Internet aplikacija za komunikaciju kratkim porukama

Merkaš, Matej

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:886748

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij

INTERNET APLIKACIJA ZA KOMUNIKACIJU KRATKIM PORUKAMA

Diplomski rad

Matej Merkaš

Osijek, 2017.

Sadržaj:

1. UVOD 1
1.1. Zadatak diplomskog rada 1
2. TEHNOLOGIJE 2
2.1. Visual Studio 2
2.2. HTML 8
2.3. CSS 10
2.4. C# 12
2.5. AngularJS 13
2.6. SignalR 14
3. IZVEDBA SUSTAVA 16
4. ZAKLJUČAK 28
LITERATURA 29
SAŽETAK 30
ABSTRACT 30
ŽIVOTOPIS 31
PRILOG A. Osnovni HTML program 32
PRILOG B. HTML program za ekran za prijavu 33
PRILOG C. HTML program za ekran za razgovor 34
PRILOG D. CSS program 35
PRILOG E. JavaScript program za funkcioniranje razgovora 38
PRILOG F. JavaScript program za prijavljivanje korisnika 39
PRILOG G. C# program za prijavu korisnika i kreiranje razgovora 41
PRILOG H. C# program za funkcioniranje razgovora 44
PRILOG I. C# program za bazu podataka 46

1. UVOD

Cilj diplomskog rada je izrada internet aplikacije za komunikaciju kratkim porukama. Potrebno je napraviti ASP.NET aplikaciju koja omogućava komunikaciju porukama između korisnika. Potrebno je izraditi korisničko sučelje, aplikacijsko-programsko sučelje te bazu podataka. Izrada aplikacije za komunikaciju kratkim porukama zahtijeva korištenje više tehnologija koje se velikim dijelom koriste u poslovnom svijetu. Aplikacija za komunikaciju kratkim porukama omogućava razgovor s prijateljima i poslovnim kolegama komunikaciju u svakom trenutku i na svakom mjestu na vrlo jednostavan način. Kroz poglavlja je opisano sve što je potrebno za rad aplikacije i kako su pojedini dijelovi objedinjeni u cjelinu. Potrebno je izraditi internet aplikaciju u Visual Studio-u kojom se omogućava korisnicima komunikaciju putem poruka u stvarnom vremenu. U prvom poglavlju, po potpoglavljima, opisane su tehnologije korištene pri izradi zadatka. Za svaki programski jezik opisane su njegove najvažnije karakteristike te na koji način se koristi i za što služi pri programiranju. U drugom poglavlju opisano je kako su pojedini dijelovi programa spojeni u cjelinu, izvedba rada, tj. način na koji pojedini dio aplikacije funkcionira. Također, opisan je i redoslijed koji je korišten pri izradi aplikacije. U trećem poglavlju, zaključku, sumirana je ukratko izvedba rada, osvrt na dobiveno rješenje, te problemi pri realiziranju sustava. Kod izrade diplomskog rada korišteni su podaci prikupljeni iz stručne literature i usmjereni su na temu rada.

1.1. Zadatak diplomskog rada

Izraditi ASP.NET internet aplikaciju za komunikaciju kratkim porukama u realnom vremenu.

2. TEHNOLOGIJE

2.1. Visual Studio

Microsoft Visual Studio je integrirano ravojno okruženje tvrtke Microsoft. Koristi se za razvoj računalnih programa za Microsoft Windows, kao i web stranice, web aplikacije, web usluge i mobilne aplikacije. Visual Studio koristi Microsoftove platforme za razvoj softvera kao što su Windows API, Windows Forms, Windows Presentation Foundation, Windows Store i Microsoft Silverlight. Može proizvesti i izvorni i upravljački kod. [1]

