Web aplikacija za fakturiranje usluga

Petrović, Ilija

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

WEB APLIKACIJA ZA FAKTURIRANJE USLUGA

Diplomski rad

Ilija Petrović

Osijek, 2023.

SADRŽAJ

1. UVOD

Web aplikacije za fakturiranje usluga kombiniraju ekonomsku i programsku stručnost kako bi pružile organizacijama učinkovito i precizno rješenje za obračun i izdavanje faktura. Ovaj diplomski rad ima cilj razviti web aplikaciju za fakturiranje usluga koristeći Angular i .NET tehnologije. Aplikacija će omogućiti registraciju i prijavu korisnika, dodavanje partnera te kreiranje i pregled faktura.

Tradicionalni pristupi fakturiranju usluga pokazali su se sporima, pronevjerenima i sklonima greškama. Stoga se sve više organizacija okreće web aplikacijama koje pružaju brže i preciznije rješenje za upravljanje financijama. Web aplikacija za fakturiranje usluga može omogućiti izradu detaljnih računa za usluge.

Aplikacija može omogućiti korisnicima automatizirano, jednostavno i intuitivno rješenje za fakturiranje i upravljanje financijama. Aplikacija će osigurati sigurnost, skalabilnost i performanse u skladu s poslovnim zahtjevima i industrijskim standardima. Namijenjena je i posebno prilagođena tvrtkama iz IT sektora koje obavljaju usluge za druge tvrtke.

Implementacija aplikacije će biti opisana, a evaluacija će provjeriti njenu funkcionalnost. Ovaj rad će pružiti smjernice za daljnje istraživanje i razvoj u ovom području kako bi se unaprijedili procesi fakturiranja usluga za organizacije diljem svijeta.

Aplikacija će pružiti korisnicima mogućnost upravljanja podacima, uključujući dohvaćanje, stvaranje, brisanje i izmjenu podataka.

Poseban fokus će biti na olakšavanju procesa fakturiranja, posebno u slučajevima kada se iste ili slične usluge ponavljaju s redovitom frekvencijom. Na primjer, fakturiranje naplate redovitog održavanja servera koje se naplaćuje mjesečno. Osim toga, aplikacija će pružiti mogućnost unosa informacija o partnerima i uslugama, te će omogućiti korisnicima kreiranje novih faktura i pregledavanje prethodno izdanih faktura. Na ovaj način osigurava se kreiranje baze potrebnih informacija koje su po potrebi lako dostupne, što osigurava i sljedivost obavljenog posla.

Za razvoj će se primijeniti Code-First pristup, što znači da će se prvo definirati modeli podataka te će se na temelju njih generirati shema baze podataka. Ovaj pristup omogućava brži i agilniji razvoj te olakšava održavanje i izmjene modela podataka.

Uz razvoj same aplikacije, rad će također naglasiti i razlike između različitih HTTP (engl. *HyperText Transfer Protocol*) metoda (GET, POST, PUT, DELETE) koji se koriste za

komunikaciju između klijentskog i poslužiteljskog dijela aplikacije. Također, rad će istražiti i istaknuti ključne prednosti Angular okvira kao suvremenog alata za izradu korisničkog sučelja, uključujući modularnost, brzinu razvoja, i sposobnost za responzivne i atraktivne korisničke interakcije.

Rad je organiziran na sljedeći način: u drugom poglavlju pružit će se pregled trenutnog stanja i problema s tradicionalnim pristupima fakturiranju usluga. Treće poglavlje će detaljnije istražiti relevantne ekonomske koncepte i principe vezane uz fakturiranje usluga. Četvrto poglavlje će se usredotočiti na programski aspekt analizirajući tehnologije i arhitekture koje će biti korištene za razvoj web aplikacije za fakturiranje usluga. U petom poglavlju bit će opisan dizajn i implementacija aplikacije.

2. POTREBA ZA RAZVOJEM WEB APLIKACIJE ZA FAKTURIRANJE USLUGA

Kako bi lakše razumjeli potrebu za kreiranjem novog rješenja te ga lakše realizirali potrebno je sagledati i analizirati nedostatke načina rada bez njega. Tako će prvo kroz ovo poglavlje biti ukazano na nedostatke tradicionalnih pristupa fakturiranju usluga te važnost automatiziranja fakturiranja.

Nakon toga će biti navedena i opisana postojeća rješenja automatiziranja fakturiranja te će biti opisana značajnost novog rješenja naspram postojećih.

2.1. Nedostatci tradicionalnih pristupa fakturiranju usluga

Nedostatci tradicionalnog pristupa fakturiranja usluga u odnosu na moderni programski su sljedeći [1]:

- Tradicionalni pristup fakturiranju usluga je često ručni i zahtijeva puno vremena i truda.
- Uspostavljanje i održavanje tradicionalnog sustava fakturiranja može biti skupo.
- Tradicionalni pristup fakturiranju usluga može biti manje precizan i podložan pogreškama.
- Tradicionalni pristup fakturiranju usluga može biti manje fleksibilan i teže se prilagođava promjenama u poslovanju.

S druge strane, programski pristup fakturiranju usluga ima sljedeće prednosti [1]:

- Programski pristup fakturiranju usluga je često automatiziran i zahtijeva manje vremena i truda.
- Programski pristup fakturiranju usluga može biti precizniji i manje podložan pogreškama.
- Programski pristup fakturiranju usluga može biti fleksibilniji i lakše se prilagođava promjenama u poslovanju.

Potreba za modernizacijom je motivirana brzim tehnološkim razvojem i promjenjivim zahtjevima tržišta i kupaca. Modernizacija omogućuje poboljšanje učinkovitosti, sigurnosti, kvalitete i konkurentnosti poslovanja. Web aplikacije su jedan od načina modernizacije koji nude brojne prednosti, kao što su:

- Pristup s bilo kojeg uređaja i lokacije putem interneta

- Jednostavnije održavanje i ažuriranje
- Veća sigurnost i zaštita podataka
- Bolje korisničko iskustvo i interakciju
- Veća skalabilnost i fleksibilnost

2.2 Automatizacija fakturiranja

Automatizacija procesa fakturiranja donosi brojne prednosti organizacijama, uključujući povećanu efikasnost, smanjenje grešaka i optimizaciju vremena i resursa [2]. Evo kako izgleda tipičan proces automatizacije fakturiranja:

1. Unos podataka: Umjesto ručnog unosa podataka za svaku fakturu, automatizacija omogućuje unos podataka na više načina. To može uključivati učitavanje podataka iz CRM sustava, baze podataka ili čak korištenje optičkog prepoznavanja znakova (OCR) za izdvajanje podataka s papirnatih dokumenata.

2. Izračunavanje i generiranje faktura: Na temelju unesenih podataka, automatizacija može automatski izračunati iznose, primijeniti poreze i popuste te generirati fakturu u željenom formatu. Ovo uključuje i pridruživanje relevantnih informacija o partneru kao što su adresa, porezni identifikacijski broj i uvjeti plaćanja.

3. Slanje faktura: Automatizacija omogućuje slanje faktura putem elektroničke pošte ili integraciju s drugim sustavima za elektroničko slanje računa (EDI). To smanjuje vrijeme i troškove vezane uz ispisivanje, slanje poštom i praćenje dostave.

4. Praćenje i upravljanje: Automatizacija omogućuje praćenje statusa faktura, uključujući informacije o isporuci, plaćanju i dospijeću. Također omogućuje automatsko slanje podsjetnika o plaćanju ili generiranje izvještaja o financijskom stanju.

5. Integracija s financijskim sustavima: Automatizacija omogućuje integraciju s financijskim sustavima poput računovodstvenih softvera ili Enterprise resource planning (ERP) sustava. To omogućuje automatsko knjiženje plaćanja, evidentiranje transakcija i generiranje financijskih izvještaja.

6. Arhiviranje dokumenata: Automatizacija omogućuje digitalno arhiviranje i organiziranje faktura kako bi se osigurala njihova dostupnost i lakša pretraga u budućnosti. To pomaže u smanjenju papirnatog otpada, poboljšava sigurnost i olakšava usklađenost s regulatornim zahtjevima.

2.3 Uloga web aplikacija u ubrzanju financijskih transakcija

Prema [1] web aplikacije igraju ključnu ulogu u ubrzanju financijskih transakcija na nekoliko načina:

1. Brža izrada i isporuka faktura: Web aplikacije omogućuju korisnicima brzo kreiranje i generiranje faktura putem intuitivnih sučelja. To smanjuje vrijeme potrebno za ručni unos podataka i izradu faktura te omogućuje trenutačno slanje faktura partnerima putem elektroničke pošte ili drugih elektroničkih kanala.

2. Automatsko knjiženje plaćanja: Integracija web aplikacija s financijskim sustavima omogućuje automatsko knjiženje plaćanja koja se vezuju za fakturirane usluge. Ovo eliminira potrebu za ručnim unosom podataka i ubrzava proces evidentiranja plaćanja u financijskom sustavu.

3. Praćenje statusa faktura u stvarnom vremenu: Web aplikacije pružaju korisnicima mogućnost praćenja statusa faktura u stvarnom vremenu. To znači da korisnici mogu vidjeti kada je faktura isporučena, pročitana ili plaćena. Ovo ubrzava proces komunikacije i smanjuje potrebu za pisanjem i slanjem fizičkih podsjetnika o plaćanju.

4. Ubrzanje procesa odobravanja faktura: Web aplikacije omogućuju korisnicima da elektronički odobravaju fakture putem sustava za upravljanje radnim procesima. To eliminira potrebu za fizičkim prijenosom i odobravanjem papirnatih dokumenata te smanjuje vrijeme potrebno za cirkulaciju faktura između odjela ili različitih odobravatelja.

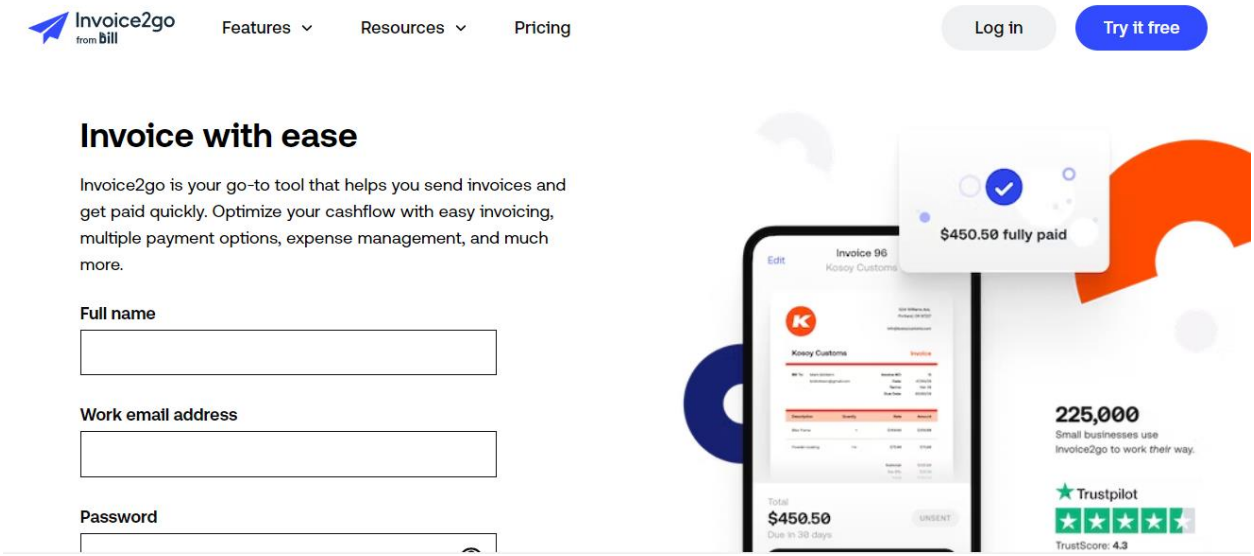
5. Automatizacija plaćanja: Web aplikacije omogućuju integraciju s različitim sustavima za elektroničko plaćanje, kao što su sustavi za online bankarstvo ili platni procesori. To omogućuje korisnicima brzo i sigurno izvršenje plaćanja na temelju primljenih faktura, čime se ubrzava proces naplate i smanjuje vrijeme između izdavanja fakture i primanja plaćanja.

