisati SQL upite koji omogućuju dohvaćanje, dodavanje, ažuriranje i brisanje podataka unutar tablica. Ti upiti su ključni za interakciju aplikacije s bazom podataka te omogućuju pristup podacima na temelju specifičnih kriterija.

Važno je napomenuti da se tijekom implementacije mora obratiti pažnja na optimizaciju performansi. Dodavanje indeksa na često korištene stupce može značajno ubrzati izvođenje upita. Također, osiguravanje sigurnosti i pouzdanosti baze podataka uključuje postavljanje ovlasti pristupa te redovito izrađivanje sigurnosnih kopija radi zaštite od gubitka podataka.

Konačno, integracija baze podataka u aplikaciju zahtijeva implementaciju komunikacijskog sloja koji omogućuje aplikaciji izvršavanje SQL upita ili korištenje ORM

(Object-Relational Mapping) alata koji pojednostavljuju interakciju s bazom podataka kroz objekte. Microsoft-ov alat za uređivanje SQL koda se može vidjeti na slici br. 33.



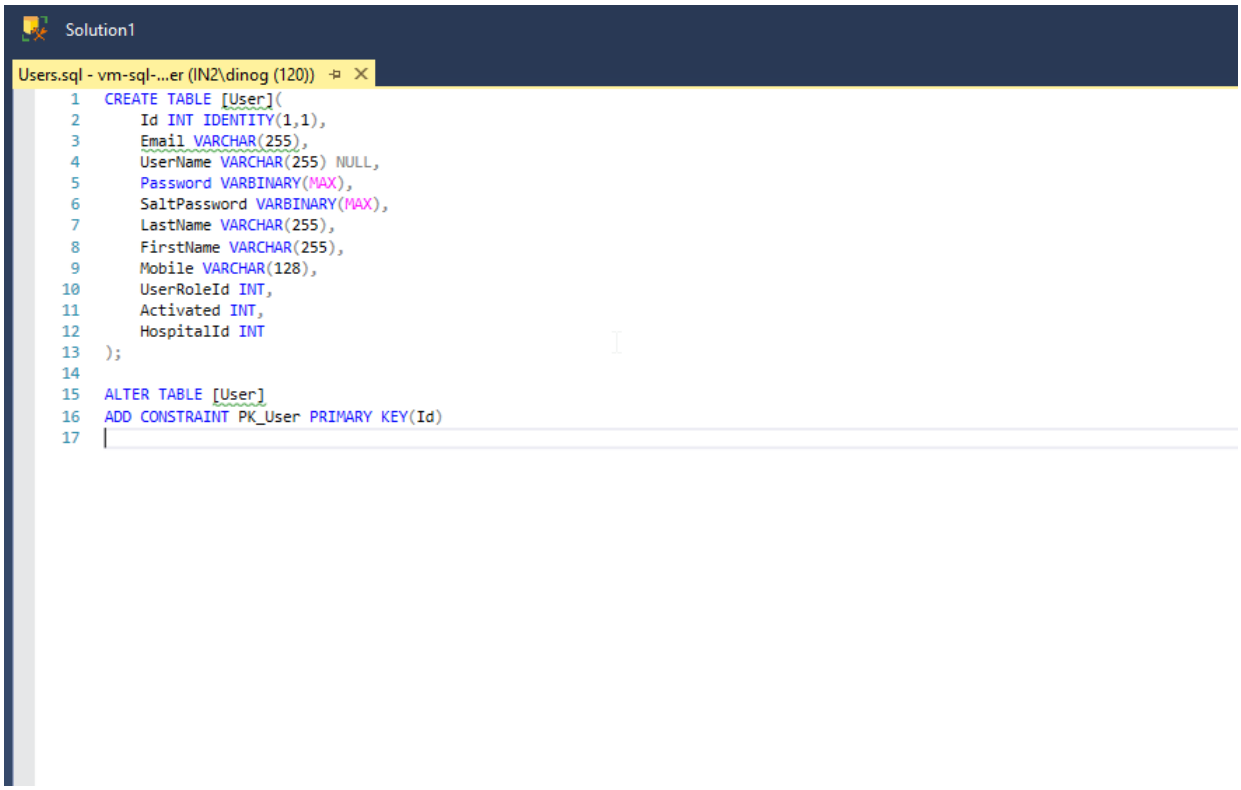
Slika 33. Alat Microsoft SQL Server Managment Studio

5.3.1. STRUKTURA BAZE PODATAKA

Struktura baze podataka predstavlja ključni temelj informacijskog sustava, pružajući strukturu za organiziranje, pohranu i upravljanje podacima. U kontekstu ovog diplomskog rada, pažljivo je oblikovana struktura baze podataka kako bi se podržale ključne funkcionalnosti sustava. Struktura baze uključuje niz komponenti poput tablica, sinonima, procedura i funkcija, koje surađuju kako bi omogućile glatko izvršavanje i upravljanje poslovnim procesima. Struktura datoteka baze podataka se može vidjeti na slici br. 34.

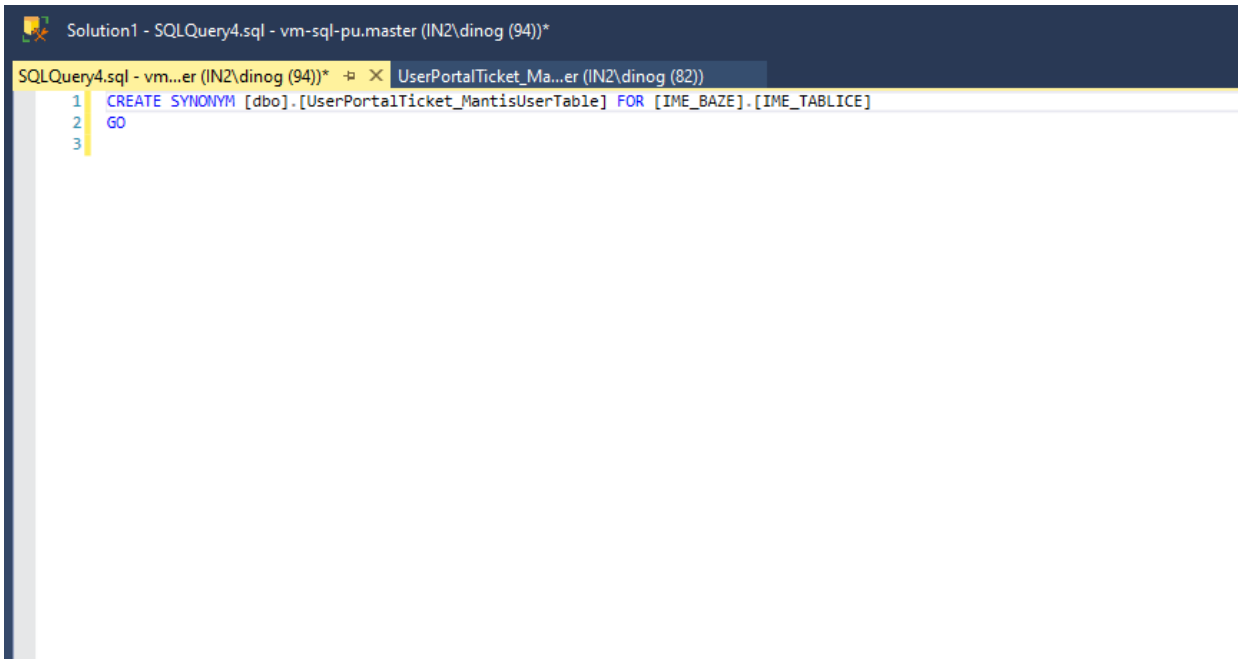
Slika 34. Struktura baze podataka

Temeljni elementi baze podataka su tablice koje služe kao entiteti za pohranu podataka. U ovom slučaju, baza podataka sadrži specifično oblikovane tablice koje odražavaju entitete relevantne za kontekst sustava. Tablice su organizirane s definiranim stupcima koji odražavaju svojstva tih entiteta. Svaka tablica ima svoj primarni ključ, osiguravajući jedinstvenost svakog zapisa. Na slici 35. se vidi definicija tablice od korisnika gdje svaki korisnik ima svoj primarni ključ odnosno ID koji je jedinstven, i još atributa koji se nalaze ispod primarnog ključa.