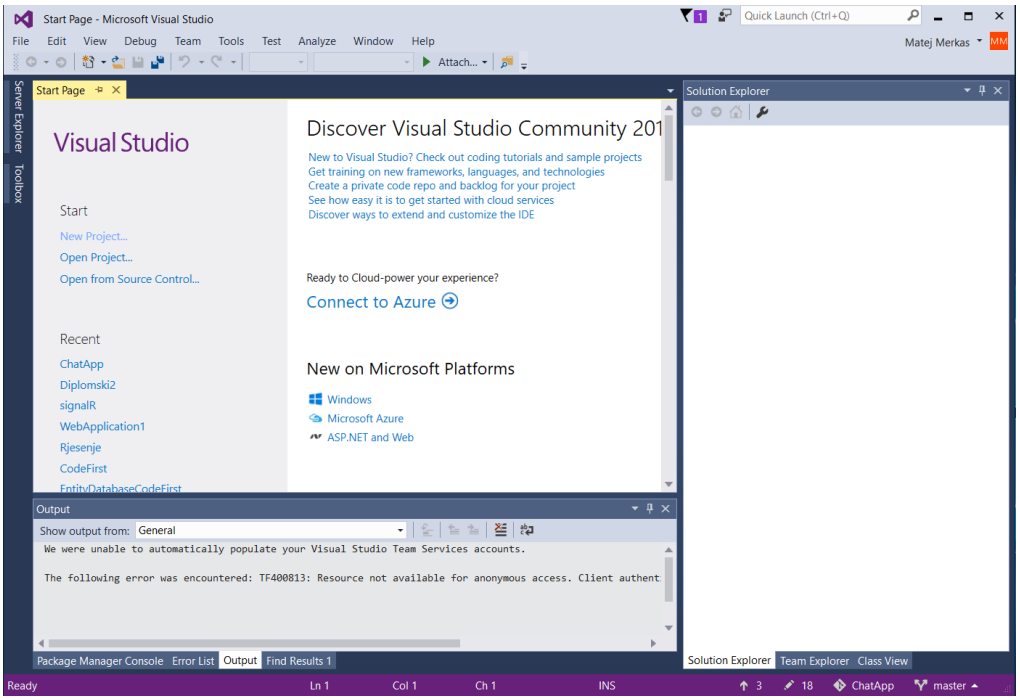
Visual Studio uključuje uređivač kôda koji podržava IntelliSense (komponentu za dovršetak koda), kao i ispravljanje koda. Ostali ugrađeni alati su code profiler, dizajn obrasci za izgradnju GUI (engl. *Graphical User Interface*) aplikacija, web dizajner, dizajner klase i dizajner sheme baze podataka. Visual Studio prihvaća dodatke koji poboljšavaju funkcionalnost na gotovo svim razinama- uključujući podršku za sustave kontrole izvornog koda i dodavanje novih alata kao što su uređivači i vizualni dizajneri, te alati za druge aspekte razvoja programske podrške.

Visual Studio podržava različite programske jezike i dozvoljava uređivaču kôda i programu za ispravljanje pogrešaka da podržava (u različitoj mjeri) skoro bilo koji programski jezik, pod uvjetom da postoji usluga specifična za jezik. Ugrađeni jezici su C, C++ i C++/CLI (preko Visual C++), VB.NET (preko Visual Basic.NET), C# (preko Visual C#) i F# (počevši od programa Visual Studio 2010). Podrška za ostale programske jezike poput M, Python i Ruby, kao i ostalih, dostupna je instalacijom jezičnih usluga zasebno. Također, podržava XML/XSLT, HTML/XHTML, Javasript i CSS. Java i J# bili su podržani u prošlosti.[1]

IDE osigurava tri usluge: SvsSolution, koji omogućava izradu projekata i rješenja; SVsUIShell, koji pruža prozore i korisničko sučelje (uključujući kartice, alatne trake i prozore alata); I SvsShel, koji se bavi registracijom Visual Studio paketa. Osim toga, IDE je također odgovaran za koordinaciju i omogućavanje komunikacije između usluga. Svi uređivači, dizajneri, vrste projekta i ostali alati implementirani su kao Visual Studio paketi.

Podrška za programske jezike dodaje se korištenjem određenog Visual Studio paketa koji se zove Language Service. Language Service definira različita sučelja koja implementacija Visual Studio paketa može implementirati kako bi se dodala podrška za različite funkcije. Funkcionalnosti koje se mogu dodati na ovaj način uključuju bojanje sintakse, završetak reda

kôda, informacije o parametrima, popise članova, i oznake za pogreške. Ako je sučelje implementirano, funkcija će biti dostupna za jezik. [1]



Slika 2.1. *Početni zaslon Visual Studio-a*

Uređivač koda

Visual Studio podržava uređivač kôda koji podržava označavanje sintakse i završetak kôda pomoću programa IntelliSense za varijable, funkcije, metode, petlje i upite. IntelliSense je podržan za sve jezike u Visual Studio-u, kao i za XML, CSS (engl. *Cascading Style Sheet*) i JavaScript prilikom izrade web stranica i web aplikacije. Prijedlozi samodovršavanja pojavljuju se u okviru složenih okvira iznad prozora za uređivanje koda u blizini kursora. [1]