Kombinacija ovih faktora omogućuje web aplikacijama za fakturiranje usluga značajno ubrzavanje financijskih transakcija, skraćuju vrijeme između izdavanja fakture i primanja plaćanja te poboljšavaju likvidnost organizacija. Osim toga, smanjuje se mogućnost grešaka i povećava transparentnost u procesu fakturiranja i plaćanja.

2.4 Primjeri web aplikacija za fakturiranje usluga

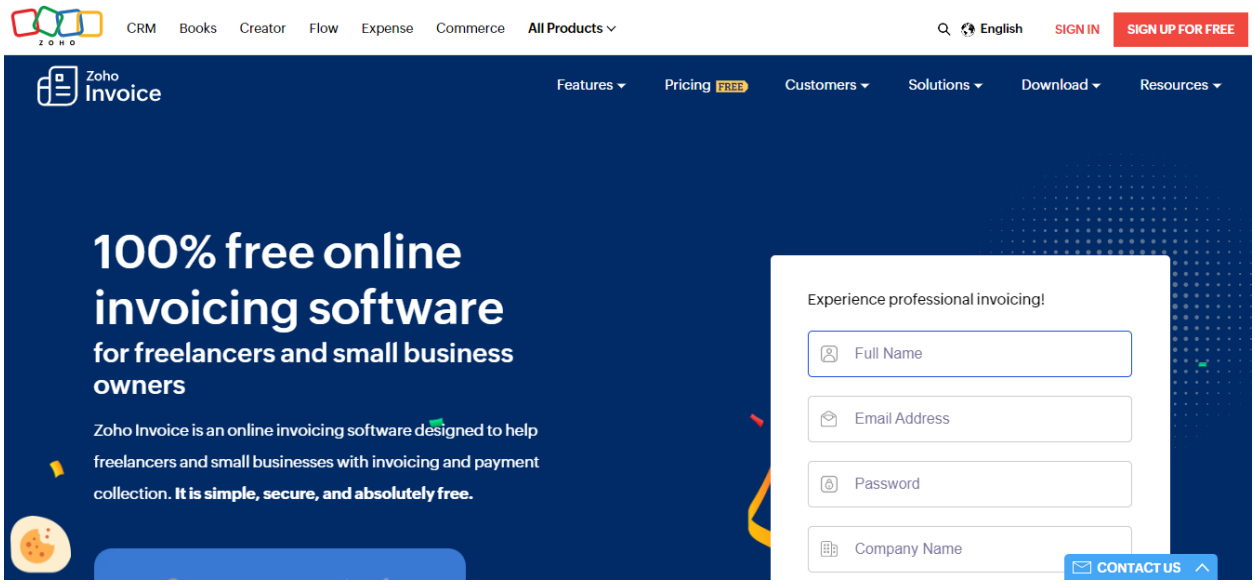
Postoji nekoliko primjera uspješnih implementacija web aplikacija za fakturiranje usluga koji su se istaknule u poslovnom svijetu. Ovi primjeri pokazuju kako korištenje web aplikacija može znatno poboljšati proces fakturiranja i financijsko upravljanje:

1. Invoice2Go: Invoice2Go [3] je popularna web aplikacija koja omogućuje malim poduzetnicima i samostalnim profesionalcima jednostavno izdavanje i upravljanje fakturama. Ova aplikacija nudi intuitivno sučelje, brzo generiranje faktura, praćenje plaćanja i mogućnost slanja faktura putem e-pošte ili mobilnih uređaja. Svojom funkcionalnošću i prilagodljivosti, Invoice2Go je postao priznata platforma za fakturiranje usluga.



Slika 2.1. Invoice2go naslovnica

2. Zoho Invoice: Zoho Invoice [4] je web aplikacija koja pruža sveobuhvatan skup alata za fakturiranje i financijsko upravljanje. Ova aplikacija omogućuje korisnicima generiranje profesionalnih faktura, upravljanje inventarom, praćenje plaćanja, slanje automatskih podsjetnika o plaćanju i integraciju s drugim Zoho alatima poput Zoho CRM-a. Zoho Invoice se ističe po svojoj jednostavnosti korištenja i prilagodljivosti potrebama različitih poduzeća.



Slika 2.2. Zoho invoice naslovnica

3. FreshBooks: FreshBooks [5] je popularna web aplikacija za fakturiranje i računovodstvo koja cilja mala i srednja poduzeća. Ova aplikacija nudi širok spektar funkcionalnosti, uključujući brzo generiranje faktura, praćenje vremena i troškova, upravljanje izvještajima, integraciju s bankovnim računima i slanje automatskih podsjetnika o plaćanju. FreshBooks se ističe po svojem intuitivnom sučelju i korisničkom iskustvu.

Slika 2.3. FreshBooks naslovnica

Ovi primjeri uspješnih implementacija web aplikacija za fakturiranje usluga pokazuju kako pravilno dizajnirane i funkcionalne aplikacije mogu značajno unaprijediti procese fakturiranja, povećati učinkovitost i smanjiti administrativne troškove. Kroz njihovu upotrebu, organizacije su

u mogućnosti pojednostaviti financijsko upravljanje, poboljšati praćenje plaćanja i stvoriti bolje korisničko iskustvo za partnere i klijente.

Aplikacija koja će biti izrađena u ovom radu je posebno usmjerena prema tvrtkama iz IT sektora, čime nudi specifične funkcionalnosti i karakteristike koje su relevantne za ovu industriju. To omogućava korisnicima iz IT sektora da pronađu rješenja koja su usklađena s njihovim potrebama. Također, korisničko sučelje je oblikovano s fokusom na intuitivnost i jednostavnost upotrebe. Ovo je ključno za korisnike koji žele brzo i efikasno upravljati fakturiranjem usluga bez potrebe za dugotrajnim obukama ili složenim postupcima.

3. ASPEKTI RAČUNOVODSTVA VEZANI UZ FAKTURIRANJE USLUGA

Fakturiranje usluga segment je poslovne ekonomije te pripada području financijskog upravljanja i računovodstva. U kontekstu ekonomije, fakturiranje usluga odnosi se na proces izdavanja faktura za pružene usluge kako bi se ostvario prihod i pratila financijska transakcija. Ovaj proces uključuje identifikaciju pruženih usluga, određivanje cijene i izrada službenog dokumenta (fakture) koji se šalje klijentu radi plaćanja. Fakturiranje usluga igra važnu ulogu u financijskom upravljanju poduzeća i pruža uvid u prihode, troškove i profitabilnost vezanu uz pružene usluge [7].

3.1. Relevantni ekonomski koncepti

Fakturiranje usluga uključuje nekoliko relevantnih ekonomskih koncepta koji se odnose na financijsko upravljanje i ekonomsku učinkovitost. Neki od tih ključnih koncepata uključuju:

1. Cjenovnu strategiju: Postavljanje pravilne cijene za pružene usluge ključno je za privlačenje klijenata, zadovoljstvo njihovih potreba i održavanje konkurentnosti. To uključuje analizu troškova, konkurentske cijene i vrijednosti koju vaše usluge pružaju klijentima.
2. Troškovnu analizu: Analiza troškova povezanih s pružanjem usluga ključna je za određivanje stvarne profitabilnosti. Razumijevanje direktnih i indirektnih troškova, fiksnih i varijabilnih troškova te troškova povezanih s infrastrukturom može pomoći u optimizaciji poslovanja.
3. Kalkulaciju profitabilnosti: Praćenje profitabilnosti svake usluge omogućuje prepoznavanje koje usluge doprinose većem prihodu i profita. Ova analiza pomaže donositi informirane odluke o ponudi usluga.
4. Ekonomiju mjerljivih efekata: Fakturiranje usluga može biti kompleksno zbog subjektivnosti kvalitete usluga. Pristup ekonomije mjerljivih efekata omogućuje bolje razumijevanje stvarne vrijednosti usluga kroz kvantifikaciju koristi koje klijenti dobivaju.
5. Ciklus gotovine: Fakturiranje usluga povezano je s upravljanjem ciklusom gotovine, što uključuje praćenje vremena od izdavanja fakture do naplate. Učinkovito upravljanje ciklusom gotovine može poboljšati likvidnost i stabilnost poslovanja.
6. Fiskalne odluke: Razumijevanje fiskalnih politika i zakonodavstva ključno je pri fakturiranju usluga. Različiti porezi i obveze mogu značajno utjecati na cjenovnu strategiju i profitabilnost.

7. Odnos s klijentima i lojalnost: Ekonomski koncepti kao što su vrijednost kupca, zadovoljstvo korisnika i lojalnost igraju značajnu ulogu u fakturiranju usluga. Dugoročni odnosi s klijentima mogu generirati ponovne poslove i preporuke, što ima direktan utjecaj na prihod.
8. Financijsko planiranje i prognozu: Planiranje budućih prihoda na temelju fakturiranja usluga ključno je za stabilno financijsko upravljanje. Analiza trenutnih trendova i projekcija budućih poslovnih performansi može pomoći u pravilnom financijskom planiranju.

Razumijevanje i primjena ovih ekonomskih koncepata pomaže u postizanju efikasnog upravljanja financijama u kontekstu fakturiranja usluga. Prema [8] to omogućuje poduzećima da donose informirane odluke, optimiziraju svoje poslovanje i ostvare uspješne financijske rezultate.

3.2. Tržište usluga

Tržište usluga je područje ekonomije gdje se trguje i razmjenjuju različite vrste usluga između pružatelja usluga i potrošača. Usluga je nematerijalna i često se temelji na znanju, vještinama ili iskustvu pružatelja usluga. Tržište usluga može biti vrlo raznoliko i obuhvaćati područja kao što su računovodstvo, konzultantske usluge, informatičke usluge, usluge zdravstvene skrbi, prijevozne usluge i mnoge druge.

Porezi su financijska sredstva koja država ili lokalne vlasti prikupljaju od pojedinaca i poduzeća kako bi financirale javne usluge i infrastrukturu. U kontekstu fakturiranja usluga, poduzeća obično moraju uzeti u obzir poreze pri određivanju cijena usluga i vođenju financijskih evidencija. Porezi mogu biti različiti po svojoj vrsti, poput poreza na dodanu vrijednost (PDV), poreza na dohodak ili lokalnih poreza, te mogu imati značajan utjecaj na financijske rezultate poduzeća.

Financijski aspekti uključuju sve financijske aktivnosti povezane s fakturiranjem usluga. To uključuje praćenje prihoda i troškova, izradu financijskih izvještaja, upravljanje novčanim tokovima, analizu profitabilnosti, planiranje budžeta i upravljanje financijskim rizicima. Upravljanje financijama uključuje donošenje informiranih odluka temeljenih na financijskim podacima kako bi se osigurala financijska stabilnost, efikasnost i dugoročna održivost poslovanja.

Ukupno gledano, tržište usluga, porezi i financijski aspekti su međusobno povezani i igraju ključnu ulogu u uspješnom fakturiranju usluga. Razumijevanje njihove dinamike i primjena odgovarajućih strategija može pomoći poduzećima da ostvare financijski uspjeh i konkurentske prednosti na tržištu [9].

3.3. Financijsko planiranje i analiza

Analiza troškova i profitabilnosti je ključni alat za poduzeća u kontekstu fakturiranja usluga kako bi razumjela strukturu troškova i profitabilnost svojih poslovnih aktivnosti. Kroz detaljnu analizu troškova, poduzeća mogu identificirati glavne troškovne komponente i njihovu raspodjelu te procijeniti njihov utjecaj na ukupne operativne troškove. To omogućuje poduzećima da prepoznaju područja s visokim troškovima ili potencijalnim izvorima ušteda te donesu informirane odluke o optimizaciji troškova.

S druge strane, analiza profitabilnosti omogućuje poduzećima da kvantificiraju prihode i usporede ih s troškovima kako bi procijenili profitabilnost svojih usluga. Identificiranje profitabilnih i manje profitabilnih segmenata poslovanja pomaže poduzećima u usmjeravanju resursa prema najprofitabilnijim područjima, optimizaciji cijena usluga ili restrukturiranju poslovnih procesa radi povećanja ukupne profitabilnosti.