Slika 35. SQL Server definicija tablice

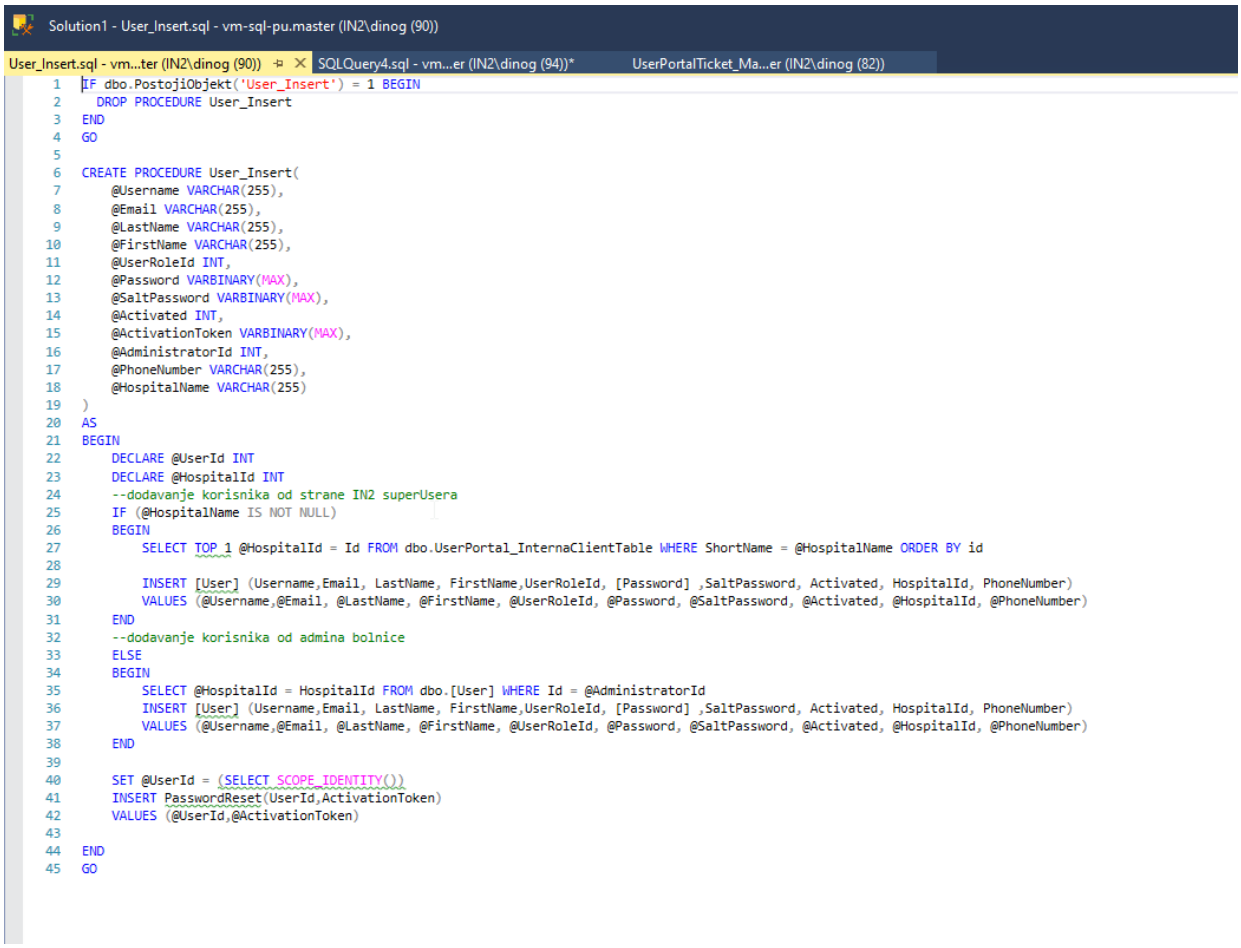
Struktura uključuje sinonime koji omogućuju pristup tablicama iz druge baze podataka, posebice iz *ticketing sustava*. Ovi sinonimi osiguravaju komunikaciju i integraciju između dvije baze podataka, omogućujući pristup podacima na udaljenom sustavu kao da su lokalni. Ova komponenta ključna je za sinkronizaciju informacija između različitih dijelova sustava. Primjer definicije sinonima može se vidjeti na slici 36. Tako da sada prilikom korištenja sinonima *„UserPortalTicket_MantisUserTable“,* koristi se tablica koja se nalazi na nekoj drugoj bazi gdje ovaj sinonim samo predstavlja referencu na nju.



Slika 36. SQL Server definicija sinonima

Procedura predstavlja skup SQL naredbi koje se izvršavaju kako bi se postigli određeni ciljevi. U ovom kontekstu, procedure se koriste za preračune i manipulaciju podacima. Te procedure pružaju alate za izračunavanje složenih vrijednosti ili izvođenje akcija koje zahtijevaju više koraka. Proceduralni pristup dopušta efikasno i strukturirano izvršavanje zadataka unutar baze podataka. Na slici 37. vidimo primjer napisane procedure za dodavanje novog korisnika unutar sustava korisničkog portala gdje na vrhu definiranja procedure vidimo ulazne parametre koji počinju s znakom „@“. Ulazni parametri se ispunjavaju u programskom sučelju i šalju se preko odabranog ORM-a na bazu podataka u ovom slučaju je to *Dapper*. Nakon slanja ulaznih podataka prema bazi, dalje se izvršava algoritam koji je opisan na slici ispod.

Nakon što procedura vrati na serversku stranu neki skup podataka, na temelju tih podataka program zna što treba vratiti na sučelje. Recimo, u primjeru koji je napisan gore kada se vrši prijava korisnika u sustav, pozvat će se procedura da provjeri postoji li korisnik s elektroničkom poštom koja je unesena na korisničkom sučelju već u bazi. Ako postoji, procedura će vratiti neki skup podataka koji će biti dovoljan serverskoj strani da zna koju poruku treba poslati na korisničko sučelje.



Slika 37. SQL Server primjer procedure za unos novog korisnika

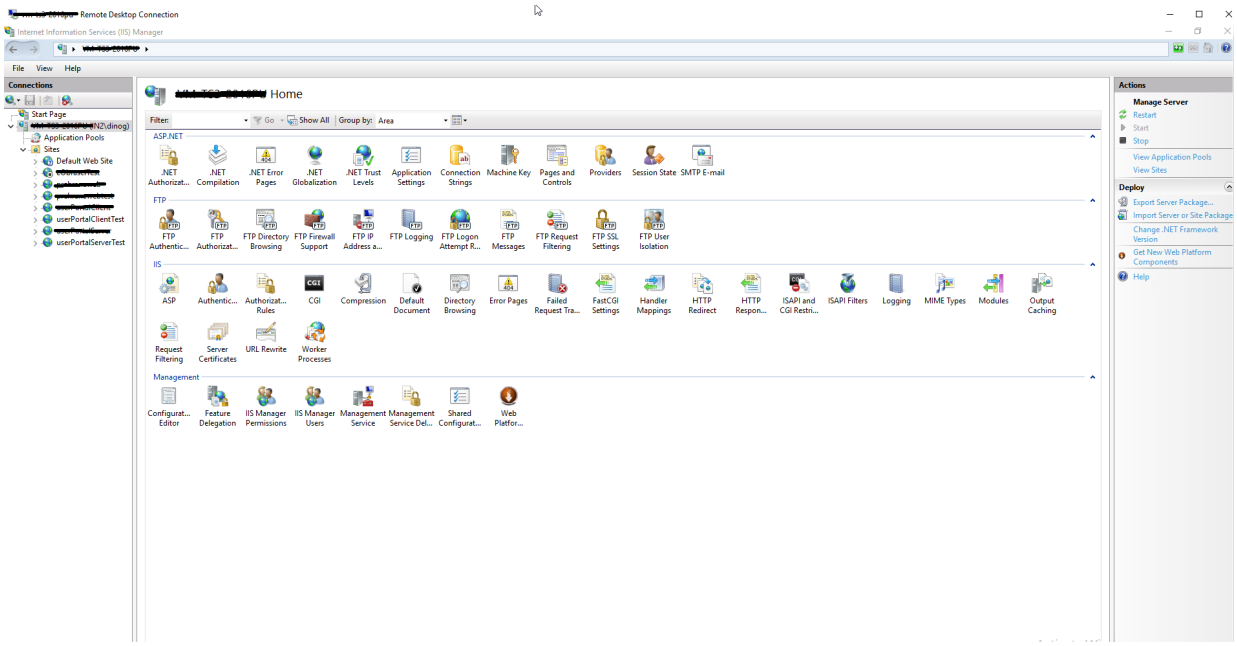
5.4. IMPLEMENTACIJA WEB APLIKACIJE NA POSLUŽITELJU

Unutar ovog diplomskog rada, proučava se postupak implementacije web aplikacije na IIS (eng. Internet Information Services) poslužitelju. Ovaj proces uključuje niz koraka za osiguravanje da se aplikacija može pravilno izvoditi putem internetskog preglednika.

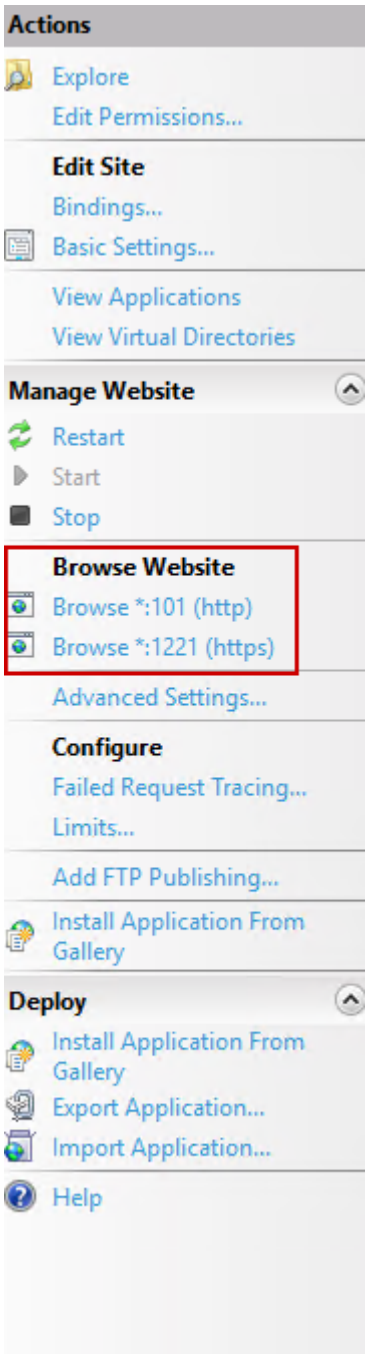
Kao prvi korak, potrebno je pripremiti aplikaciju za postavljanje na poslužitelj. To uključuje završavanje razvoja i testiranja aplikacije, kao i osiguravanje prisutnosti svih potrebnih datoteka. Nadalje, poslužitelj mora imati instaliran IIS. Ako to već nije učinjeno, instalacija IIS-a je nužna kako bi poslužitelj mogao podržati web aplikacije. Nakon instalacije IIS-a, slijede koraci za konfiguraciju aplikacije unutar IIS-a. To uključuje stvaranje mape na poslužitelju za aplikaciju te kopiranje svih datoteka unutar te mape. Nakon toga, konfigurira se web stranica unutar IIS-a pomoću informacija kao što su naziv, fizički put, IP adresa i port preko kojeg će aplikacija biti dostupna.