Uređivač kôda Visual Studio-a podržava i postavljanje oznaka u kôdu za brzu navigaciju. Ostale navigacijske potpore uključuju inkrementalno pretraživanje, uz normalno pretraživanje teksta. Uređivač koda također uključuje i međuspremnik s više stavki i popis zadataka. Uređivač kôda podržava isječke kôda, koji su spremljeni kao predlošci koji se ponavljaju i mogu se umetnuti u kod i prilagoditi za projekt na kojem se koriste. Uređen je i alat za upravljanje isječcima kôda. Ovi alati se pojavljuju kao prozori koji se mogu postaviti da se automatski skrivaju kada se ne koriste ili da se pričvrste uz rub zaslona.

Visual Studio zadrži pozadinsku kompilaciju. Kako se kôd piše, Visual Studio ga sastavlja u pozadini kako bi pružio povratne informacije o pogreškama koje su označene crvenim valovitim

podcrtavanjem. Upozorenja su označena zelenim podcrtavanjem. Pozadinska kompilacija ne generira kôd koji se izvršava jer zahtijeva drugačiji prevoditelj nego onaj koji se rabi za generiranje izvršnog kôda. Pozadinska kompilacija u početku je uvedena uz Microsoft Visual Basic, ali je sada proširena za sve uključene jezike. [2]

Program za ispravljanje pogrešaka (engl. *Debugger*)

Visual Studio uključuje program za ispravljanje pogrešaka koji funkcionira i kao alat za ispravljanje pogrešaka na razini izvornog kôda i kao program za ispravljanje pogrešaka na razini strojnog jezika. Može se koristiti za uklanjanje pogrešaka aplikacija napisanih na bilo kojem jeziku koji podržava Visual Studio. Osim toga, može se pridružiti i pokretanju procesa, praćenju i ispravljanju procesa. Ako je dostupan izvorni kôd za proces koji se izvodi, program za ispravljanje pogrešaka prikazuje kôd dok se pokreće. Višenitni programi također su podržani. Program za ispravljanje pogrešaka može biti konfiguriran za pokretanje kada se program pokrenut izvan Visual Studio okruženja ne uspije pokrenuti.

Program za ispravljanje pogrešaka omogućava postavljanje prekidnih točaka (engl. *Breakpoints*) koje omogućuju privremeno zaustavljanje izvođenja na određenom dijelu kôda. Prekidne točke mogu biti uvjetovane, što znači da se aktiviraju kada se određeno stanje ispuni. Program za ispravljanje pogrešaka podržava uređivanje i nastavak, tj. dopušta da se kôd uređuje prilikom ispravljanja. Prilikom otklanjanja pogrešaka, ako se pokazivač miša nalazi iznad bilo koje varijable, njezina trenutna vrijednost prikazana je u posebnom alatu gdje se može po želji mijenjati. Tijekom kodiranja, Visual Studio program za ispravljanje pogrešaka omogućava određenim funkcijama ručno pozivanje iz prozora alata.

Dizajneri

Dizajner windows formi koristi se za izradu GUI aplikacija koristeći Windows Forms. Izgled se može kontrolirati tako da se kontrole ugrađuju unutar drugih spremnika ili pričvrščuju uz formu. Kontrole koje prikazuju podatke (tekstualni okvir, popis, rešetka) mogu biti vezani za izvore podataka kao što su baze podataka ili upiti. UI (engl. *user interface*) je povezan s kôdom pomoću modela programiranja temeljenog na događajima. Dizajner generira C# ili VB.NET kod za aplikaciju.

WPF dizajner, kodnog naziva Cider, uveden je u Visual Studio 2008. Kao dizajner Windows formi podržava drag and drop metodu. Upotrebljava se za kreiranje korisničkih sučelja koja ciljaju engl. *Windows Presentation Foundation*. Podržava sve WPF funkcionalnosti, uključujući

vezivanje podataka i automatsko upravljanje izgledom. Generira XAML(prezentacijski jezik napravljen od Microsoft-a) kôd za UI. Generirana XAML datoteka kompatibilna je s Microsoft Extenssion Designom, dizajnom orijentiranim proizvodom. XAML kod povezan je s kôdom pomoću code-behind modela.