Financijsko planiranje je ključni korak u postizanju financijskih ciljeva poduzeća. To uključuje definiranje dugoročnih ciljeva, kao što su rast prihoda, poboljšanje profitabilnosti ili širenje tržišnog udjela, te izrada strategija za postizanje tih ciljeva. Financijsko planiranje omogućuje poduzećima da donose informirane odluke o alokaciji resursa, investicijama, istraživanju i razvoju te marketinškim aktivnostima kako bi ostvarili željene financijske rezultate.

Budžetiranje je proces izrade detaljnog financijskog plana za određeno razdoblje, obično godinu. Kroz budžetiranje, poduzeća mogu predvidjeti prihode i identificirati sve potrebne troškove kako bi postigli planirane ciljeve. Budžetiranje omogućuje praćenje stvarnih financijskih rezultata u odnosu na planirane te identifikaciju odstupanja i poduzimanje korektivnih mjera ako je potrebno. To pruža financijsku kontrolu i omogućuje poduzećima da ostanu usklađena s planiranim ciljevima.

Kombinacija analize troškova i profitabilnosti s financijskim planiranjem i budžetiranjem omogućuje poduzećima da donose informirane odluke o upravljanju financijskim resursima. Ovi procesi pomažu u optimizaciji troškova, identifikaciji područja za poboljšanje profitabilnosti i postizanju dugoročne financijske održivosti. Također omogućuju poduzećima da budu proaktivna u odgovoru na promjene na tržištu, identificirajući prilike za rast i razvoj te upravljajući rizicima kako bi ostvarili stabilne financijske rezultate [9].

4. KORIŠTENE TEHNOLOGIJE

U današnjem brzom tehnološkom okruženju, web razvoj je postao osnovna komponenta modernog poslovanja i interakcije s korisnicima. Korištene tehnologije u razvoju web aplikacija imaju ključnu ulogu u oblikovanju i stvaranju korisničkog iskustva, pružajući alate za kreiranje responzivnih, dinamičnih i visoko funkcionalnih platformi. Kroz usklađivanje različitih tehnoloških slojeva - od klijentskih okvira i jezika za programiranje do poslužiteljske infrastrukture i baza podataka - korištene tehnologije omogućuju razvoj timova da transformiraju zamisli u stvarnost, donoseći inovacije koje zadovoljavaju zahtjeve suvremenih korisnika. U nastavku, istražit ćemo glavne komponente korištenih tehnologija i njihov doprinos u stvaranju naprednih web aplikacija koje oblikuju digitalno okruženje današnjice.

4.1. PROGRAMSKI OKVIR .NET

U početku devedesetih godina prošlog stoljeća, započeo je razvoj World Wide Weba, koji je postao najpoznatiji i najkorišteniji dio Interneta. Paralelno s tim, razvijale su se tehnologije koje su omogućavale implementaciju Web-a u različita područja, uključujući i programiranje. U tom kontekstu, Microsoft je uveo svoj Web programski jezik nazvan Active Server Pages (ASP) u okviru Internet Information Services (IIS) platforme.

Međutim, ASP je postao zastario u usporedbi s drugim jezicima kao što je PHP, koji je bio open-source i podržavao Linux/Unix operativne sustave. Stoga je Microsoft odlučio razviti novu tehnologiju koja bi predstavljala revoluciju u svijetu Web-a i informacijske tehnologije općenito. Tako je nastao .NET, arhitektura koja je objavljena 2001. godine.

Prva verzija .NET Frameworka izašla je 2002. godine zajedno s Microsoft Visual Studio razvojnim okruženjem. Ova tehnologija je brzo privukla veliku pozornost i interes programera diljem svijeta. Od tada, .NET je postao sveprisutan i nastavlja biti ključni dio Microsoftove strategije razvoja softvera.

.NET Framework [10] donosi mnoge prednosti, uključujući podršku za više jezika, bogatu biblioteku kôda, snažan alat za razvoj aplikacija i integrirano okruženje za programiranje. Također, .NET omogućuje razvoj aplikacija za različite platforme, uključujući Windows, Linux i macOS.

.NET je skup tehnologija i okvira koji omogućavaju razvoj, izvođenje i upravljanje različitim vrstama aplikacija na Microsoft platformi. Arhitektura .NET-a temelji se na nekoliko ključnih komponenti i koncepta.

Jezgru .NET arhitekture čini Common Language Runtime (CLR), koja je odgovorna za izvršavanje .NET programa. CLR je virtualno okruženje u kojem se .NET aplikacije izvršavaju. Ona pruža upravljanje memorijom, garbage collection (sakupljanje smeća), sigurnosne mehanizme, podršku za više jezika i druge ključne funkcionalnosti.

Još jedna bitna komponenta .NET arhitekture [11] je Class Library (klasna biblioteka), koja sadrži veliki skup preddefiniranih klasa, metoda i funkcionalnosti koje programeri mogu koristiti prilikom razvoja aplikacija. Klasna biblioteka obuhvaća različite područja, uključujući rad s mrežom, korisničkim sučeljima, bazama podataka, sigurnošću i mnoge druge.

.NET arhitektura također podržava više programskih jezika, kao što su C#, Visual Basic.NET, F# i drugi. Programeri mogu odabrati jezik koji najbolje odgovara njihovim potrebama i preferencijama, a svi ti jezici koriste zajednički CLR i imaju pristup klasnoj biblioteci.

Komponente aplikacije razvijene u .NET-u organizirane su u skupine koje se nazivaju assembly-ji. Svaki assembly sadrži jednu ili više komponenti, kao što su klase, resursi i druge datoteke potrebne za izvođenje aplikacije. Assembly-ji omogućuju modularnost, ponovnu upotrebu koda i jednostavno upravljanje aplikacijama.

Još jedan važan koncept u .NET arhitekturi je interoperabilnost. .NET omogućuje komunikaciju između različitih aplikacija i tehnologija putem standardnih protokola i formata, kao što su SOAP (Simple Object Access Protocol) i XML (eXtensible Markup Language). To omogućuje integraciju .NET aplikacija s drugim sustavima i servisima.

Uz to, .NET arhitektura podržava različite vrste aplikacija, uključujući desktop aplikacije, web aplikacije, mobilne aplikacije, servise, mikroservise i druge. Razvoj takvih aplikacija olakšan je kroz različite alate, kao što su razvojna okruženja (Visual Studio), alati za testiranje, upravljanje paketima, sustavi za verzioniranje i druge [6].

4.2. Struktura poslužitelja u .NET frameworku

4.2.1. Servis

U kontekstu .NET razvoja, *service* (usluga) [10] predstavlja komponentu koja obavlja određene zadatke ili operacije unutar aplikacije. Servisi su ključni dio arhitekture aplikacije jer sadrže poslovnu logiku i omogućavaju odvajanje te logike od ostalih dijelova aplikacije poput kontrolera i entiteta.

Evo detaljnijeg objašnjenja ključnih aspekata servisa u .NET:

1. Poslovna logika: Servisi sadrže poslovnu logiku aplikacije. To su operacije ili funkcionalnosti koje se trebaju izvršiti kako bi aplikacija pravilno funkcionirala. Primjeri poslovne logike u servisima mogu uključivati pristup bazi podataka, obradu poslovnih pravila, izračune, komunikaciju s vanjskim servisima i još mnogo toga.
2. Odvajanje od korisničkog sučelja: Servisi omogućavaju jasno odvajanje poslovne logike od korisničkog sučelja (kontrolera i pogotovo prikaza). To znači da korisničko sučelje (npr. web stranica ili aplikacija) ne bi trebalo sadržavati složenu poslovnu logiku, već bi ta logika trebala biti izdvojena u servise. To olakšava testiranje i održavanje, jer se poslovna logika može mijenjati ili nadopunjavati bez utjecaja na korisničko sučelje.
3. Reusability (Ponovna upotreba): Servisi omogućavaju ponovnu upotrebu poslovne logike u različitim dijelovima aplikacije. Na primjer, isti servis za pristup bazi podataka može se koristiti u različitim kontrolerima ili čak u različitim aplikacijama unutar istog sustava.
4. Skalabilnost: Pravilno izrađeni servisi omogućavaju lakše skaliranje aplikacije. To znači da se može dodati više instanci servisa kako bi se nosilo s većim brojem zahtjeva, bez potrebe za značajnim promjenama u samoj poslovnoj logici.
5. Testiranje: Servisi olakšavaju testiranje aplikacije. Budući da sadrže većinu poslovne logike, možete izolirano testirati svaki servis kako biste provjerili je li ispravno implementiran. Ovo olakšava jedinično testiranje i osigurava da svaki dio poslovne logike radi ispravno.
6. Dependency Injection: Dependency Injection (DI) je čest pristup u .NET razvoju gdje se servisi ubacuju ili "ubrizgavaju" u kontrolere ili druge servise. Ovo olakšava prilagodbu i zamjenu servisa te omogućava bolju kontrolu nad ovisnostima između komponenata.

Svaki servis bi trebao imati dobro definiranu odgovornost i biti usredotočen na obavljanje specifične operacije ili funkcionalnosti. Pravilno izrađeni servisi pridonose čistoći koda, modularnosti i održavanju sustava te čine aplikaciju skalabilnom i fleksibilnom.

4.2.2. Kontroler

Prema [10], [11] kontroler je ključna komponenta u arhitekturi Model-View-Controller (MVC) i Model-View-ViewModel (MVVM) koje se često koriste u .NET razvoju. Kontroler je odgovoran za obradu ulaznih zahtjeva korisnika, koordinaciju poslovne logike i pripremu podataka za prikazivanje putem korisničkog sučelja.

Evo detaljnog objašnjenja ključnih aspekata kontrolera u .NET:

1. Obrada zahtjeva: Kontroler prima HTTP zahtjeve od korisnika (ili drugih izvora) i odlučuje kako će aplikacija reagirati na taj zahtjev. Na primjer, zahtjev za prikazom određene stranice web aplikacije ili izvođenjem određene operacije.
2. Koordinacija poslovne logike: Kontroler koordinira izvođenje poslovne logike. To znači da poziva odgovarajuće servise (ili druge komponente) kako bi obavio operacije potrebne za obradu zahtjeva. Na primjer, kontroler može pozvati servis za pristup bazi podataka ili servis za izračune.
3. Priprema podataka za prikaz: Kontroler priprema podatke koji će biti prikazani korisniku putem korisničkog sučelja (View). Ovi podaci mogu biti dohvaćeni iz baze podataka, servisa ili drugih izvora. Kontroler oblikuje ove podatke u oblik koji je pogodan za prikaz na korisničkom sučelju.
4. Upravljanje sesijama i stanjem: Kontroler može upravljati sesijama korisnika i održavati stanje između različitih zahtjeva. Na primjer, kontroler može upravljati prijavom korisnika ili praćenjem odabira korisnika.
5. Upravljanje navigacijom: Kontroler može upravljati navigacijom korisnika kroz aplikaciju. To uključuje odlučivanje koje će stranice ili akcije biti prikazane korisniku ovisno o njegovim interakcijama.
6. Interakcija s korisničkim sučeljem: Kontroler komunicira s korisničkim sučeljem (View) kako bi prikazao rezultate operacija ili ažurirao podatke nakon korisnikovih interakcija. Ovo uključuje generiranje odgovora koji će biti poslan korisniku (npr. HTML za web aplikacije).
7. Testiranje: Kontroler olakšava testiranje aplikacije. Budući da kontroler koordinira poslovnu logiku, možete izolirano testirati svaki kontroler kako biste provjerili je li ispravno implementiran.
8. Dependency Injection: Slično kao i kod servisa, Dependency Injection (DI) se često koristi za ubacivanje servisa ili drugih komponenata u kontrolere. Ovo omogućava bolju kontrolu nad ovisnostima i olakšava testiranje.

Kontroleri čine važan sloj u arhitekturi aplikacije jer omogućavaju organizaciju i upravljanje interakcijama s korisnicima te pružaju sredstva za učinkovitu koordinaciju poslovne logike.