Kritični korak je testiranje aplikacije na poslužitelju kako bi se provjerilo je li implementacija ispravna. Ovo uključuje otvaranje web preglednika i unosa URL-a aplikacije te provjeru njenog pravilnog izvođenja. Ovisno o potrebama i zahtjevima, također se može implementirati sigurnost, uključujući autentikaciju i autorizaciju, kako bi se zaštitili osjetljivi dijelovi aplikacije. Napredniji koraci, kao što su SSL certifikati za enkripciju i redovito održavanje, također su bitni kako bi se osigurala stabilnost i sigurnost aplikacije na IIS poslužitelju.

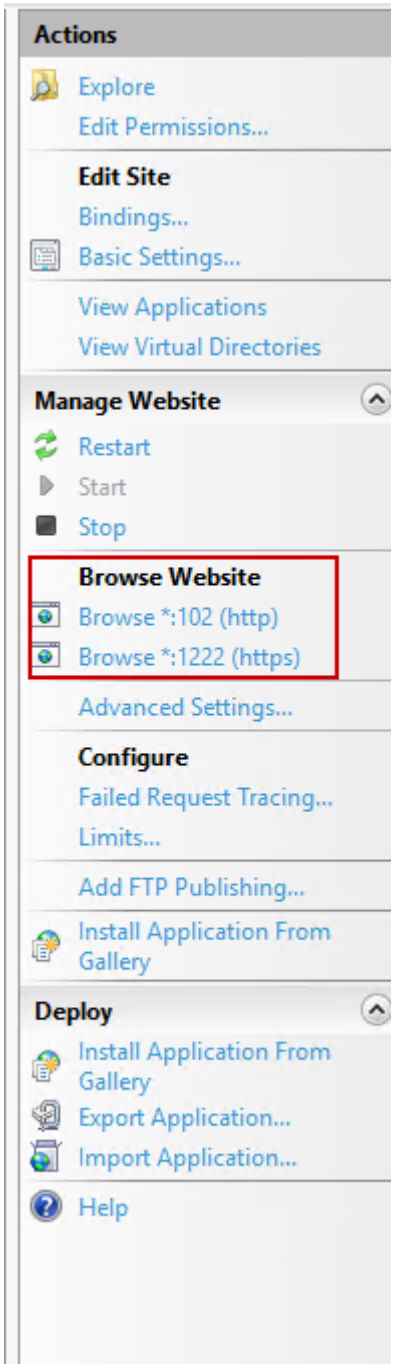
Na slici br. 38. se vidi primjer IIS-a gdje je konfigurirana klijentska i serverska strana aplikacije. Klijentska i serverska strana aplikacije je odvojena, te se svaka nalazi na posebnoj adresi. Primjer toga je prikazan na slikama 39. i 40.



Slika 38. Primjer IIS-a

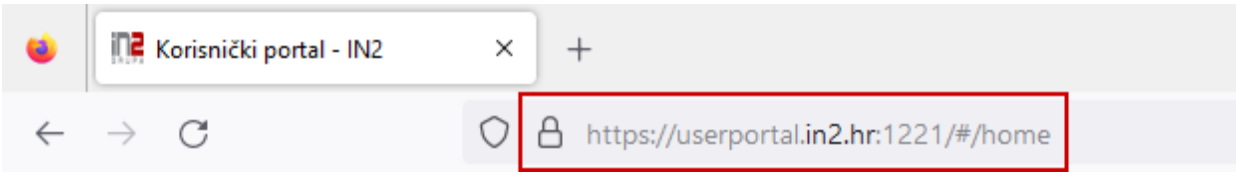


Slika 39. IIS - prikaz portova http i https klijentske strane



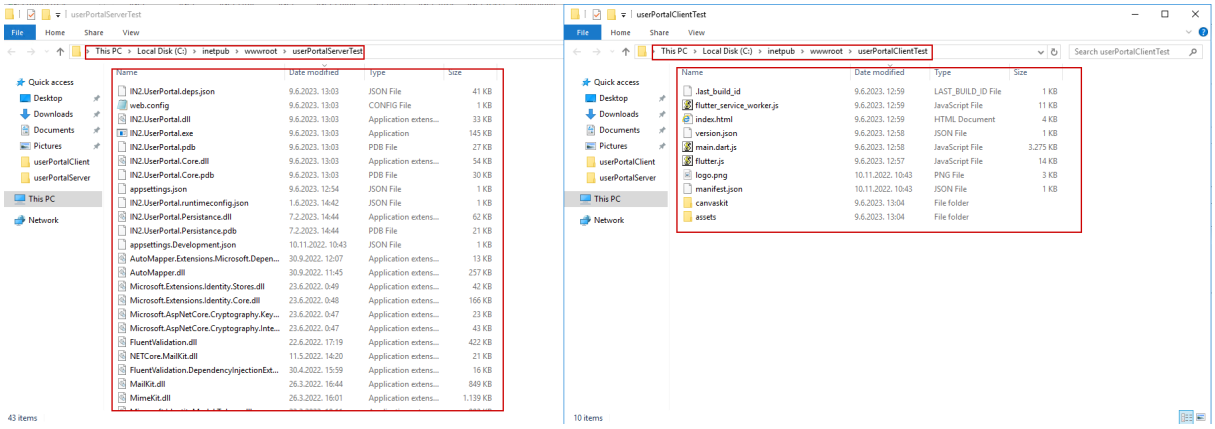
Slika 40. IIS - prikaz portova http i https serverske strane

Ako je sve dobro konfigurirano dobije se konačno rješenje, tako da se može pozvati klijentska strana na web pregledniku koju smo prethodno postavili. Primjer toga je prikazan na slici 41.



Slika 41. Prikaz url aplikacije konfigurirane u IIS-u

Na slici br. 42. je prikazan grafički prikaz datoteka koje pripadaju serverskoj i klijentskoj strani unutar IIS (eng. Internet Information Services) okruženja. Na ovoj ilustraciji jasno se vizualizira organizacija datoteka



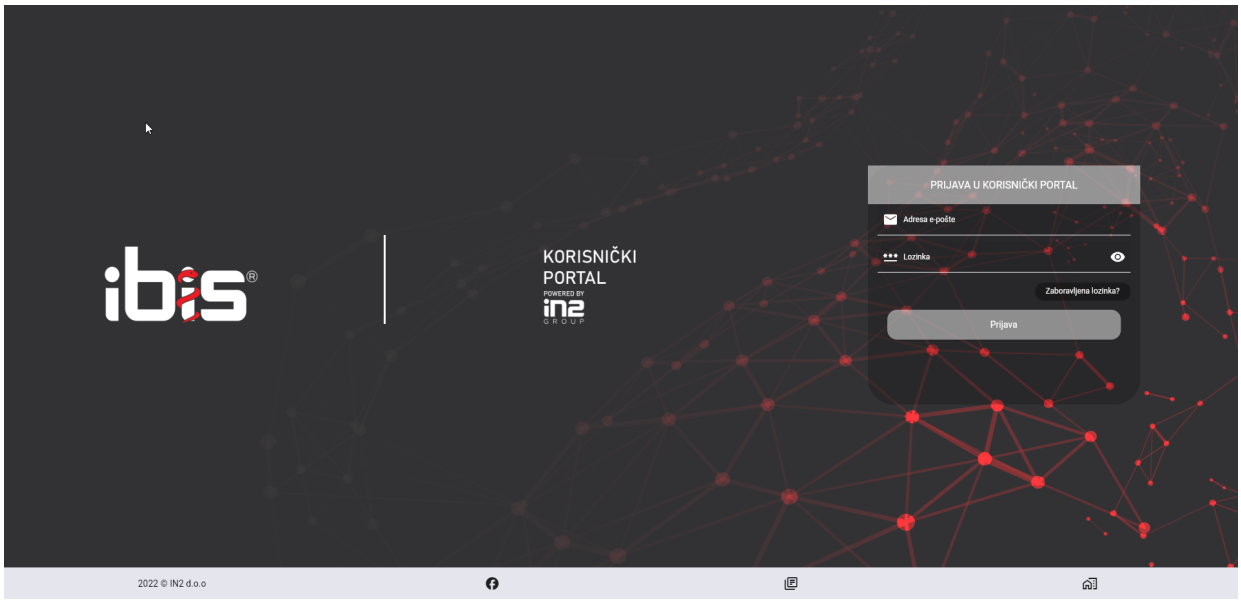
Slika 42. Prikaz datoteka serverske i klijentske strane unutar IIS-a

6. KORISNIČKE UPUTE

U korisničkim uputama koje će biti prikazane postoje tri vrste prikaza ekrana: Super-administratora, administratora (na ustanovi) te običan korisnik (na ustanovi). U ovisnosti o podacima za prijavu, aplikacija će se prebaciti na ekran na koji korisnik ima pravo pregleda.