Uređivač web stranica i dizajner omogućava stvaranje web stranica povlačenjem i ispuštanjem widgeta. Koristi se za razvoj ASP.NET aplikacija i podržava HTML, CSS i JavaScript. Koristi model za kodiranje koji se povezuje s ASP.NET kodom. Tu je i ASP.NET MVC podrška za MVC tehnologiju kao zasebni download i ASP.NET Dynamic Data projekt dostupan od Microsofta.

Dizajner klasa koristi se za pisanje i uređivanje klasa (uključujući njene članove i pristup) pomoću UML modela. Dizajner klasa može generirati C# i VB.NET obrasce koda za klase i metode. Također može generirati dijagrame klase iz rukom pisanih klasa.

Dizajner podataka može se koristiti za grafičko uređivanje sheme baze podataka, uključujući tablice, primarne i strane ključeve te ograničenja. Također se može koristiti za izradu upita iz grafičkog prikaza.

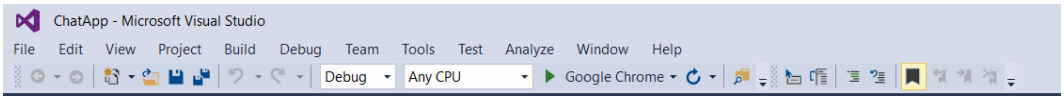
Proširivost Visual Studio-a

Visual Studio dopušta programerima da pišu proširenja za Visual Studio koja proširuju njegove mogućnosti. Ova se proširenja „uključuju“ u Visual Studio i proširuju njegovu funkcionalnost. Proširenja dolaze u obliku makronaredbi, dodataka i paketa. Makronaredbe predstavljaju ponovljive zadatke i radnje koje programeri mogu programski snimiti za spremanje, reprodukciju i distribuciju. Makronaredbe, međutim, ne mogu implementirati nove naredbe ili stvarati alatne prozore. Dodaci daju pristup objekt modelu Visual Studio-a i mogu komunicirati s IDE alatima. Dodaci se mogu koristiti za implementaciju novih funkcija i mogu dodati nove alatne prozore. Paketi se izrađuju pomoću Visual Studio SDK i pružaju najveću razinu proširivosti. Mogu stvoriti dizajnerske i druge alate, kao i integrirati druge programske jezike. Proširenja su podržana u standardnim verzijama Visual Studio-a 2005. i novijim verzijama.

Visual Studio 2008 predstavio je Visual Studio Shell koji omogućuje izradu prilagođene inačice IDE-a. Visual Studio Shell definira skup VS paketa koji pružaju funkcionalnost potrebnu u bilo kojem IDE-u. Osim toga, mogu se dodati i drugi paketi za prilagodbu instalacije. Namijenjen je razvoju prolagođenih razvojnih okruženja, bilo za određeni jezik ili za određeni scenarij.

Integrirani način instalira pakete u aplikaciju izdanja Professional/ Standard/ Team System, tako da se alati integriraju u ta izdanja. Visual Studio Shell dostupan je besplatno.

Nakon objavljivanja Visual Studio-a 2008, Microsoft je izradio Visual Studio Gallery. Ona služi kao središnje mjesto za objavljivanje podataka o proširenjima Visual Studio-a. Razvojni programeri mogu prenijeti informacije o svojim proširenjima u Visual Studio .NET 2012 kroz Visual Studio 2010. Korisnici web stranica mogu procjenjivati i pregledati proširenja kako bi procjenili kvalitetu objavljivanja proširenja. Proširenje je spremljeno u VSIX datoteci. Unutar VSIX datoteke je ZIP datoteka koja sadrži neke XML datoteke, te jedan ili više DLL-ova. Jedna od glavnih prednosti tih proširenja je da oni ne zahtijevaju instalaciju administratorskih prava. Na slici 2.2. prikazana je alatna traka u Visual Studio-u [2]



Slika 2.2. *Alatna traka u Visual Studio-u*

Izdanja

Community

Community izdanje objavljeno je 12. Studenog 2014, kao nova besplatna inačica slična funkcionalnosti Visual Studio Professional-u. Prije toga, jedina besplatna izdanja Visual Studio-a bila su ograničena Express izdanja. Za razliku od Express izdanja, Visual Studio Community podržava više jezika i pruža podršku za proširenja. Visual Studio Community izdanje usmjereno je prema pojedinačnim programerima i malim timovima.