4.2.3. DTO

DTO, ili Data Transfer Object (Objekt za prijenos podataka), je obrazac dizajna [10], [11] koji se koristi za prenošenje podataka između različitih slojeva aplikacije ili između različitih komponenata. U .NET razvoju, DTO je čest koncept koji se koristi za učinkovit prijenos podataka

između kontrolera, servisa i pogleda te omogućava odvajanje stvarne strukture podataka od podataka koji se prikazuju korisniku.

Evo detaljnog objašnjenja ključnih aspekata DTO-a u .NET:

1. Prijenos podataka: DTO je prvenstveno namijenjen prijenosu podataka između različitih slojeva aplikacije. To može uključivati prijenos podataka između kontrolera i servisa, servisa i baze podataka te između različitih servisa.
2. Smanjenje opterećenja mreže: Kada se komunicira između različitih komponenti ili servisa, ponekad je nepotrebno prenositi cijele entitete s kompleksnim strukturama. DTO omogućava odabir samo relevantnih podataka za prijenos, smanjujući tako opterećenje mreže i povećavajući učinkovitost komunikacije.
3. Skalabilnost: Kada aplikacija raste i postaje složenija, upotreba DTO-ova može olakšati skaliranje pojedinih dijelova aplikacije. Na primjer, ako više servisa komunicira putem API-ja, svaki servis može koristiti samo potrebne podatke, čime se smanjuje nepotrebno komuniciranje.
4. Pristup sigurnosnim i osjetljivim podacima: DTO-ovi omogućavaju izolaciju osjetljivih podataka. Na primjer, ako aplikacija sadrži osjetljive informacije o korisnicima, kao što su lozinke, te informacije ne bi trebale biti dio DTO-a koji se šalje korisničkom sučelju.
5. Povezivanje više entiteta: Ponekad su podaci potrebni za prikaz sastavljeni iz više izvora ili entiteta. DTO omogućava agregaciju podataka iz različitih izvora u jedan objekt koji se može koristiti za prikaz na korisničkom sučelju.
6. Razdvajanje poslovne logike i prikaza: DTO omogućava odvajanje stvarne strukture podataka (entiteta) od oblika podataka koji se prikazuju korisniku. Na primjer, možete imati kompleksne entitete s puno atributa, ali za prikaz samo odabrane atribute korisniku.
7. Jednostavnost prilagodbe promjenama: Kada se promijeni struktura podataka unutar aplikacije, prilagodba promjenama u DTO-ima može olakšati kompatibilnost između različitih komponenata ili servisa.
8. Testiranje: Upotreba DTO-ova olakšava testiranje pojedinih komponenata aplikacije jer omogućava izolirano testiranje poslovne logike, neovisno o prikazu.

U .NET razvoju, DTO-ovi obično su jednostavne klase koje sadrže samo svojstva (atribute) i nemaju poslovnu logiku. Oni su pasivni objekti čija je svrha omogućiti efikasan prijenos podataka između različitih komponenata aplikacije.

4.2.4. Prikaz podataka putem entiteta

Prema [10], [11] entiteti su ključna komponenta u arhitekturi aplikacije, posebno u kontekstu baze podataka. Entiteti se koriste za modeliranje stvarnog svijeta u digitalnom obliku, čime se omogućava interakcija s podacima unutar aplikacije. U .NET razvoju, entiteti predstavljaju strukturu i ponašanje tablica u bazi podataka te omogućuju manipulaciju podacima.

Evo detaljnog objašnjenja ključnih aspekata entiteta u .NET:

1. Modeliranje podataka: Entiteti predstavljaju strukturu podataka u bazi. Svaki entitet obično odgovara jednoj tablici u bazi podataka, a svojstva entiteta odgovaraju kolonama u toj tablici. Na primjer, entitet "Korisnik" može imati svojstva kao što su "Ime", "Prezime", "Email" itd.
2. Povezanost i relacije: Entiteti omogućavaju definiranje odnosa između tablica. Na primjer, entitet "Narudžba" može biti povezan s entitetom "Proizvod" preko vanjskog ključa. Ovo omogućava uspostavljanje složenih relacija između podataka.
3. Validacija i poslovna logika: U entitetima se često provode osnovne provjere valjanosti podataka. Na primjer, možete definirati da polje "Cijena" u entitetu "Proizvod" ne smije biti negativno. Također, određena poslovna logika može biti ugrađena u entitete kako bi se osiguralo dosljedno ponašanje aplikacije.
4. Prilagodba bazi podataka: Entiteti omogućavaju definiranje kako će se struktura podataka odražavati u bazi podataka. Ovo uključuje informacije kao što su tipovi podataka, ograničenja, indeksi i druge karakteristike tablica.
5. ORM (Object-Relational Mapping): U .NET razvoju, ORM alati (kao što je Entity Framework) omogućavaju generiranje entiteta iz baze podataka i mapiranje objekata na tablice. Ovo olakšava interakciju s bazom podataka, jer se entiteti ponašaju kao most između baze i aplikacije.

4.2.5. Mapiranje baze podataka pomoću migracija

Prema [10], [11] migracije su mehanizam koji omogućava upravljanje promjenama u strukturi baze podataka tijekom vremena. Kada se razvija aplikacija, često će doći do potrebe za izmjenama u bazi podataka, kao što su dodavanje novih tablica, promjene stupaca ili indeksa. Migracije omogućavaju automatizirano primjenjivanje ovih promjena.

Evo detaljnog objašnjenja ključnih aspekata migracija u .NET:

1. Inkrementalne promjene: Migracije omogućavaju izgradnju i primjenu promjena u bazi podataka korak po korak. Svaka migracija predstavlja jednu promjenu i može se primijeniti na bazu kako se aplikacija razvija.
2. Praćenje verzija: Migracije omogućavaju praćenje verzija baze podataka. Svaka migracija ima svoj redni broj i opis promjene. Ovo olakšava praćenje povijesti promjena i omogućava vraćanje na prethodne verzije.
3. Automatizacija: Migracije omogućavaju automatizirano primjenjivanje promjena. Kroz migracije, promjene u bazi podataka mogu se primijeniti na testnoj, razvojnoj i produkcijskoj bazi podataka uz minimalan napor.
4. Sigurnost i dosljednost: Migracije osiguravaju dosljednost između različitih instanci baze podataka. Bez migracija, svaka instanca baze mogla bi biti u različitom stanju, što bi moglo uzrokovati nepredvidive rezultate.
5. Rollback: Migracije često omogućavaju povrat na prethodno stanje baze podataka. Ako promjene nisu uspješno primijenjene ili izazivaju probleme, moguće je koristiti migracije za povrat na prethodno stanje.

U .NET razvoju, Entity Framework je čest alat koji omogućava generiranje entiteta i upravljanje migracijama. Kroz Entity Framework migracije, promjene u entitetima mogu se automatski prenijeti na bazu podataka.

4.2.6. Code-First način razvoja

Prema [12] Code-First razvoj je pristup razvoju aplikacija koji se često koristi u kreiranju baza podataka i modela podataka. Ovaj pristup omogućava programerima da prvo definiraju modele podataka putem programskog koda, a zatim se baze podataka generiraju automatski na temelju tih modela. Evo detaljnijeg opisa Code-First pristupa:

1. Definiranje Modela: Prvi korak u Code-First razvoju je definiranje modela podataka putem koda. Ovi modeli predstavljaju entitete i odnose između njih. Na primjer, ako razvijate aplikaciju za fakturiranje usluga, mogli biste imati modele kao što su Invoice (Faktura), Customer (Klijent), Service (Usluga) i slično.
2. Korištenje Anotacija: U Code-First pristupu, programer koristi anotacije ili atribute iz određenog ORM (Object-Relational Mapping) alata kako bi dodatno obogatio modele informacijama o bazama podataka. Ove anotacije mogu sadržavati informacije o primarnim ključevima, stranim ključevima, indeksima i drugim svojstvima baze podataka.

3. DbContext Klasa: DbContext je klasa koja predstavlja vezu s bazom podataka. U njoj se definiraju DbSet-ovi, koji su svojevrsne reprezentacije tablica u bazi podataka. Svaki DbSet odgovara jednom modelu.
4. Konvencije i Konfiguracije: Code-First razvoj koristi konvencije kako bi automatski prepoznao odnose između modela. Na primjer, ako imate navigacijska svojstva u modelima (npr. Invoice ima Customer), EF (Entity Framework) će automatski prepoznati te odnose. Ako je potrebno, programer može koristiti i konfiguracijske klase za precizno definiranje odnosa i svojstava baze podataka.
5. Generiranje Baze Podataka: Nakon definiranja modela i konfiguracija, EF koristi te informacije kako bi generirao bazu podataka. To uključuje kreiranje tablica, veza, ključeva i ostalih struktura prema definiranom modelu.
6. Migracije: Migracije su mehanizam za ažuriranje baze podataka na temelju promjena u modelima. Kada se modeli promijene, EF može generirati skripte koje ažuriraju bazu podataka prema novim definicijama.
7. Debugging i Testiranje: Code-First pristup olakšava debugging i testiranje jer se svi modeli i konfiguracije nalaze unutar vašeg koda. To olakšava otkrivanje problema i ispravke.

Code-First razvoj omogućava agilno i fleksibilno stvaranje baza podataka te olakšava razvoj web aplikacija. Međutim, treba napomenuti da ova metoda može biti složena u slučaju većih i kompleksnijih baza podataka, pa je važno dobro planirati modele i konfiguracije prije nego što se krene s implementacijom. Glavni razlog korištenja ovog načina razvoja je brzi i jako jednostavni prijelaz s jedne na drugu vrstu baze podataka.

4.2.7. Usporedba HTTP metoda

HTTP metode igraju ključnu ulogu u komunikaciji između klijenta i servera u web aplikacijama. One omogućavaju različite akcije koje se mogu izvršiti nad resursima na serveru. Svi imaju svoje specifične svrhe i primjene. HTTP metode uključuju GET, POST, PUT, DELETE, PATCH, OPTIONS, HEAD, CONNECT i TRACE.

GET metoda se koristi za dohvaćanje podataka sa servera. Ona ne mijenja stanje servera i odgovara s informacijama koje su zatražene.

POST metoda se koristi za slanje podataka na server kako bi se stvorio novi resurs ili izvršila neka akcija. Ona mijenja stanje servera i nekad može vratiti odgovor s novim resursom ili dodatnim informacijama.

PUT metoda se koristi za ažuriranje postojećeg resursa na serveru. Ona zamjenjuje postojeće podatke novim podacima koje je klijent poslao.

DELETE metoda se koristi za brisanje resursa na serveru. Ona obriše traženi resurs i obično ne vraća nikakav sadržaj osim statusne oznake.

PATCH metoda se koristi za parcijalno ažuriranje resursa na serveru. Umjesto zamjene cijelog resursa, klijent šalje samo promjene koje treba primijeniti.

OPTIONS metoda se koristi za dobivanje informacija o podržanim metodama i mogućnostima na određenom resursu.

HEAD metoda je slična GET metodi, ali vraća samo zaglavlje odgovora bez stvarnog sadržaja.

CONNECT metoda se koristi za uspostavljanje mrežne veze s resursom, često se koristi za uspostavljanje sigurnih veza.

TRACE metoda se koristi za dobivanje dijagnostičkih informacija o putanji do resursa.

Svaka metoda ima svoj namjenski scenarij i koristi se ovisno o tome što se želi postići u komunikaciji između klijenta i servera.

4.3. Razvoj korisničkog sučelja pomoću Angular-a

Prema [13] Angular je open-source JavaScript okvir za izgradnju modernih, dinamičnih web aplikacija. Razvijen i održavan od strane Google-a, Angular omogućava razvoj aplikacija korištenjem TypeScript jezika, proširene verzije JavaScripta koja pruža strože tipiziranje i napredne značajke za razvoj softvera. Angular se temelji na arhitekturi komponenti, što olakšava organizaciju i održavanje koda.

Evo detaljnog objašnjenja ključnih aspekata Angulara:

1. Komponente: Angular se temelji na konceptu komponenata. Komponente su osnovne građevne jedinice aplikacije i predstavljaju dijelove korisničkog sučelja. Svaka komponenta sadrži HTML za prikaz, TypeScript kod za logiku i stilove koji definiraju izgled komponente.
2. Moduli: Aplikaciju u Angularu organiziramo kroz module. Modul je kontejner koji grupira srodne komponente, servise i druge resurse. To omogućava bolju organizaciju i omogućava lakše dijeljenje komponenata između različitih dijelova aplikacije.