6.1. PRIJAVA U APLIKACIJU

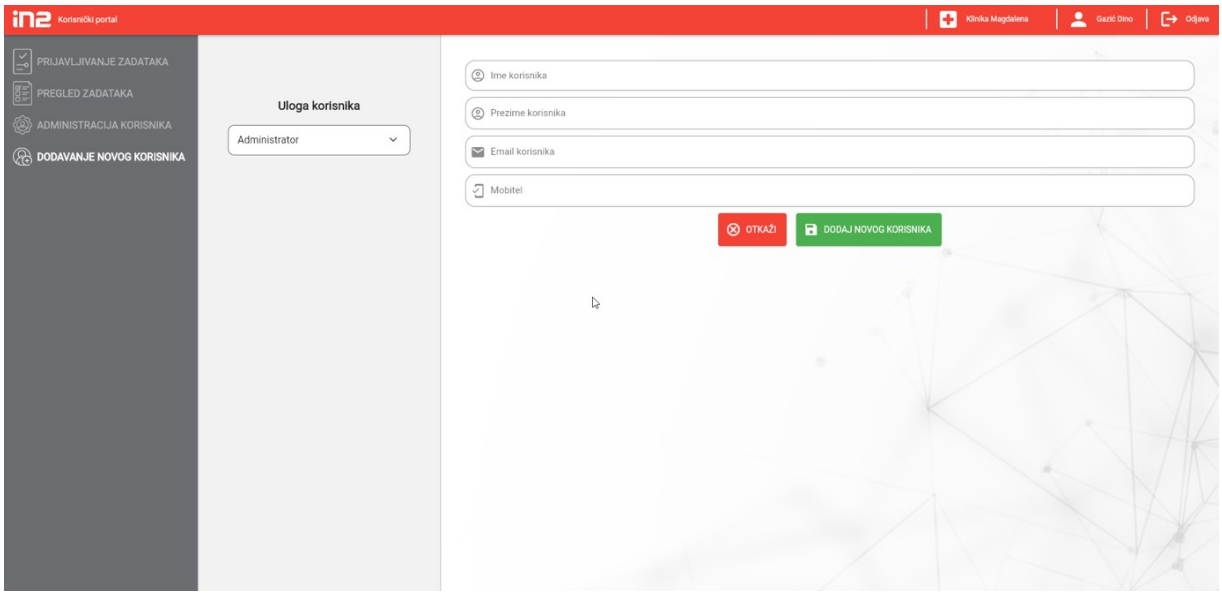
Pri svakom pokretanju aplikacije, ukoliko smo se prethodno odjavili, otvorit će nam se sučelje za prijavu u aplikaciju, što se može vidjeti na slici br. 43. Prijava u aplikaciju je vrlo jednostavna odnosno dosta slična kao i prijava u svaku drugu aplikaciju. Za prijavu u aplikaciju je potrebna email adresa te lozinka koja je postavljena pri registraciji. U aplikaciji postoje tri vrste korisnika kojima se otvara određeno sučelje nakon prijavljivanja u sustav ovisno o ulozi koja mu je postavljena (super-administrator, administrator bolnice, običan korisnik).



Slika 43.Početna stranica aplikacije

Aplikacija je napravljena u smislu da se novi korisnik ne može sam registrirati, nego ga u aplikaciju dodaje korisnik koji je administrator ustanove u ovom slučaju bolnice. Što je u ovom slučaju prikazano na slici br. 44.

Administrator tada upiše ime i prezime, elektroničku poštu te njegov broj telefona. Nakon upisivanja podataka i klikom na gumb „Dodaj novog korisnika“ novom korisniku na e-mail stigne aktivacijski link gdje si sam postavlja lozinku za portal.

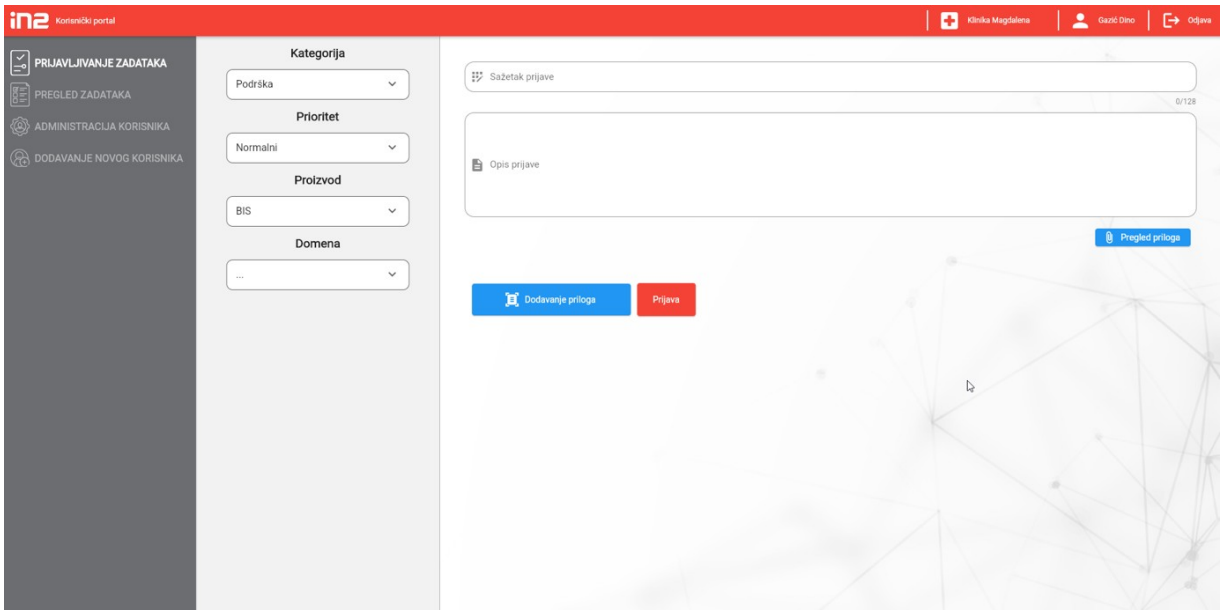


Slika 44. Registracija korisnika

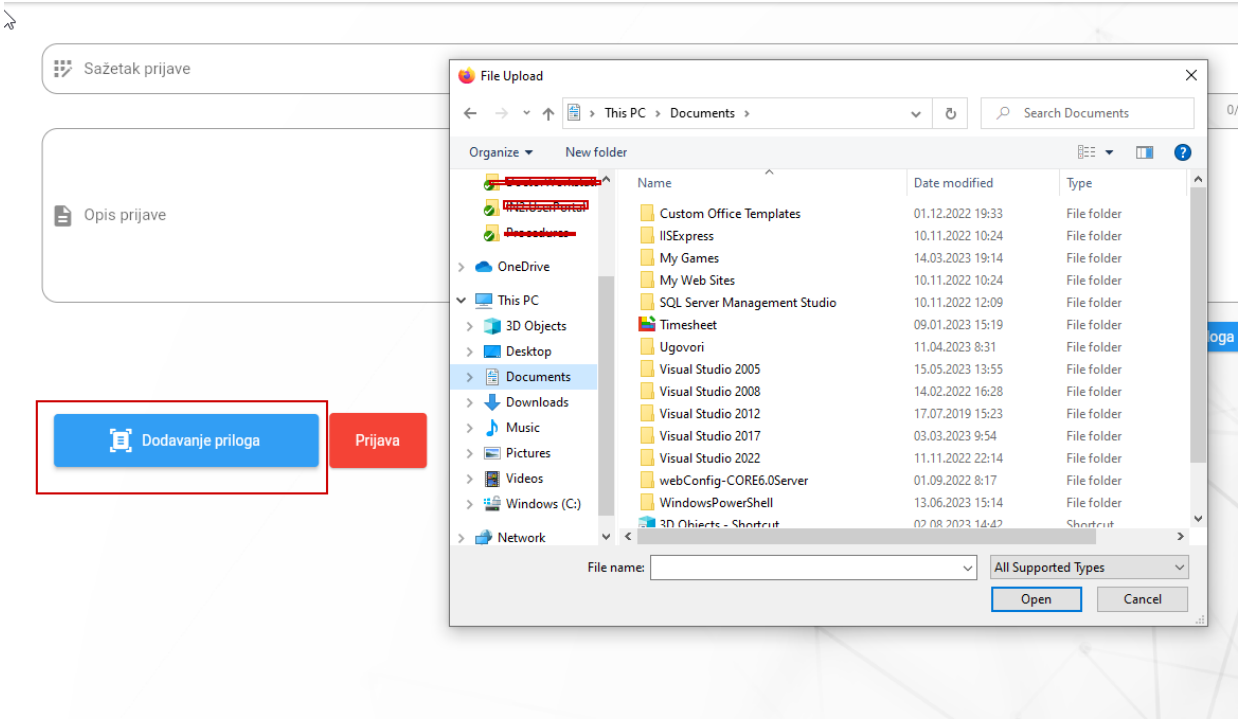
6.2. PRIJAVLJIVANJE ZADATAKA

Na ovom prikazu u slici br. 45. se vidi ekran koji služi za prijavljivanje zadataka, gdje korisnik odabire kategoriju zadatka (greška, podrška, novi zahtjev), također određuje i prioritet zadatka koji prijavljuje, proizvod na kojemu se dogodila greška ili proizvod na kojem žele doraditi nekakve promjene te domena koja je potkategorija proizvoda. Uz odabir nabrojanih stvari, korisnik još mora upisati i „Sažetak prijave“ odnosno naslov prijavka te opis zahtjeva. Vidimo i gumb „Dodaj priloge“ koji služi da se sa uređaja koji koristi aplikaciju odaberu nekakvi prilozi u obliku slike, videa ili nekakvog dokumenta.

Akcija pritiska na gumb „Dodaj priloge“ se vidi na slici br. 46.