Professional

Od Visual Studio-a 2010, Professional izdanje je komercijalno izdanje Visual Studio-a. Omogućuje IDE za sve podržane razvojne jezike. MSDN podrška dostupna je kao MSDN Essentials ili MSDN library ovisno o licenciranju. Podržava XML i XSLT uređivanje i može kreirati pakete za implementaciju koji koriste samo ClickOnce i MSI. To uključuje alate kao što su Server explorer i integraciju s Microsoft SQL Server-om. Podrška za razvoj sustava Windows

Mobile uljučena je u Visal Studio 2005 Standard, no s Visual Sudio 2008 dostupna je samo u Professional i višim izdanjima. Razvojna podrška za Windows Phone 7 dodana je svim izdanjima u Visual Studio 2010. Razvoj za Windows Mobile više nije podržan u Visual Studio-u 2010; nadomješta ga Windows Phone 7.

Enterprise

Osim značajki koje nudi Professional izdanje, Enterprise izdanje pruža novi set za razvoj softvera, baze podataka, suradnju, arhitekturu, testiranje i alate za izvješća.

Test Professional

Test Professional izdanje predstavljeno je uz Visual Studio 2010. Njegov fokus je na posvećenoj ulozi testera. Uključuje podršku za upravljanje testnim okruženjima, sposobnost pokretanja i izvješćivanja o testovima i povezivanje s Team Foundation Server-om. Ne uključuje podršku za razvoj ili stvaranje testova.

Express

Visual Studio Expres je verzija Visual Studio-a namjenjena studentima i hobistima, prvi put predstavljena uz Visual Studio 2005. Izvorno se sastojala od nekoliko izdanja, od kojih je svako usmjeravalo na jedan programski jezik. Visual Studio Express 2005, 2008 i 2010 sastojao se od sljedećih izdanja koja bi mogla biti instalirana paralelno:

- Visual Basic Express
- Visual C++ Express
- Visual C# Express
- Visual J# Express (samo 2005)
- Visual Web Developer Expres
- Visual Studio Express za Windows Phone (samo 2010)

Umjesto toga, Visual Studio 2012, 2013 i 2015 sastoje se od izdanja usmjerenih prema različitim platformama:

- Express for Web: usredotočava se na razvoj web aplikacija.
- Express for Windows: usredotočava se na razvoj aplikacija za Universal Windows Platform.

- Express for Desktop: usredotočava se na razvoj tradicionalnih aplikacija za Windows pomoću Windows API-ja.
- Team Foundation Server Express: omogućava upravljanje izvornog koda i upravljanje životnim ciklusom aplikacija.
- Express for Windows Phone (2012): usredotočava se na razvoj softvera za Windows Phone 7.5 i 8.0.

Verzije prije 2013 ne uključuju podršku za dodatke. Nakon početne najave da će izdanje Express 2012 biti ograničeno na stvaranje Windows 8 Metro aplikacija, Microsoft je reagirao na negativne povratne informacije razvojih programera preokrenuvši tu odluku i najavljujući da će podrška za razvoj aplikacija i dalje biti podržana. [2]