3. Direktive: Direktive su upute koje se primjenjuju na HTML elemente kako bi se mijenjao izgled ili ponašanje tih elemenata. Angular dolazi s ugrađenim direktivama poput ngIf za uvjetno prikazivanje elemenata ili ngFor za iteraciju kroz liste.
4. Servisi: Servisi su komponente koje sadrže poslovnu logiku i omogućavaju dijeljenje podataka i funkcionalnosti između različitih dijelova aplikacije. Servisi se obično koriste za komunikaciju s poslužiteljem, pristup bazi podataka ili manipulaciju podacima.
5. Dependency Injection: Angular koristi Dependency Injection (DI) za ubacivanje servisa i drugih komponenata u druge komponente. Ovo olakšava prilagodbu i testiranje, omogućava bolju kontrolu nad ovisnostima i promovira modularnost.
6. Rute: Angular omogućava razvoj više stranica u aplikaciji kroz upravljanje rutama. Rute definiraju koje komponente će se prikazati ovisno o URL-u, omogućavajući razvoj dinamičkih višestraničnih aplikacija.
7. Reaktivno programiranje: Angular podržava reaktivno programiranje kroz RxJS (Reactive Extensions for JavaScript). Ovo omogućava elegantno rješavanje asinkronih operacija, kao što su HTTP zahtjevi ili promjene korisničkog unosa.
8. Testiranje: Angular pridaje veliku važnost testiranju i dolazi s okvirom za testiranje koji omogućava jednostavno pisanje i izvođenje testova za komponente, servise i ostale dijelove aplikacije.
9. Optimizacija i performanse: Angular uključuje alate za optimizaciju i analizu performansi aplikacije, što omogućava otkrivanje i rješavanje problema koji bi mogli usporiti aplikaciju.

Angular je popularan odabir za izradu dinamičnih i složenih web aplikacija, pružajući programerima alate i strukturu potrebnu za razvoj modernih korisničkih sučelja.

4.3.1. NGX-bootstrap

Prema [14] ngx-bootstrap je biblioteka komponenata za Angular koja omogućava jednostavno korištenje Bootstrap CSS okvira unutar Angular aplikacija. Bootstrap je popularni okvir za izradu responsivnih i modernih korisničkih sučelja, a ngx-bootstrap pruža gotove Angular komponente koje se lako mogu integrirati u web aplikaciju.

Evo detaljnog objašnjenja ključnih aspekata ngx-bootstrap:

1. Kompatibilnost s Angularom: ngx-bootstrap je specifično prilagođen za Angular okruženje. To znači da su komponente optimizirane za korištenje u Angular aplikacijama, koriste Angular mehanizme poput detekcije promjena i životnih ciklusa komponenata.

2. Bootstrap komponente: ngx-bootstrap pruža širok spektar Bootstrap komponenata koje možete koristiti unutar Angular aplikacije. To uključuje komponente poput modala, tooltipova, karusela, padajućih izbornika, tablica i još mnogo toga.
3. Jednostavna integracija: Integriranje ngx-bootstrap komponenti u Angular aplikaciju je jednostavno. Samo treba instalirati paket, uključiti potrebne module i početi koristiti komponente u kodu.
4. Prilagodljivost: Komponente iz ngx-bootstrap-a su prilagodljive. To znači da se može prilagoditi izgled i ponašanje komponenti kako bi se uklopile u dizajn aplikacije. Također, mogu se koristiti Bootstrap teme ili prilagoditi temu prema svojim potrebama.
5. Reaktivno programiranje: ngx-bootstrap podržava reaktivno programiranje kroz integraciju s RxJS. Ovo omogućava rukovanje asinkronim događajima i promjenama stanja komponenti na elegantan način.
6. Lako održavanje: Korištenjem ngx-bootstrap komponenata može se smanjiti količinu koda potrebnu za izradu različitih funkcionalnosti. To olakšava održavanje aplikacije jer se više ne mora brinuti o detaljima implementacije pojedinih Bootstrap komponenata.
7. Vodiči i dokumentacija: ngx-bootstrap dolazi s obuhvatnom dokumentacijom i primjerima koji olakšavaju učenje i korištenje komponenata.
8. Zajednica i podrška: Biblioteka ima aktivnu zajednicu korisnika i održavatelja. Ako se naiđe na poteškoće ili pitanja, odgovor se može pronaći putem foruma ili dokumentacije.

U konačnici, ngx-bootstrap olakšava integraciju popularnog Bootstrap okvira u Angular aplikacije, čime omogućava brži razvoj i bolju konzistentnost korisničkog sučelja.

4.3.2. Angular material

Prema [15] Angular Material je open-source biblioteka koja pruža set gotovih materijalnih dizajnerskih komponenata i temi za Angular aplikacije. Razvijen od strane Google-a, Angular Material omogućava brz i jednostavan razvoj modernih i atraktivnih korisničkih sučelja, slijedeći smjernice Material Designa.

Evo detaljnog objašnjenja ključnih aspekata Angular Materiala:

1. Materijalne komponente: Angular Material pruža širok spektar gotovih komponenata koje su dizajnirane prema smjernicama Material Designa. To uključuje komponente kao što su gumbi, kartice, padajući izbornici, obrasci, tablice, alatne trake, modalni prozori, ikone i mnoge druge.

2. Responzivni dizajn: Sve komponente iz Angular Materiala su responsivne, što znači da se prilagođavaju različitim uređajima i veličinama ekrana. To olakšava izradu aplikacija koje izgledaju dobro na desktop računalima, tabletima i mobilnim uređajima.
3. Teme i prilagodljivost: Angular Material dolazi s nekoliko unaprijed definiranih tema koje slijede smjernice Material Designa. Teme se mogu jednostavno prilagoditi prema potrebama specifične aplikacije, uključujući promjene boja, tipografije i drugih stilskih elemenata.
4. Kompaktibilnost s Angularom: Angular Material je izgrađen posebno za Angular aplikacije. To znači da su komponente optimizirane za korištenje u Angular okruženju, a integracija je jednostavna i prirodna.
5. Accessibility (Pristupačnost): Angular Material se fokusira na pristupačnost i dizajniran je takav da bude koristan i za korisnike s posebnim potrebama. Komponente dolaze s ugrađenim pristupačnim atributima i smjernicama.
6. Animacije: Mnoge komponente dolaze s unaprijed definiranim animacijama koje dodaju interaktivnost i bolje korisničko iskustvo.
7. Modularnost: Angular Material je modularan, što znači da se mogu uključiti samo one komponente koje vam trebaju u vašoj aplikaciji, čime se smanjuje veličina koda i optimizira brzina učitavanja.
8. Dokumentacija i vodiči: Angular Material dolazi s detaljnom dokumentacijom, primjerima i vodičima koji olakšavaju učenje i korištenje komponenata.

Korištenjem Angular Materiala, mogu se brzo izraditi atraktivna i moderna korisnička sučelja za Angular aplikacije, slijedeći smjernice koje potiču kvalitetu i konzistentnost dizajna.

4.4. Postgres

PostgreSQL je moćan i open-source objektno-relacijski sustav [16] za upravljanje bazama podataka (DBMS). Razvijen je kao nastavak istraživačkog projekta na sveučilištu Kalifornija u Berkeleyju te se često naziva skraćenicom "Postgres". PostgreSQL pruža napredne značajke, stabilnost i podršku za širok raspon upita i operacija, čineći ga popularnim izborom za aplikacije različitih razmjera i kompleksnosti.

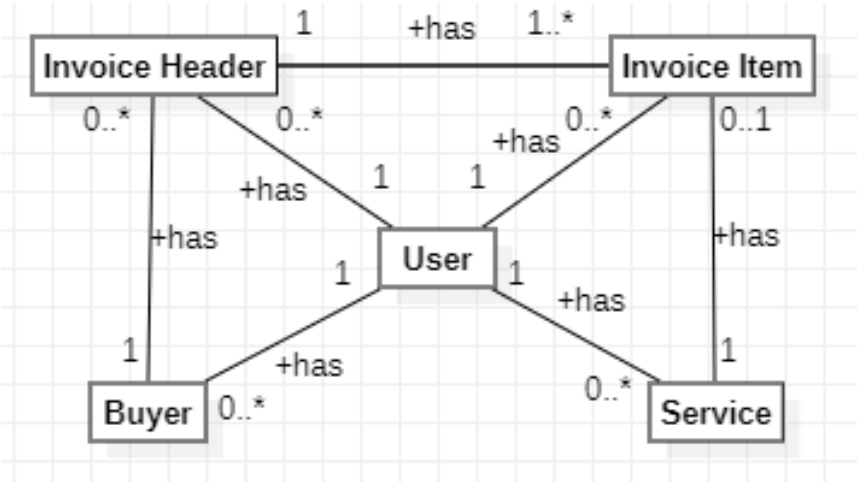
Evo detaljnog objašnjenja ključnih aspekata PostgreSQL-a:

1. Objektno-relacijski sustav: PostgreSQL kombinira prednosti relacijskih i objektnih baza podataka. Ovo omogućava programerima da koriste svojstva objektnog modela, poput

nasljeđivanja i polimorfizma, zajedno s prednostima tradicionalnih relacijskih baza podataka.

2. Podrška za SQL: PostgreSQL podržava standardni SQL jezik, što omogućava izgradnju kompleksnih upita i manipulacija podacima. Osim standardnog SQL-a, PostgreSQL također nudi prošireni jezik nazvan PL/pgSQL koji se koristi za pisanje funkcija i procedura.
3. Napredne značajke: PostgreSQL ima širok spektar naprednih značajki kao što su indeksiranje punog teksta, geografski informacijski sustavi (GIS), podrška za JSON i JSONB tipove podataka te podrška za transakcije i ACID (Atomicity, Consistency, Isolation, Durability) svojstva.
4. Višekorisnička podrška i sigurnost: PostgreSQL pruža složen sustav za upravljanje korisnicima, pravima pristupa i sigurnošću podataka. Ovo omogućava preciznu kontrolu nad tko može pristupiti i manipulirati podacima.
5. Pouzdanost i skalabilnost: PostgreSQL je poznat po svojoj pouzdanosti i stabilnosti. Pruža opcije replikacije podataka i klasteriranja za postizanje visokih performansi i dostupnosti.
6. Open Source i aktivna zajednica: PostgreSQL je open-source projekt s velikom i aktivnom zajednicom razvijatelja i korisnika. Ova zajednica stalno radi na unaprjeđenju i razvoju PostgreSQL-a te pruža podršku i resurse korisnicima.
7. Proširenja i dodaci: PostgreSQL omogućava instalaciju proširenja i dodataka koji dodatno proširuju funkcionalnost sustava. Postoje proširenja za specifične potrebe kao što su dodatne vrste podataka, alati za optimizaciju i više.
8. Performanse: PostgreSQL je poznat po svojim dobrim performansama, posebno uz pravilno indeksiranje i optimizaciju. Ima mehanizme automatskog optimiziranja upita i mogućnost podešavanja performansi za različite vrste opterećenja.
9. Razvojni alati: PostgreSQL dolazi s različitim alatima za upravljanje bazama podataka, uključujući interaktivne sučelja i API-je za automatizaciju upravljanja bazom.
10. Kompatibilnost: PostgreSQL je kompatibilan s različitim operativnim sustavima, uključujući Unix-bazirane sustave, Windows i macOS.

PostgreSQL se često koristi kao baza za web aplikacije jer pruža pouzdanost, fleksibilnost i podršku za napredne značajke. Ovisno o potrebama, PostgreSQL može biti izvrstan izbor za pohranu i upravljanje podacima vaše web aplikacije.



Slika 4.1. Konceptualni klasni dijagram

Na slici 4.1 prikazan je konceptualni klasni dijagram koji opisuje strukturu baze podataka web aplikacije.

5. POSTUPAK IZRADE WEB APLIKACIJE

U ovom poglavlju, bit će opisan proces instalacije i konfiguracije okruženja, kao i sam proces izrade web aplikacije. Tok podataka i komunikacija s bazom bit će objašnjeni, zajedno s prikazom podataka. Interakcija između poslužiteljskog i klijentskog dijela će biti analizirana, a naglasak će biti stavljen na glavni slučaj korištenja aplikacije i procese koji se odvijaju u pozadini.