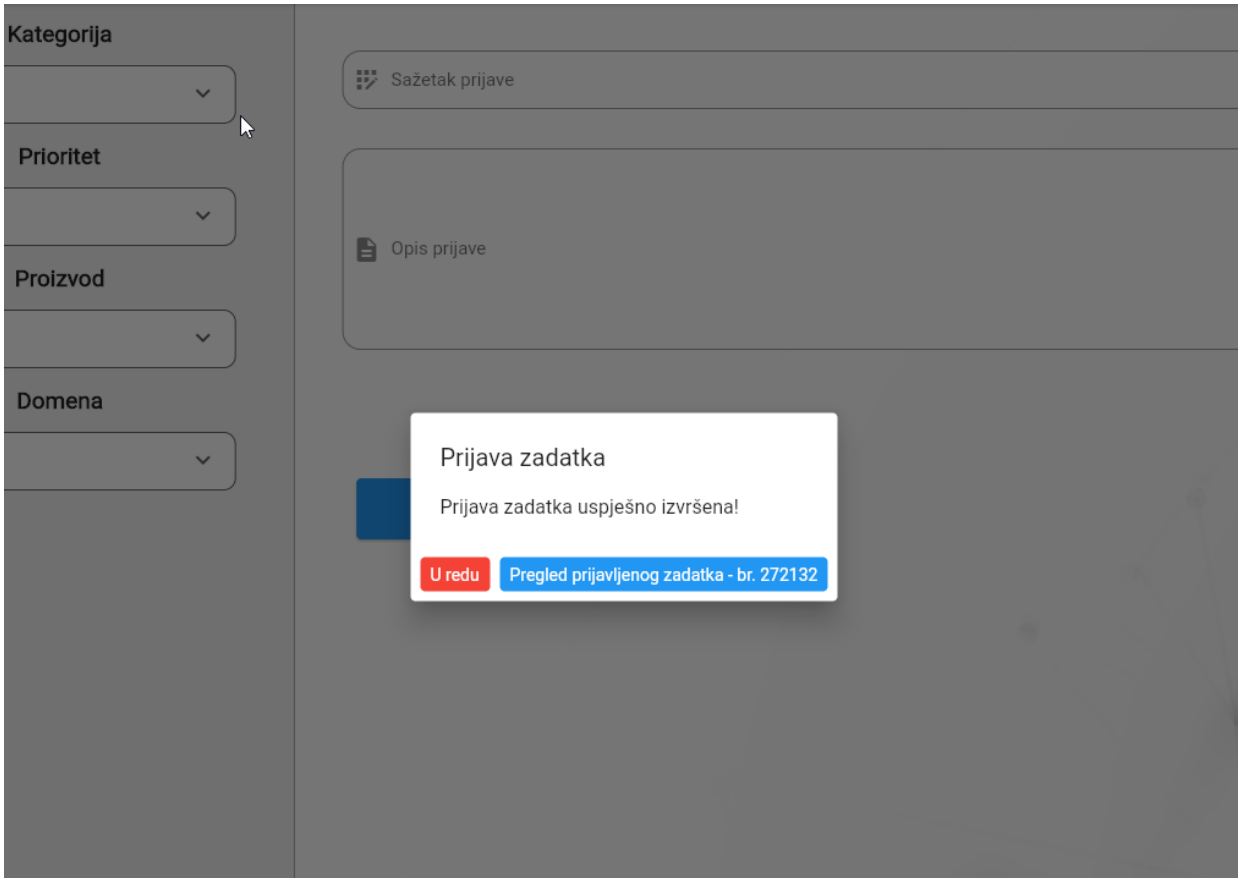


Slika 45. Prijavljivanje novih zadataka



Slika 46. Dodavanje priloga

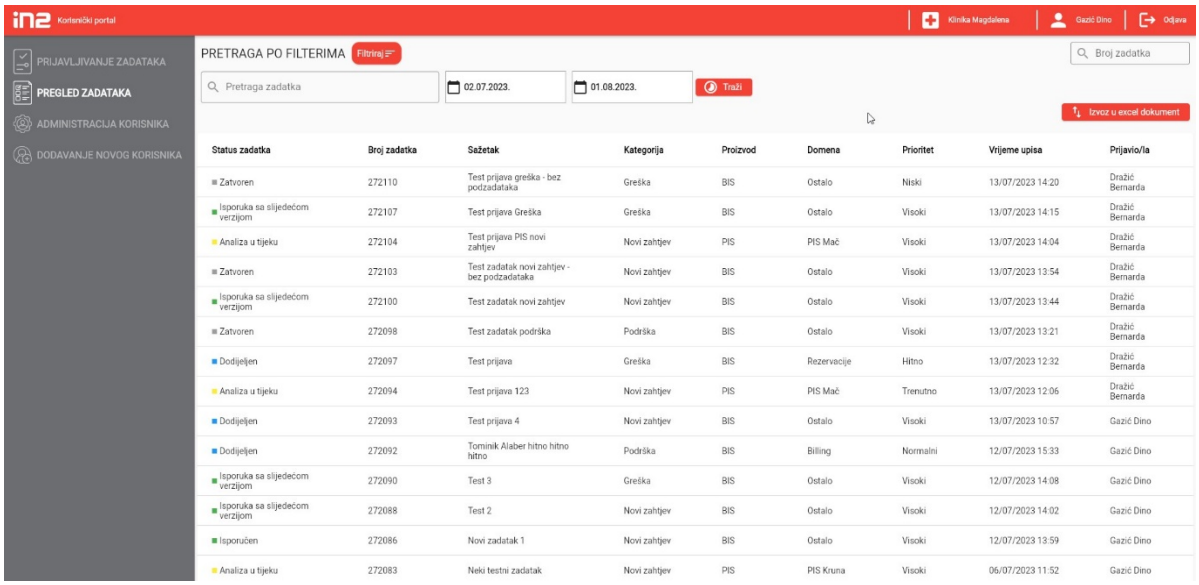
Na slici br. 47. se vidi poruka kada je uspješno prijavljen zadatak tada se može automatski otići na detaljan pregled zadatka ili pritisnuti gumb "U redu".



Slika 47. Poruka uspješnosti prijavljenog zadatka

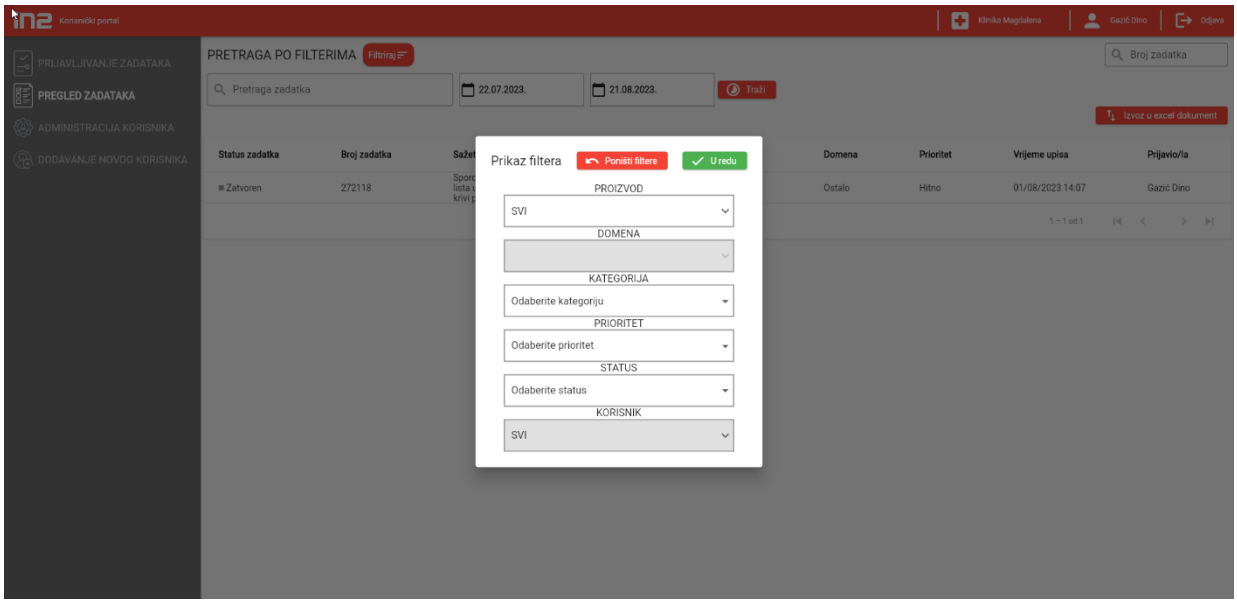
6.3. PREGLED PRIJAVLJENIH ZADATAKA

Na slici br. 48. je prikazana lista svih prijavljenih zadataka na nekoj ustanovi, administrator vidi sve zadatke koji su korisnici prijavili, dok običan korisnik vidi samo svoje zadatke. Na slici dolje se vide sve važne karakteristike prijavljenih zadataka: Status zadatka, broj zadatka, sažetak, kategorija, proizvod, prioritet, vrijeme upisivanja te tko je prijavio zadatak. Čvor „Pregled zadataka“ u navigaciji još omogućava filtriranje zadataka prema filterima koji su prikazani na slici br. 49. na kojoj se vidi filtriranje prema: proizvodu, domeni, kategoriji, prioritetu i statusu. Također, postoje i polja za pretragu prema sažetku zadatka te broju zadatka gdje se automatski odlazi na detalje zadatka koje će biti prikazani malo kasnije. Važna funkcionalnost se nalazi još i u desnom kutu ekrana a to je izvoz zadatka u Microsoft alat „Excel“.