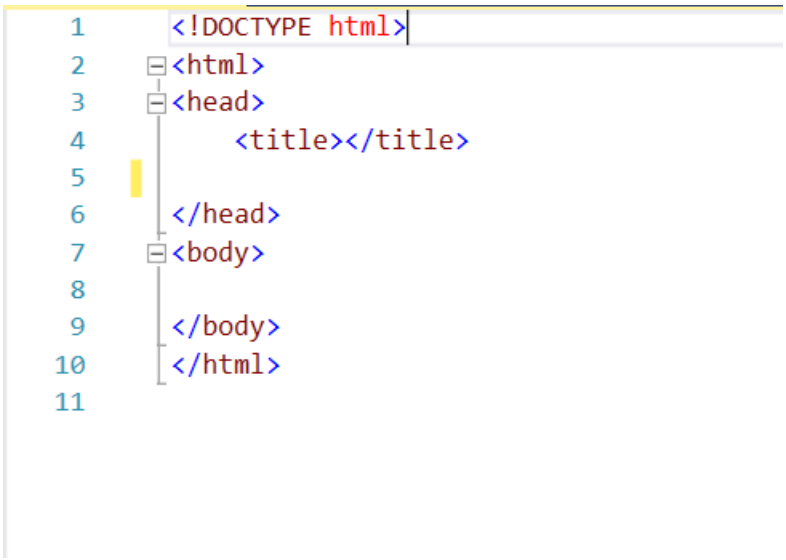
2.2. HTML

HTML (engl. *Hypertext Markup Language*) je standardni prezentacijski jezik za izradu web aplikacija i web stranica. Uz CSS (engl. *Cascading Style Sheets*) i JavaScript čini temeljnu trojku tehnologija za WWW (engl. *World Wide Web*). Internet preglednici primaju HTML dokumente s internet poslužitelja ili lokalnog spremnika i prikazuju ih kao multimedijalne internet stranice. HTML semantički opisuje strukturu internet stranice i izvorno uključene znakove za izgled dokumenta. HTML prezentacijski jezik se vrlo lako uči i jednostavan je za uporabu, a s obzirom na to i da je besplatan vrlo je raširen u upotrebi. Osnovni zadatak HTML jezika je upućivanje internet preglednika kako na najbolji način prikazati HTML dokument. S obzirom da HTML nije pravi programski jezik, njime nije moguće izvoditi nikakve zadaće, uključujući i operacije kao što su zbrajanje i oduzimanje. Ekstenzija HTML dokumenta je .htm ili .html. [3]

HTML elementi su građevni blokovi HTML stranica. Uz pomoć HTML elemenata, slike i drugi objekti, poput interaktivnih obrazaca, mogu biti ugrađeni u prikazanu stranicu. To pruža sredstvo za stvaranje strukturiranih dokumenata označavanjem strukturne semantike za tekst poput naslova, odlomaka, popisa, linkova, citata i drugih stavki. HTML elementi su obilježeni oznakama, napisanim pomoću kutnih zagrada. Oznake poput <img/> i <input/> izravno uvode sadržaj na stranicu. Druge oznake, poput ... , okružuju i pružaju informacije o tekstu dokumenta i mogu uključivati druge oznake kao podizbornike. Internet preglednici ne prikazuju HTML oznake, ali ih upotrebljavaju za tumačenje sadržaja stranice. [3]

U HTML dokument mogu se ugraditi programi napisani na skriptnom jeziku kao što je JavaScript koji utječu na ponašanje i sadržaj internet stranica. Uključivanje CSS-a definira izgled

i postavljanje sadržaja. World Wide Web Consortium (W3C), održavatelj HTML i CSS standarda, potiče korištenje CSS-a preko eksplicitnog prezentacijskog HTML-a od 1997. Na slici 2.3. prikazana je osnovna struktura HTML dokumenta. [3]



Slika 2.3. *Osnovna struktura HTML dokumenta*

HTML 5 je prezentacijski jezik koji se koristi za strukturiranje i predstavljanje sadržaja na World Wide Webu. To je peta i trenutna verzija HTML standarda. HTML 5 objavljen je u listopadu 2014. godine od World Wide Web Consortiuma(W3C) kako bi unaprijedio jezik uz podršku najnovijih multimedijskih sadržaja, čineći ga lako čitljivim korisniku i razumljivim računalu i web-preglednicima. [3]

HTML5 uključuje detaljne modele obrade radi poticanja više implementacija: proširuje, poboljšava i racionalizira oznake dostupne za dokumente, te uvodi sučelja za programiranje aplikacija (API-s) za složene web aplikacije. HTML5 također može biti koristan i pri izradi mobilnih aplikacija na različitim platformama.

U HTML5 uključene su mnoge nove sintaktičke značajke. Za uključivanje i upravljanje multimedijskim i grafičkim sadržajem dodani su novi elementi, <video>, <audio> i <canvas>, te podrška za sadržaj skalabilnih vektorskih grafika(SVG) i MathML za matematičke formule.

Kako bi obogatili semantički sadržaj dokumenta, dodani su novi elementi strukture stranice kao što su: <main>, <section>, <article>, <header>, <footer>, <aside>, <nav> i <figure>. Uvedeni su

novi atributi, neki elementi i atributi su uklonjeni, a drugi, kao što su <a>, <cite> i <menu>, promjenjeni su, redefinirani ili standardizirani.