5.1. Instalacija i konfiguracija okruženja

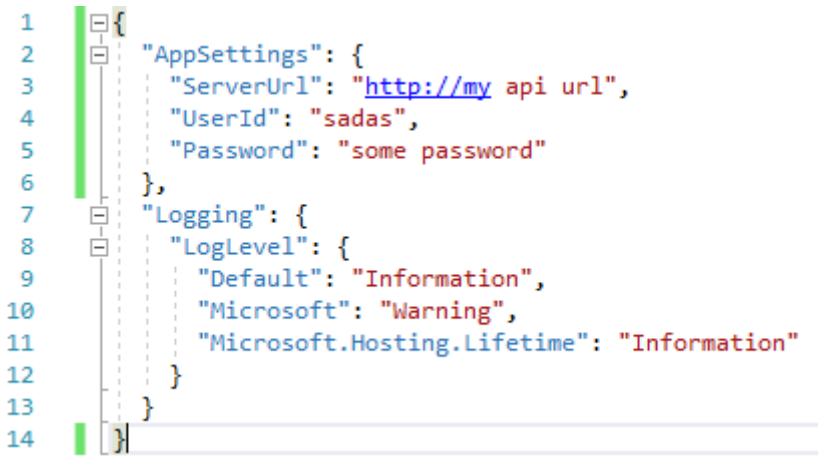
U ovom dijelu detaljno će biti istražen postupak instalacije i konfiguracije [11] razvojnog okruženja za razvoj poslužiteljskog dijela aplikacije za fakturiranje usluga. Ključni alati koji su korišteni u ovom procesu uključuju Visual Studio za poslužiteljski dio, Visual Studio Code za klijentski dio, te PGAdmin za upravljanje bazom podataka.

Prvi korak bio je instaliranje Visual Studio alata, koji je pružio integrirano razvojno okruženje za .NET aplikacije. Kroz jednostavan instalacijski postupak, omogućena su sva potrebna sredstva za razvoj poslužiteljskog dijela. Za klijentski dio razvoja, korišten je Visual Studio Code, lagani i moćan uređivač koda. Instalirani su relevantni dodaci kako bi se podržala Angular razvojna okolina. Za upravljanje bazom podataka korišten je PGAdmin, alat za PostgreSQL bazu podataka. Nakon instalacije, konfigurirane su veze s bazom kako bi se omogućio pristup podacima i upravljanje strukturom baze. Za konfiguriranje veze s bazom bilo je potrebno urediti „appsettings.json“ datoteku. Datoteka "appsettings.json" predstavlja konfiguracijsku datoteku koja se često koristi u .NET aplikacijama, posebno u ASP.NET Core projektima. Ova datoteka služi za pohranu različitih konfiguracijskih parametara i postavki koje su potrebne za pravilno funkcioniranje aplikacije.

U *appsettings.json* datoteci, kao što je prikazano na slici 5.1., mogu se definirati različite vrste postavki, uključujući:

1. Veze s Bazom Podataka: Konfiguracija veza prema bazama podataka, kao što su tip baze podataka, server, korisničko ime, lozinka i drugi relevantni parametri.
2. Tokeni i Ključevi: Pohranjivanje tajnih ključeva, API tokena i drugih osjetljivih podataka koji se koriste za autentikaciju i autorizaciju.
3. URL-ovi i Adrese: Definiranje vanjskih servisa, API-ja ili drugih vanjskih resursa s kojima aplikacija komunicira.
4. Postavke Aplikacije: Parametri koji utječu na ponašanje aplikacije, poput jezika, vremenske zone, logika poslovnih pravila itd.

5. Konfiguracije Dijagnostike i Logiranja: Postavke za praćenje i logiranje događaja i grešaka unutar aplikacije.



Slika. 5.1. Primjer "appsettings.json" datoteke

5.2. Opis razvoja poslužiteljskog dijela aplikacije

U okviru ovog poglavlja, detaljno će se istražiti ključni aspekti razvoja poslužiteljskog dijela web aplikacije za fakturiranje usluga. Razvoj poslužitelja ima ključnu ulogu u osiguravanju stabilnosti, sigurnosti i funkcionalnosti aplikacije. Kroz naredne sekcije, bit će upoznato s postupcima, tehnologijama i konceptima koji su omogućili implementaciju tog bitnog dijela aplikacije.

U ovom dijelu rada usmjerit će se pažnja na temeljne elemente poslužiteljskog razvoja i njegovu važnost unutar cjelokupne strukture aplikacije. Razumijevanje poslužiteljskog razvoja omogućit će dublji uvid u proces obrade podataka, osiguranja podataka i interakcije s korisničkim sučeljem.

Očekuje se istraživanje izazova vezanih uz sigurnost, autentikaciju i autorizaciju korisnika te upravljanje podacima u bazi. Kroz naredne sekcije, istražit će se kako je osigurana autentikacija korisnika, kako se generiraju i provjeravaju JWT (JSON Web Token) [18] tokeni te kako su modelirani entiteti i implementirani servisi za obradu podataka.

Razumijevanje poslužiteljskog dijela bitno je kako bi kvalitetno razvijala aplikacija za fakturiranje usluga. Kroz analizu ovog procesa, stvorit će se čvrsta osnova za daljnji rad na implementaciji funkcionalnosti aplikacije te će biti osigurano da poslužiteljski dijelovi budu stabilni i efikasni.

Isprva je bilo potrebno napraviti sustav za prijavljivanje korisnika. Započeto je kreiranjem klase samog korisnika i servisa za token. *AppUser* klasa predstavlja model korisnika u web aplikaciji.

Ova klasa sadrži ključne informacije o korisniku te ima ulogu u autentikaciji i upravljanju fakturama, uslugama i kupcima.

1. *Id*: Svaki korisnik ima jedinstveni identifikator (*Id*) koji se koristi za precizno prepoznavanje korisnika unutar baze podataka. Ovo je osnovna referenca za identifikaciju korisnika i povezivanje s drugim entitetima.
2. *UserName*: Polje *UserName* predstavlja korisničko ime korisnika. Ovo korisničko ime je važno za autentikaciju korisnika prilikom prijave te se koristi kao tvrdnja unutar autentikacijskog tokena.
3. *PasswordHash* i *PasswordSalt*: Ova dva polja predstavljaju hash vrijednost i "salt" (sol) lozinke korisnika. Hashiranje lozinke je važna sigurnosna mjera kako bi se spriječilo pohranjivanje stvarnih lozinki u bazi podataka. *Salt* dodatno povećava sigurnost hashiranja.
4. *InvoiceHeaders*: Lista *InvoiceHeaders* povezuje korisnika s njegovim izdanjima faktura. Ovaj atribut omogućava praćenje faktura povezanih s određenim korisnikom.
5. *Services*: Lista *Services* povezuje korisnika s uslugama koje pruža ili koristi. Ovaj atribut omogućava praćenje usluga koje su relevantne za korisnika.
6. *Buyers*: Lista *Buyers* povezuje korisnika s njegovim kupcima. Ovaj atribut omogućava praćenje odnosa korisnika s različitim kupcima.

Kroz ovu klasu, aplikacija omogućava registraciju, prijavu i upravljanje korisničkim računima. Svaki korisnik ima svoj jedinstveni identifikator, korisničko ime te pohranjene hash vrijednosti i salt njegove lozinke. Osim toga, kroz veze s drugim entitetima, kao što su faktura, usluge i kupci, omogućava se organizacija i povezivanje relevantnih podataka za svakog korisnika.

U web aplikaciji, autentikacijski tokeni se koriste kako bi se korisnicima omogućio siguran pristup različitim dijelovima aplikacije. Klasa *TokenService* ima ključnu ulogu u generiranju tih autentikacijskih tokena.

Prvi korak je inicijalizacija ključa za potpisivanje tokena. Ovaj ključ je osnova za generiranje digitalnog potpisa koji osigurava integritet i vjerodostojnost autentikacijskog tokena. Ključ se stvara iz konfiguracijskih podataka aplikacije.

Nakon što je ključ za potpisivanje spremljen, kreiranje autentikacijskog tokena započinje prikupljanjem informacija o korisniku. U ovom slučaju, to uključuje korisničko ime kao identifikator. Ove informacije se koriste za generiranje "tvrdnji" (engl. *claims*), koje su osnovne informacije sadržane unutar autentikacijskog tokena.

Nakon stvaranja tvrdnji, definiraju se ostali detalji autentikacijskog tokena. To uključuje definiranje trajanja tokena, odnosno koliko će dugo token biti važeći. Ovo je važno kako bi se osiguralo da token ne može biti zloupotrijebljen nakon isteka vremenskog razdoblja.

Konačno, token se generira. To se postiže stvaranjem opisa tokena, koji uključuje tvrdnje, trajanje i informacije o potpisu. Prema [17] sama generacija tokena koristi sigurnosni algoritam (HMAC-SHA512) kako bi se osigurala njegova autentičnost.

Nakon što je token generiran, on se pretvara u tekstualni format kako bi bio spreman za korištenje. Ovaj tekstualni token može se koristiti u HTTP zahtjevima kako bi korisnik dokazao svoj identitet i pristupio određenim dijelovima aplikacije.

Kroz ovaj proces, klasa *TokenService* osigurava da se korisniku generira siguran autentikacijski token koji omogućava pristup aplikaciji na način koji je zaštićen i pouzdan. Ovaj token igra ključnu ulogu u osiguravanju sigurnosti aplikacije te omogućava korisnicima siguran pristup njihovim fakturama i uslugama.

Nadalje, bilo je potrebno razviti i implementirati funkcionalnosti za registraciju i prijavu korisnika, te provjeriti postojanje korisnika unutar aplikacije. To je postignuto kroz *AccountController* klasu, koja nasljeđuje *BaseApiController* kako bi iskoristila njegove osnovne funkcionalnosti. Ključne metode ove klase su *Register* i *Login*, koje omogućavaju korisnicima registraciju novih računa i prijavu putem svojih vjerodajnica.

Metoda *Register* je ozbiljno zaokupljena procesom registracije novih korisnika. Kroz HTTP POST zahtjev, ova metoda prima podatke koji su uneseni prilikom registracije, poput korisničkog imena i lozinke. Prvo, metoda provjerava ispravnost unesenih podataka provjerom *ModelState*. Ako podaci nisu ispravni, vraća se *BadRequest* odgovor.

Zatim se provjerava postoji li korisnik s unesenim korisničkim imenom koristeći metodu *UserExists*. Ako postoji, registracija se ne može nastaviti jer korisničko ime već postoji, te se vraća *BadRequest* sa porukom *Username is taken*.

Ako je korisničko ime slobodno, generira se tzv. HMACSHA512 objekt za hashiranje i šifriranje lozinke. Kreira se novi *AppUser* objekt sa korisničkim imenom i hashiranom lozinkom. Korisničko ime se sprema u malim slovima kako bi se osigurala dosljednost prilikom provjere autentikacije.

Nakon toga, korisnik se dodaje u bazu podataka kroz *_context.Users.Add(user)* i sprema se koristeći *_context.SaveChangesAsync()*. Kao odgovor se vraća *UserDTO* objekt sa korisničkim imenom i generiranim autentikacijskim tokenom.

Metoda *Login* se koristi za provjeru vjerodajnica i izdavanje autentikacijskog tokena za prijavljene korisnike. Koristi se HTTP POST zahtjev s unesenim korisničkim imenom i lozinkom. Prvo, provjerava se postoji li korisnik s unesenim korisničkim imenom. Ako korisnik ne postoji, vraća se *Unauthorized* s porukom *Invalid username*.

Zatim se generira HMACSHA512 objekt s korisničkom salt vrijednosti i provjerava se usklađenost hash vrijednosti lozinke iz zahtjeva s pohranjenom hash vrijednošću u bazi podataka. Ako se hash vrijednosti ne podudaraju, korisnik se nije ispravno prijavio i vraća se *Unauthorized* s porukom *Invalid password*.