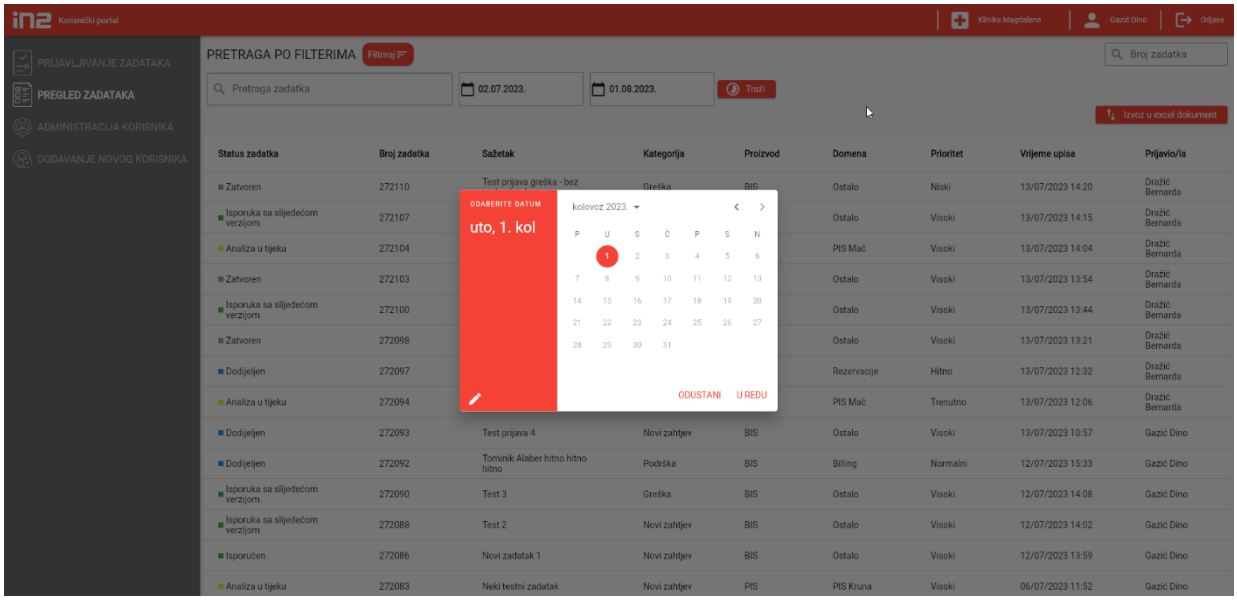
Slika 48. Pregled prijavljenih zadataka

Na slici br. 49. još se može raditi filtriranje prema proizvodu koji je ugovoren sa ustanovom, domenom koja je podtip proizvoda, kategorija, prioritet, status i korisnik koji je prijavio zadatak ali samo korisnik sa administratorove ustanove.



Slika 49. Prikaz filtera nad listom prijavljenih zadataka

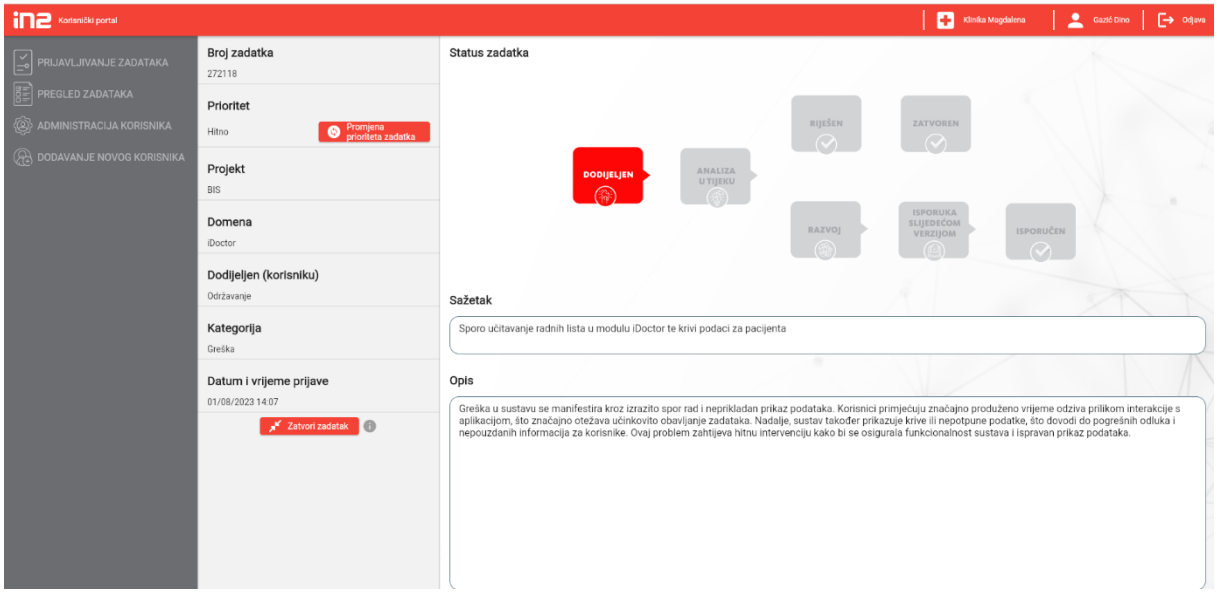
Također, aplikacija ima i filriranje prema datumu koje se otvara klikom na polja gdje pišu datumi te gdje se odabire datum od i datum do. Prikaz tog filtera je na slici br. 50.



Slika 50. Prikaz datumskog filtera nad listom zadataka

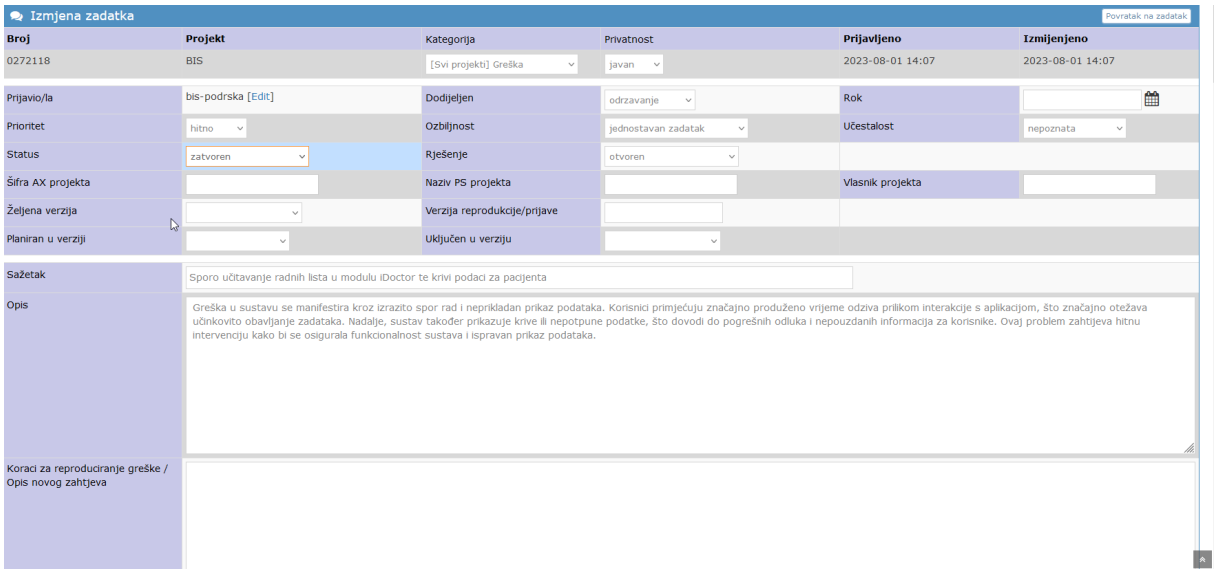
6.4. DETALJAN PREGLED PRIJAVLJENOG ZADATKA

Na slici 51. prikazan je detaljan pregled prijavljenog zadatka gdje su u prvom planu prikazani statusi zadatka tj. u kojem se trenutno statusu nalazi otvoreni zadatak. Također, vide se i opisi naslova prijave i sam opis prijave. Kao mogućnosti upravljanja na ovom ekranu vidi se gumb promjena prioriteta zadatka, ukoliko korisnik koji ga je prijavio odluči promijeniti prioritet zadatka ili ukoliko ga odluči zatvoriti to može učiniti klikom na gumb „Zatvori zadatak“ ali samo isključivo u slučaju dok je još u statusu „Dodijeljen“ takva informacija se prikaže ako korisnik pređe pokazivačem preko informativnog kruga.



Slika 51. Detaljan pregled prijavljenog zadatka

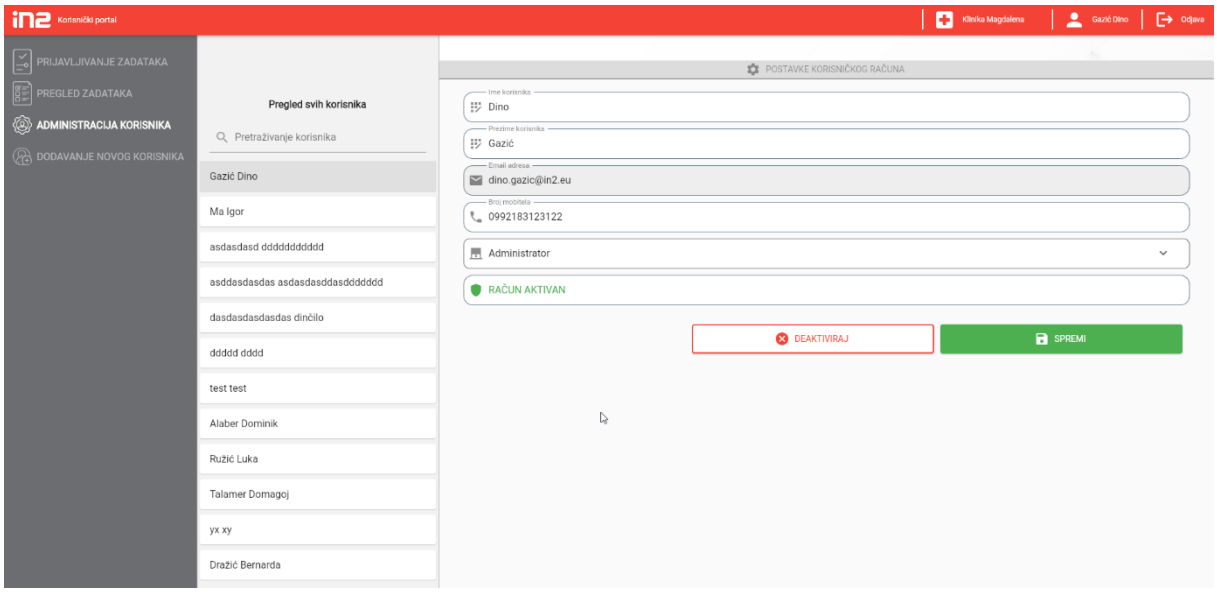
Promjenu statusa zadatka na prethodnoj slici uzrokuje promjena zadatka u ticketing sustavu MantisBT gdje zaposlenik unutar kompanije kada se prijavljeni zadatak riješi, zatvori ili ga pak postavi u neki drugi status to reflektira na graf statusa zadataka koji je prikaz na slici koja je prikazana iznad. Promjenu statusa i nekakvih drugih opcija možemo vidjeti na slici br. 52.