U slučaju ispravne prijave, generira se autentikacijski token pomoću *_tokenService.CreateToken(user)* i vraća se u *UserDTO* objektu s korisničkim imenom i tokenom.

Kroz ove dvije ključne metode, *AccountController* omogućava korisnicima registraciju i prijavu na siguran način, osiguravajući autentikaciju putem generiranih autentikacijskih tokena. Također, *UserExists* metoda služi za provjeru postojanja korisnika u bazi podataka pri registraciji.

5.3. Opis razvoja klijentskog dijela aplikacije

Klijentski dio razvoja predstavlja ključni aspekt stvaranja suvremenih web aplikacija koji se fokusira na kreiranje vizualnog i interaktivnog korisničkog iskustva. On predstavlja most između korisnika i tehničkog sloja aplikacije, omogućavajući korisnicima intuitivan i ugodan način interakcije s sadržajem i funkcionalnostima.

Započeto je s komponentom navigacije. Ovaj segment koda predstavlja implementaciju gornje navigacijske trake (*navbar*), prikazane na slici 5.2., unutar web aplikacije. Navigacijska traka pruža korisnicima brzi pristup različitim dijelovima aplikacije i omogućava interakciju s korisničkim računom.

U početnom dijelu koda definirani su različiti elementi navigacijske trake, kao što su logotip *IT4 invoicing* i opcije za navigaciju kao što su *Invoices*, *Partners* i *Services*. Ovisno o stanju korisničkog računa, određeni dijelovi trake bit će vidljivi ili skriveni.

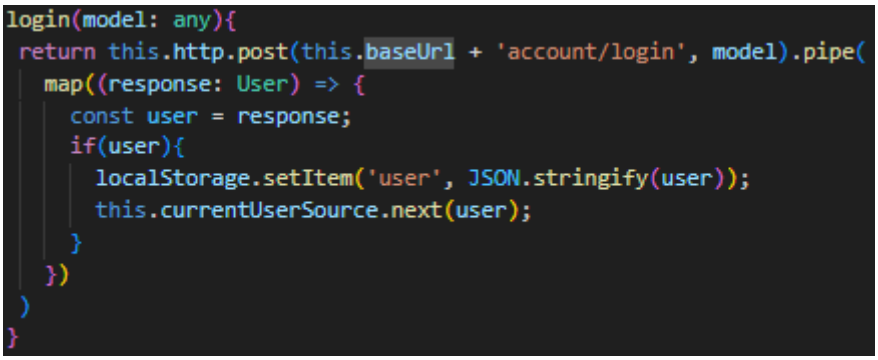
Nadalje, implementirana je padajuća lista s korisničkim imenom, omogućavajući korisniku da se odjavi. Također, postoji forma za prijavu koja se prikazuje samo ako korisnik nije prijavljen, pružajući korisnicima način da se prijave u sustav.

U konačnici, ovaj segment koda kombinira HTML strukturu s Angular direktivama i komponentama kako bi stvorio funkcionalnu i dinamičku navigacijsku traku, poboljšavajući time ukupno korisničko iskustvo u web aplikaciji.

Slika 5.2 Navigacijska traka

Također, bilo je potrebno kreirati klijentski servis koji obavlja pozive na poslužitelja u vezi korisničkih računa. Nazvan je *account.service.ts*. *AccountService* ima nekoliko ključnih zadaća vezanih uz upravljanje korisničkim računima i autentikaciju unutar aplikacije:

1. Prijavljivanje (*Login*): Metoda *login*, prikazana na slici 5.3., omogućava korisnicima da se prijave u aplikaciju. Korisnik unosi svoje korisničko ime i lozinku, a *AccountService* šalje zahtjev poslužitelju za provjerom tih podataka. Ako su podaci točni, korisnik se uspješno prijavljuje, a njegovi podaci i token pohranjuju se u lokalnoj memoriji.
2. Registracija (*Register*): Metoda *register* omogućava novim korisnicima da se registriraju u aplikaciju. Unose osnovne podatke kao što su korisničko ime, lozinka i ostali relevantni podaci. Nakon uspješne registracije, korisnički podaci se pohranjuju u lokalnoj memoriji.
3. Odjava (*Logout*): Metoda *logout* omogućava korisnicima da se odjave iz aplikacije. Svi podaci o prijavljenom korisniku i tokenu se brišu iz lokalne memorije, što korisniku onemogućava pristup osjetljivim podacima i funkcionalnostima nakon odjave.
4. Praćenje Trenutno Prijavljenog Korisnika: Pomoću *currentUser*$ *observable*-a i metoda kao što su *currentUser*$ i *setCurrentUser*, *AccountService* prati trenutno prijavljenog korisnika tijekom sesije. To omogućava praćenje korisnika i omogućava pristup njegovim podacima i ovlastima.
5. Dohvaćanje *Bearer* Tokena: Metoda *getBearerToken* omogućava pristup JWT (JSON Web Token) token koji se koristi za autentikaciju korisnika pri komunikaciji s poslužiteljem. Ovo omogućava slanje ovlaštenih zahtjeva na poslužiteljske dijelove aplikacije.
6. Održavanje Sigurnosti: *AccountService* također ima ulogu u održavanju sigurnosti podataka i korisničkih sesija. Omogućava kontrolu pristupa i provjerava identitet korisnika pri svakom zahtjevu prema poslužitelju.
7. Centralizacija Logike: Sve gore navedene zadaće centralizirane su unutar *AccountService* klase. To olakšava upravljanje i održavanje autentikacijske i korisničke logike.



Slika 5.3. Isječak koda iz servisa

Kroz ove zadaće, *AccountService* pruža osnovne funkcionalnosti vezane uz korisničke račune, autentikaciju i upravljanje korisničkim sesijama, čime omogućava sigurno i pravilno korisničko iskustvo unutar aplikacije.

5.4. Oblikovanje i komunikacija s bazom podataka

Nadalje, bilo je potrebno razviti i glavni dio aplikacije, servise koji komuniciraju s PostgreSQL bazom u svrhu dohvaćanja i manipulacije podataka o uslugama, kupcima i fakturama.

Razvoj service-a i kontrolera za CRUD operacije (Create, Read, Update, Delete) za entitete usluga, kupaca i faktura pruža osnovnu funkcionalnost za upravljanje tim entitetima unutar aplikacije za fakturiranje usluga. Evo općeg opisa procesa razvoja za svaki od entiteta:

Usluge (Services):

- Razvoj Service-a: Kreiran je *ServiceService* koji sadrži metode za dohvaćanje, kao što je prikazano na slici 5.4., stvaranje, ažuriranje i brisanje podataka vezanih uz usluge. Ove metode koriste HTTP zahtjeve prema odgovarajućem poslužiteljskom endpointu.
- Razvoj Kontrolera: *ServiceController* je razvijen kako bi obrađivao HTTP zahtjeve koji dolaze od klijentskog dijela aplikacije. Kontroler definira rute za CRUD operacije i upravlja pozivanjem odgovarajućih metoda iz *ServiceService*.

Kupci (Buyers/Partners):

- Razvoj Service-a: Stvoren je *BuyerService* koji sadrži metode za manipulaciju podacima o kupcima. Metode uključuju dohvaćanje svih kupaca, stvaranje novog kupca, ažuriranje postojećeg i brisanje kupca.
- Razvoj Kontrolera: *BuyerController* je razvijen kako bi obrađivao HTTP zahtjeve povezane s kupcima. Kontroler definira rute za različite CRUD operacije i koristi metode iz *BuyerService* za njihovu implementaciju.

Razvoj servisa i kontrolera za fakturu uključuje upravljanje podacima o fakturi, konkretno njenim zaglavljem, ali i indirektno povezanim stavkama fakture putem stranog ključa. Ovdje je više detalja o ovom aspektu:

- Razvoj Service-a: *InvoiceService* je osmišljen kako bi omogućio kreiranje, dohvaćanje, ažuriranje i brisanje faktura. Osim toga, ovaj servis pruža metode za upravljanje stavkama faktura koje su povezane s fakturama putem stranog ključa.
- Razvoj Kontrolera: *InvoiceController* upravlja HTTP zahtjevima vezanim uz fakturu. Kontroler pruža rute za stvaranje, dohvaćanje, ažuriranje i brisanje faktura. Također, kontroler koristi odgovarajuće metode iz *InvoicesService* za manipulaciju fakturama.
- Indirektno Upravljanje: Premda se servis i kontroler za fakturu bave prvenstveno zaglavljem faktura, te entitete omogućavaju povezivanje s drugim entitetima, kao što su stavke fakture. Stavke fakture povezane su s fakturama preko stranog ključa (npr. faktura ID), iako direktna manipulacija stavkama nije središnji fokus ovih komponenti.

Ovim se osigurava da servis i kontroler za fakturu omogućavaju operacije s fakturama, uključujući stvaranje, dohvaćanje, ažuriranje i brisanje, te indirektnu manipulaciju stavkama fakture putem njihovog povezanosti s fakturama. Ovo omogućava potpunu funkcionalnost vezanu uz fakturiranje usluga unutar aplikacije.

```
public async Task<IEnumerable<ServiceDTO>> GetServices(string username)
{
    var user = await _userRepository.GetUserByUsernameAsync(username);
    var services = await _dbContext.Services
        .Where(x => x.AppUserId == user.Id)
        .ToListAsync();

    return services.Select(service => MapToDTO(service));
}
```

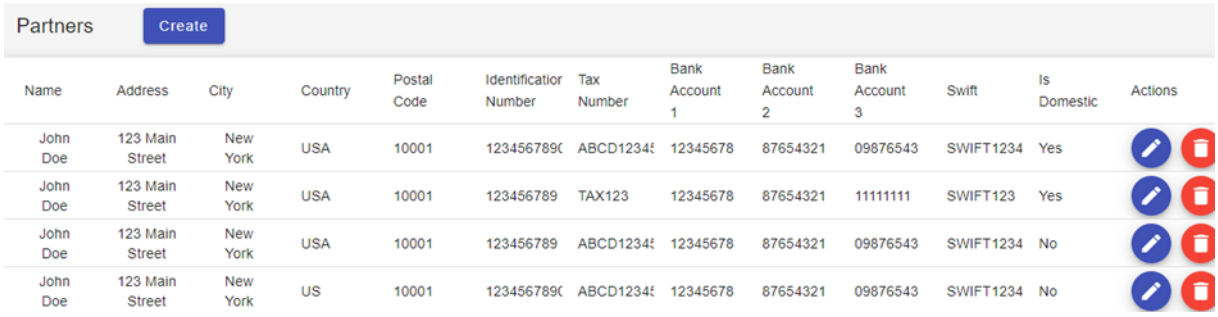
Slika 5.4. Isječak koda za dohvat usluga

5.5. Interakcija s podacima na klijentskom dijelu

U izradi aplikacije bilo je potrebno prikazati ove podatke koji se šalju s poslužitelja, kao što je prikazano na slici 5.6. Taj dio odrađen je pomoću Angular Material *mat-table* strukture prikazane na slici 5.5. Ona omogućava responzivan dizajn s dodatnim mogućnostima za integriranje drugih struktura kao što su gumbi i ikonice. Također, omogućava rukovanje klikom na redak.