Slika 52. Detaljan pregled zadatka unutar MantisBT ticketing sustava

6.5. ADMINISTRACIJA KORISNIKA

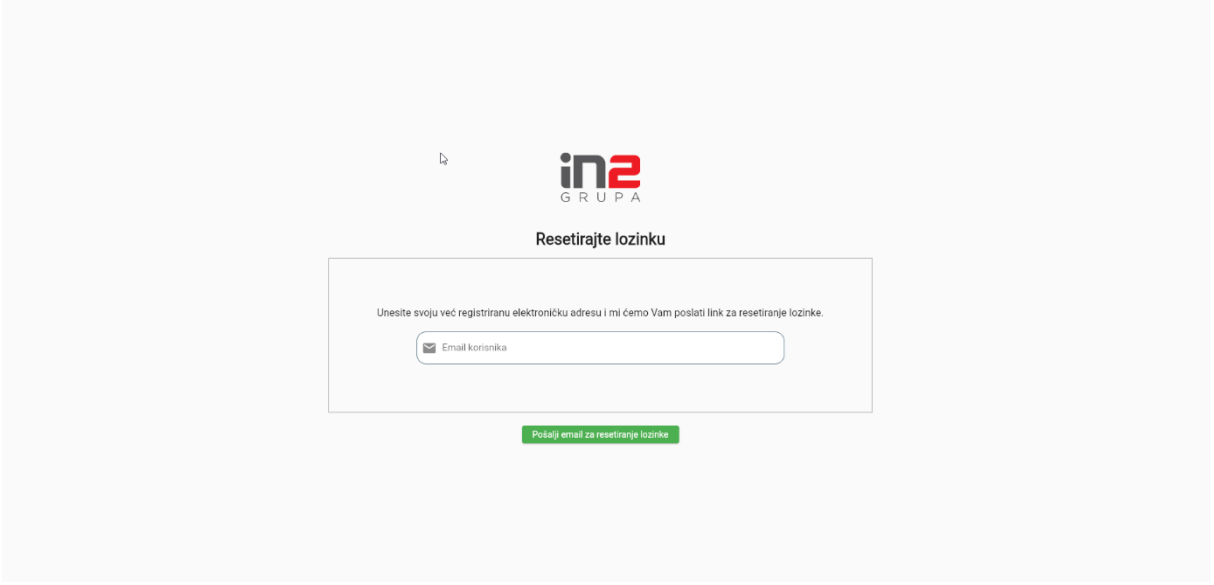
Čvor administracija korisnika je prikazana na slici br. 53. Na ovom prikazu ako je korisnik prijavljen kao administrator bolnice može vidjeti u listi s lijeve strane sve korisnike koji su dodani na ustanovu. Također, može napraviti ažuriranje polja kao što su: ime, prezime, broj telefona te ulogu korisnika. Elektronička adresa se ne može mijenjati jer je unikatna i služi za registraciju i prijavljivanje. Moguće je još i napraviti deaktivaciju ili aktivaciju odabranog korisnika da mu omogućimo ili onemogućimo ponovno prijavljivanje.



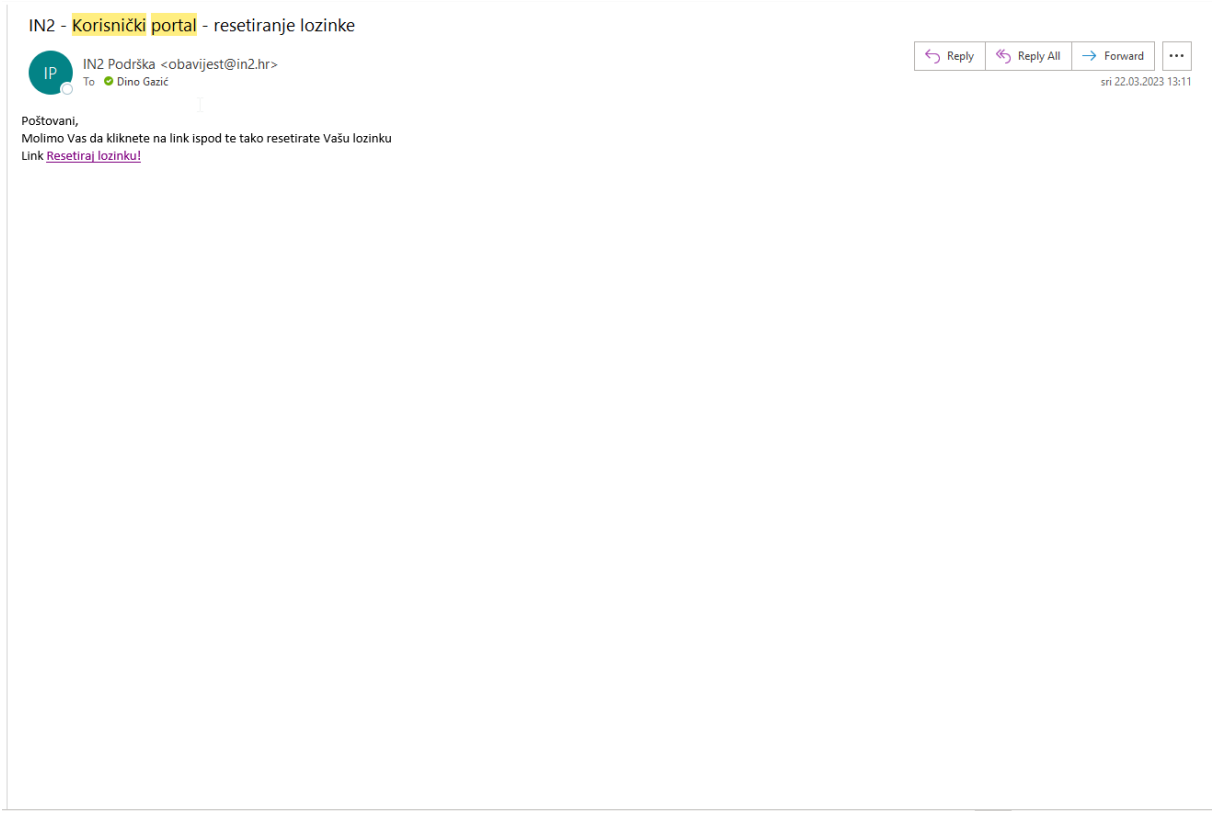
Slika 53. Prikaz administracije korisnika

6.6. PONOVNO POSTAVLJANJE LOZINKE

Ukoliko je korisnik zaboravio lozinku, pritiskom na gumb zaboravljena lozinka će mu se otvoriti ekran gdje može upisati elektroničku adresu, te će mu nakon toga stići elektronička pošta sa linkom gdje može ponovno postaviti lozinku za račun. Ekran prikaza za postavljanje nove lozinke može se vidjeti na slici br. 54., a automatski generirana elektronička pošta koja pristiže u pretinac je vidljiva na slici br. 55.

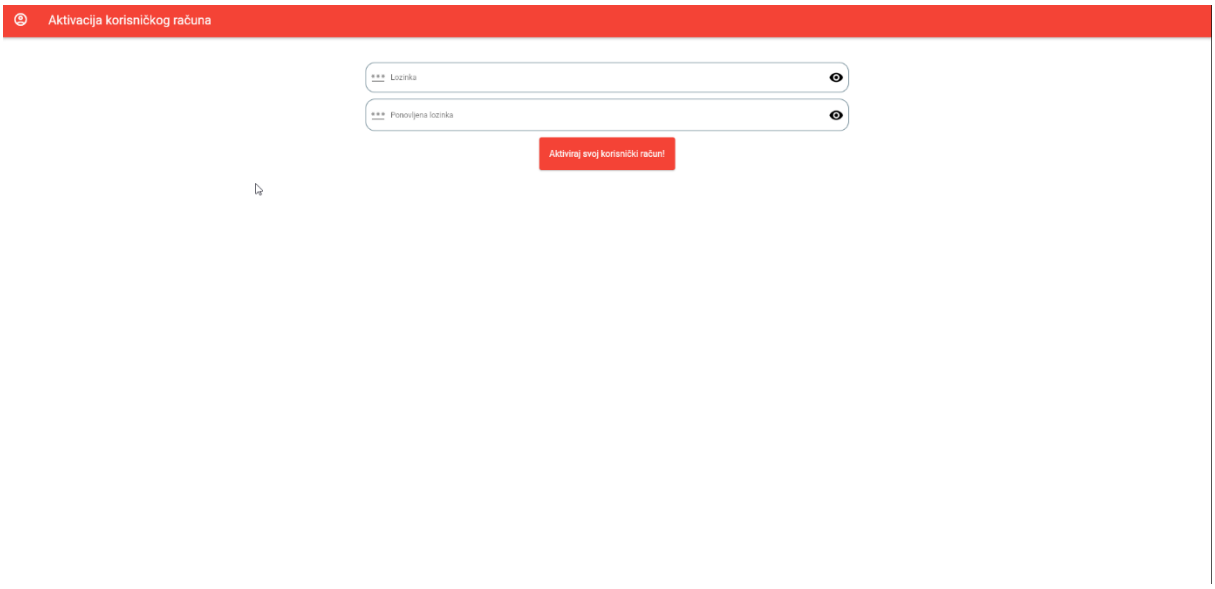


Slika 54. Prikaz ekrana resetiranje lozinke



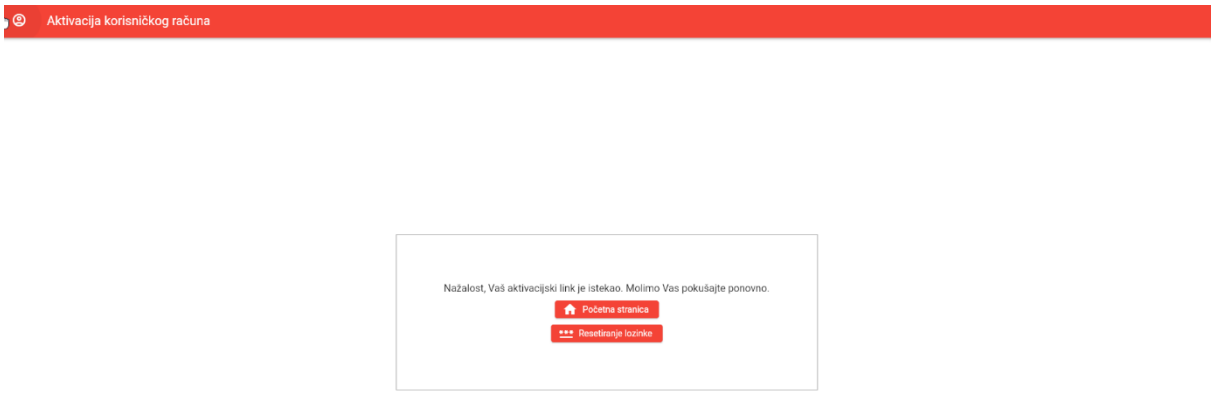
Slika 55. Elektronička pošta prilikom resetiranja lozinke

Nakon što korisnik pritisne na poveznicu resetiraj lozinku, proces će ga odvesti natrag na aplikaciju gdje će moći ponovno upisati lozinku. Što se može vidjeti na slici br. 56.



Slika 56. Prikaz ekrana ponovno postavljanje lozinke

Ukoliko istekne validacijska poveznica, ili poveznica dođe u ruke pogrešne osobe, a korisnik je već jednom iskoristio poveznicu tako što je postavio već jednom novu lozinku, automatski će prilikom slijedećeg ulaska na tu poveznicu doći slijedeći ekran koji se vidi na slici br. 57. koja govori da je aktivacijska poveznica istekla te da se pokuša napraviti proces ponovno.



Slika 57. Prikaz ekrana kada je validacijski link istekao

7. ZAKLJUČAK

U okviru diplomskog rada, postavljeni ciljevi su uspješno ostvareni kroz razvoj visoko funkcionalne web aplikacije za kontinuirano praćenje statusa zadataka, prijavljivanje istih te administraciju korisnika. Ova aplikacija postaje ključni most između korisnika i unutarnjeg ticketing sustava tvrtke, omogućujući brzo i efikasno rješavanje prijavljenih problema.

Arhitektura aplikacije oslanja se na tehnološki temeljit pristup. Za bazu podataka korišten je SQL Server, gdje je koncipirana nova baza s potrebnim entitetima, funkcijama i procedurama. Razvoj je izveden pomoću Microsoft Visual Studio Code i Microsoft Visual Studio alata za programiranje. Programski kod je strukturiran u backend i frontend segmente. Backend, razvijen kao web API u .NET Core 6.0 tehnologiji, pruža rute i kontrolere za interakciju s bazom podataka, osiguravajući učinkovitu manipulaciju podacima. Backend je povezan s bazom podataka putem connection stringa. Korisničko sučelje je implementirano koristeći Flutter programski okvir i Dart jezik, čime je osiguran suvremen i atraktivan korisnički interfejs.

Rezultat je aplikacija koja se uspješno implementirala u više od trideset hrvatskih bolnica i ima više od četiristo korisnika koje ju svakodnevno koriste. Kroz ovaj uspjeh, aplikacija je postala ključna podrška za upravljanje zadacima u bolnicama, omogućavajući brzo rješavanje problema i unaprjeđenje ukupne operativne efikasnosti. Ovaj projekt posjeduje nekoliko prednosti: efikasnost u upravljanju zadacima, jasno definirana uloga kao most između korisnika i ticketing sustava, tehnološki temeljit pristup razvoju i fleksibilnost kroz DI (eng. Dependency Injection) princip.

Međutim, ograničenja se mogu primijetiti u zahtjevima tehničke ekspertize za implementaciju, te potrebnom vremenu korisnika za prilagodbu novoj platformi. Što se tiče budućnosti, ovoj aplikaciji se otvaraju mnoge mogućnosti. Dodatno poboljšanje korisničkog iskustva, implementacija novih značajki i širenje na druge sektore ili industrije predstavljaju potencijalne putove za daljnji razvoj. Također, kontinuirano osiguravanje sigurnosti podataka i prilagodba brzim tehnološkim promjenama će biti ključni faktori u osiguranju relevantnosti i uspjeha aplikacije u budućnosti.

8. LITERATURA

1. AKTUALNE FLUTTER APLIKACIJE. (s.d.). Intelivita. https://www.intelivita.com/blog/apps-made-with-flutter [Pristupio: 15.07.2023.]
2. TEHNOLOGIJE ALTERNATIVNE FLUTTER-U. (s.d.). SPEC INDIA. https://www.spec-india.com/blog/flutter-alternatives [Pristupio: 15.07.2023.]
3. Flutter dokumentacija. (s.d.). Flutter. https://docs.flutter.dev/ [Pristupio: 20.07.2023.]
4. .NET CORE 6.0. (s.d.). DotNetTricks. https://www.dotnettricks.com/learn/aspnetcore/what-is-new-in-aspnet-core- 6.0 [Pristupio: 15.07.2023.]
5. Microsoft SQL Server. (s.d.). SQL Server Tutorial. https://www.sqlservertutorial.net/ [Pristupio: 15.07.2023.]
6. MantisBT dokumentacija. (s.d.). MantisBT. https://www.mantisbt.org/documentation.php [Pristupio: 25.07.2023.]
7. Postman dokumentacija. (s.d.). Postman. https://www.postman.com/company/about-postman/ [Pristupio: 25.07.2023.]
8. Internet Information Services. (s.d.). TechTarget. https://www.techtarget.com/searchwindowsserver/definition/IIS [Pristupio: 25.07.2023.]
9. Postavljanje razvojnog okruženja za .NET Core 6.0. (s.d.). Jason Watmore's Blog. https://jasonwatmore.com/post/2020/05/26/aspnet-core-setup-development-environment [Pristupio: 28.07.2023.]
10. Dijagram slučajeva korištenja. (s.d.). What is use case diagram? https://www.techtarget.com/searchsoftwarequality/definition/use-case [Pristupio 05.09.2023.]
11. Sekvencijalni dijagram. (s.d.). MindOnMap. https://www.mindonmap.com/hr/blog/flowchart-maker/ [Pristupio: 01.08.2023.]
12. Klasni dijagram. (s.d.). Repozitorij Sveučilišta Jurja Dobrile u Puli. https://repozitorij.unipu.hr/islandora/object/unipu%3A2301/datastream/PDF/view [Pristupio: 01.08.2023.]

9. POPIS SLIKA

10. SAŽETAK

Ovaj diplomski rad je strukturiran u dva osnovna dijela: teorijski i praktični segment razvoja web aplikacije za upravljanje procesom prijave grešaka. U početnom dijelu rada, detaljno su opisane tehnologije i alati korišteni tokom izgradnje same aplikacije. Nadalje, pruženi su ilustrativni dijagrami koji približavaju način funkcioniranja aplikacije. Ovaj teorijski dio kulminira SWOT analizom, omogućavajući nam sveobuhvatan uvid u interne prednosti i nedostatke aplikacije, kao i vanjske prilike i prepreke s kojima se aplikacija suočava. U nastavku, usredotočujemo se na praktičnu stranu rada - prezentirana je implementacija web aplikacije, što je srž praktičnog segmenta ovog diplomskog istraživanja. Konačno, rad kulminira s korisničkim uputama koje su osmišljene kako bi olakšale korisnicima upotrebu same aplikacije. Aplikacija je uspješno implementirana na više od trideset hrvatskih bolnica te ima više od 400 korisnika koji ju koriste svakodnevno.

Ključni pojmovi: diplomski rad, web aplikacija, sustav za prijavu problema (ticketing), Flutter/Dart, .NET, C#, SQL Server.

ABSTRACT

This master's thesis is structured into two main parts: the theoretical and practical segments of the development of a web application for managing the error reporting process. In the initial section of the thesis, technologies and tools used during the construction of the application are described in detail. Furthermore, illustrative diagrams are provided to elucidate the functioning of the application. This theoretical section culminates with a SWOT analysis, providing us with a comprehensive insight into the internal strengths and weaknesses of the application, as well as external opportunities and challenges that the application encounters. Moving forward, we focus on the practical aspect of the work - the implementation of the web application is presented, which constitutes the core of the practical segment of this master's research. Finally, the thesis concludes with user instructions designed to facilitate users' utilization of the application. Application is succesfully implemented on more than thirty Croatian hospitals and have more than four hundred satisfied users.

Key terms: master's thesis, web application, issue reporting system (ticketing), Flutter/Dart, .NET, C#, SQL Server.

PRILOZI

Github poveznice do pojedinačno svake komponente projekta (frontend, backend i baza).

Frontend – https://github.com/dgazic/userportalFRONT

Backend – https://github.com/dgazic/userportalBACK

Baza podataka – https://github.com/dgazic/userportalDB