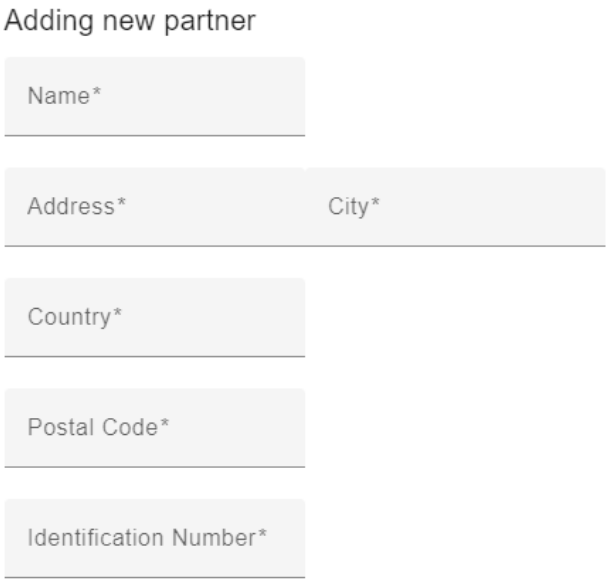
```
<mat-table [dataSource]="services" class="mat-elevation-z8">
  <ng-container matColumnDef="name">
    <mat-header-cell *matHeaderCellDef class="mat-header-cell">Name</mat-header-cell>
    <mat-cell *matCellDef="let service" class="mat-cell">{{ service.name }}</mat-cell>
  </ng-container>

  <ng-container matColumnDef="price">
    <mat-header-cell *matHeaderCellDef class="mat-header-cell">Price</mat-header-cell>
    <mat-cell *matCellDef="let service" class="mat-cell">{{ service.price }}</mat-cell>
  </ng-container>

  <ng-container matColumnDef="priceInEuros">
    <mat-header-cell *matHeaderCellDef class="mat-header-cell">Price in Euros</mat-header-cell>
    <mat-cell *matCellDef="let service" class="mat-cell">{{ calculatePriceInEuros(service.price)
  </ng-container>
```

Slika 5.5. Isječak koda tablice za prikaz podataka



Slika 5.6. Primjer izgleda tablice za kupce/partnere

Također, bilo je potrebno napraviti forme za unos novih i uređivanje postojećih kupaca i usluga, te naposljetku i računa.



Slika 5.7. Dio forme za unos kupca/partnera

Nakon završetka ovog koraka, cjelokupni radni tok aplikacije je potpun. Korisniku se pruža interaktivno sučelje izrađeno u HTML-u i stilizirano putem CSS-a. Ovo sučelje omogućava interakciju s korisnikom i njegove akcije obrađuje TypeScript programski jezik unutar Angular okvira.

Kada korisnik klikne na određene gumbe ili inicira akciju, TypeScript programski jezik upućuje poziv ka unaprijed definiranim Angular servisima. Ovi servisi komuniciraju putem HTTP metoda i šalju prateći token za autentikaciju.

Ovaj poziv se prenosi do odgovarajućeg kontrolera na poslužiteljskom dijelu aplikacije, koji se nalazi unutar .NET okvira. Ovdje se zahtjev dalje obrađuje, bilo da se radi o pretraživanju baze podataka ili izmjeni podataka. Kontroler dalje prosljeđuje zahtjev specifičnom servisu koji obavlja operaciju nad bazom podataka.

Nakon što servis obavi operaciju nad bazom podataka, ako je potrebno, vraća rezultate. Ti rezultati se zatim prosljeđuju klijentskom dijelu aplikacije, gdje se prikazuju korisniku. Na ovaj način se kompletira ciklus interakcije između korisnika, klijentskog dijela temeljenog na Angularu i poslužiteljskog dijela temeljenog na .NET-u.

Ovaj integrirani proces omogućava korisniku efikasno koristiti aplikaciju, pristupajući i manipulirajući podacima putem intuitivnog sučelja, a sve se odvija uz čvrstu sigurnost autentikacije i integraciju s bazom podataka.

6. ZAKLJUČAK

Ovaj diplomski rad obuhvatio je širok raspon aspekata vezanih uz razvoj web aplikacije za fakturiranje usluga. Kroz analizu relevantnih ekonomskih koncepta i tehnoloških alata, radom je istaknuta povezanost između ekonomske učinkovitosti i preciznosti u upravljanju financijama te uloga koju web aplikacije imaju u ovom kontekstu. Naglašeno je da automatizacija procesa fakturiranja pruža učinkovitost, ubrzava financijske transakcije i minimizira rizike povezane s ljudskim pogreškama.

Kroz rad je ukazano na pozitivne aspekte primjene web aplikacije za fakturiranje. Istaknuta je važnost integracije klijentskih i poslužiteljskih komponenti te se detaljno istražile tehnologije poput Angulara i .NET-a, koje su osigurale stabilan temelj za razvoj aplikacije.

Analizirajući specifične komponente aplikacije, ukazano je na to da su entiteti, migracije, servisi i DTO (Data Transfer Object) ključni za postizanje kvalitetne i skalabilne arhitekture. Kroz primjere koda, naglasak je stavljen na njihovu svrhu i funkcionalnost u okviru aplikacije.

U sklopu budućeg razvoja ove aplikacije za fakturiranje usluga, postoji niz mogućnosti za proširenje funkcionalnosti i poboljšanja korisničkog iskustva. Posebno je važno istaknuti mogućnost omogućavanja suradnje više korisnika unutar iste tvrtke, što se može postići primjenom različitih strategija.

Jedna od ključnih strategija je implementacija sustava korisničkih uloga i pristupa. Ovime bi se omogućila definicija različitih razina pristupa i ovlasti za različite korisničke profile, uključujući administratore, menadžere i obične korisnike. Ova prilagodba omogućava svakom korisniku pristup samo onim funkcionalnostima koje su relevantne za njegovu ulogu.

Uspostavljanje radnih tokova i procesa odobrenja je još jedan važan korak. Kroz implementaciju radnih tokova, faktura bi prolazila kroz seriju korisnika na odobrenje prije finalnog izdavanja. Ovo omogućava bolju kontrolu i transparentnost tokom samog procesa fakturiranja.

Integracija funkcionalnosti komentiranja i interne komunikacije unutar aplikacije također bi donijela značajne koristi. Ovo bi omogućilo korisnicima da međusobno komuniciraju, ostavljaju komentare uz fakture te razmjenjuju informacije, čime bi se ubrzao protok informacija.

Također, aplikacija bi mogla podržavati dodjelu zadataka i odgovornosti. Na primjer, menadžeri bi mogli jednostavno dodjeljivati zadatke članovima tima s jasno definiranim rokovima i ciljevima.

Praćenje aktivnosti i revizija omogućava potpunu transparentnost. Uvođenjem mehanizma za praćenje aktivnosti, svaka korisnička akcija unutar aplikacije bi bila zabilježena, omogućavajući bolje praćenje promjena i događanja.

Dalje, razvoj analitičkih alata i izvještavanja omogućava tvrtki da analizira poslovne podatke i generira relevantne izvještaje. Ovo pomaže donositeljima odluka da budu informirani i temeljene svoje strategije na stvarnim podacima.

Nadalje, integracija s drugim alatima, kao što su CRM (engl. *Customer relationship management*) sustavi ili financijski softveri, može pružiti povezanost s drugim ključnim poslovnim aspektima.

Razvoj mobilne verzije aplikacije dodatno bi povećao korisničku fleksibilnost. Omogućio bi korisnicima da pristupaju i upravljaju fakturama i izvan ureda, čime bi se povećala produktivnost i omogućila rad s bilo kojeg mjesta.

Ovaj diplomski rad istražio je interdisciplinarnu prirodu web aplikacije za fakturiranje usluga, kombinirajući ekonomske principe s programskim vještinama. Postignuta je jasna slika o važnosti učinkovitog financijskog upravljačkog alata u suvremenom poslovnom okruženju te je istaknuto kako tehnologija može biti ključna za ostvarivanje ovog cilja.

LITERATURA

[1] B Koch – „Significant market transition lies ahead“, Billentis, 2017

[2] Assessing the Carbon Footprint of Paper vs. Electronic Invoicing, dostupno na:

https://aisel.aisnet.org/acis2010/95/ [Datum posjete web sjedišta: 21.06.2023.]

[3] Invoice2Go dostupno na:

https://invoice.2go.com/ [Datum posjete web sjedišta: 25.08.2023.]

[4] Zoho Invoice dostupno na:

https://www.zoho.com/invoice/ [Datum posjete web sjedišta: 25.08.2023.]

[5] FreshBooks dostupno na:

https://www.freshbooks.com/ [Datum posjete web sjedišta: 25.08.2023.]

[6] .NET Blog dostupno na:

https://devblogs.microsoft.com/dotnet/ [Datum posjete web sjedišta: 27.06.2023.]

[7] O Korkman, K Storbacka, B Harald, „Practices as Markets: Value Co-Creation in E-Invoicing“, Australasian Marketing, 2010

[8] J Godfrey, A Hodgson, A Tarca, J Hamilton, S Holmen, „Accounting“, 2010 dostupno na:

http://ellisarchive.org/sites/default/files/2019-09/Document_20190903_0008_0.pdf [Datum posjete web sjedišta: 28.06.2023.]

[9] RE Lane, „The market experience“ , Cambridge University Press, 1991

[10] N Barbettini – „The Little ASP.NET Core Book“, Creative Commons Attribution, 2017

[11] S Chiaretta, U Lattanzi, „Asp.Net Core Succinctly“, Syncfusion, 2017

[12] Code First to a New Database dostupno na:

https://learn.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database

[Datum posjete web sjedišta: 23.08.2023.]

[13] D Kumar, „Angular Essentials: The Essential Guide to Learn Angular“, BPB, 2019

[14] Angular Bootstrap dostupno na:

https://valor-software.com/ngx-bootstrap/#/ [Datum posjete web sjedišta: 23.08.2023.]

[15] V K Kotaru, „Angular for Material Design“, Apress, 2020

[16] B Momjian, „PostgreSQL: Introduction and Concepts“, Addison-Wesley, 2004

[17] Ravilla, Dilli and R Putta, C Shekar, „Security of Zone Routing Protocol using HMAC-SHA512 Algorithm“, dostupno na:

https://eprints.manipal.edu/148052/ [Datum posjete web sjedišta: 23.08.2023.]

[18] JSON Web Token Introduction dostupno na:

https://jwt.io/introduction [Datum posjete web sjedišta: 29.08.2023.]

SAŽETAK

U ovom diplomskom radu je detaljno istražena izgradnja web aplikacije za fakturiranje usluga s naglaskom na ekonomske i tehnološke aspekte. Analizirani su ekonomski koncepti u vezi s financijskom učinkovitošću i točnošću, ističući ulogu automatizacije u procesu fakturiranja.

Proučavane su uspješne primjene sličnih aplikacija koje su pokazale njihov pozitivan utjecaj na poslovne procese. Detaljno su opisane tehnologije kao što su Angular i .NET, ključne za izgradnju aplikacije. Kroz primjere kodova, istražene su komponente kao što su entiteti, migracije, servisi i DTO, koje su osigurale strukturu i funkcionalnost aplikacije.

Na temelju istraživanja, zaključeno je da su efikasnost i točnost od vitalnog značaja za financijsko upravljanje, a web aplikacije igraju ključnu ulogu u ubrzanju financijskih transakcija. Kroz integraciju ekonomskih principa i tehnoloških rješenja, rad je naglasio kako tehnologija može podržati optimizaciju financijskog poslovanja u modernom poslovnom okruženju.

Ključne riječi: baza podataka, fakturiranje, poslužitelj, usluge, web aplikacija

ABSTRACT

Invoicing web application

This master's thesis extensively explored the development of a web application for service invoicing, emphasizing both economic and technological aspects. Economic concepts related to financial efficiency and accuracy were analyzed, highlighting the role of automation in the invoicing process.

Successful implementations of similar applications were examined, demonstrating their positive impact on business processes. Technologies such as Angular and .NET were thoroughly described as fundamental to the application's construction. Through code examples, components like entities, migrations, services, and DTOs were explored, providing the structure and functionality of the application.

Based on the research, it was concluded that efficiency and accuracy are vital for financial management, and web applications play a pivotal role in expediting financial transactions. By integrating economic principles and technological solutions, the thesis underscored how technology can support optimizing financial operations in the modern business environment.

Keywords: backend, database, invoicing, services, web app

ŽIVOTOPIS

Ilija Petrović rođen je 5. srpnja 1999. godine u Orašju, Bosna i Hercegovina. Završio je osnovnu školu Braće Radić u Domaljevcu, nakon čega je nastavio svoje obrazovanje u srednjoj školi fra Martina Nedića u Orašju, gdje je odabrao smjer opća gimnazija. Godine 2018. donosi odluku da će se upisati na Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, na smjer Računarstvo. Nakon uspješno završenog preddiplomskog studija, odlučuje nastaviti svoje obrazovanje na istom fakultetu, ovaj put upisujući smjer Programsko inženjerstvo.

_____

Potpis autora

PRILOZI

Prilog 1: Pristup kodu izrađene aplikacije: https://github.com/PetrovicIlija/InvoicingWebApp