Iako se neke značajke HTML5 često uspoređuju s Adobe Flashom, ove dvije tehnologije su vrlo raličite. Obje sadrže značajke za reprodukciju zvuka i videozapisa unutar web stranica, te za upotrebu skalabilne vektorske grafike. Međutim, HTML5 sam po sebi ne može se upotrebljavati za animaciju ili interaktivnost- mora biti dopunjen s CSS3 ili JavaSriptom. HTML5 specificira sučelja za programiranje aplikacija (API) koji se mogu koristiti s JavaScriptom.

2.3. CSS

CSS (engl. *Cascading Style Sheet*) je stilski jezik koji se koristi za opis prezentacije dokumenta napisanog prezentacijskim jezikom. Iako se najčešće koristi pri postavljanju vizualnog stila internet stranica i korisničkih sučelja napisanih u HTML-u i XHTML-u, CSS se može primjeniti na bilo koji XML dokument, uključujući običan XML, SVG i XUL. Uz HTML i JavaScript, CSS je tehnologija koju koristi većina web stranica za stvaranje vizualno zanimljivih web stranica, korisničkh sučelja za web aplikacije i korisnička sučelja za mnoge mobilne aplikacije. [4]

CSS je prvenstveno dizajniran da omogući odvajanje sadržaja dokumentra od prikaza dokumenta, uključujući aspekte kao što su izgled, boje i fontovi. Ovo razdvajanje može poboljšati dostupnost sadržaja, pružiti veću fleksibilnost i kontrolu u specifikaciji značajki prezentacije, omogućiti više HTML stranica da dijele oblikovanje određivanjem relevantnog CSS-a u zasebnoj .css datoteci i smanjiti složenost i ponavljanje u strukturnom sadržaju.

Odvajanje oblikovanja i sadržaja omogućuje prikazivanje iste prezentacijske stranice u različitim stilovima za različite načine prikazivanja. Također, može prkazati internet stranicu drugačije ovisno o veličini zaslona ili uređaju za gledanje. Korisnici mogu odrediti i drugi stilski list, kao što je .css datoteka pohranjena na vlastitom računalu, kako bi nadjačali onu koju je autor odredio. Promjene grafičkog dizajna dokumenta (ili stotina dokumenata) mogu se brzo i jednostavno primjeniti uređivnjem nekoliko redaka u CSS datoteci koju upotrebljavaju, a ne mjenjanjem označavanja u dokumentima. CSS specifikacija opisuje prioritetnu shemu kako bi se utvrdilo koja se pravila stila primjenjuju ako se više od jednog pravila podudara s određenim elementom.

CSS informacije mogu se dobiti iz različitih izvora. Ti izvori mogu biti internet preglednik, korisnik i autor. Informacije od strane autora mogu se klasificirati kao linijske, prema vrsti

medija, prema važnosti, specifičnosti selektora, pravilnoj naredbi, nasljedstvu i definiciji imovine. CSS informacije o stilu mogu biti u zasebnom dokumentu ili se mogu ugraditi u HTML dokument. Može se uvesti više stilskih listova. Različiti stilovi mogu se primjeniti ovisno o izlaznom uređaju koji se koristi. Na primjer, verzija zaslona može biti sasvim drugačija od tiskane verzije, tako da autori mogu prilagoditi prezentaciju prikladno za svaki medij. Stilski obrazac s najvišim prioritetom kontrolira prikaz sadržaja. Deklaracije koje nisu postavljene u izvoru s najvišim prioritetom prenose se na izvor nižeg prioriteta, poput stila korisničkog agenta.

Jedan od ciljeva CSS-a je omogućiti korisnicima veću kontrolu nad prezentacijom. Netko kome je teško čitati kurzivna zaglavlja može primjeniti drugi stil. Ovisno o pregledniku i web stranici, korisik može odabrati različite stilske listove koje daju dizajneri ili ukloniti sve dodane stilove i pregledavati web stranice pomoću zadanog dizajna preglednika.

CSS ima jednostavnu sintaksu i koristi brojne engleske ključne riječi kako bi odredio imena različitih stilskih svojstava. Stilski list sastoji se od popisa pravila. Svako pravilo ili skup pravila sadrži jedan ili više selektora i blok za deklaraciju.

U CSS-u, selektori određuju kojem dijelu prezentacije je pridružen određeni stil. Selektor može biti određen za:

- Sve elemente određenog tipa, npr. Zaglavlja h2.
- Elemente određene atributom: ID – identifikator jedinstven u dokumentu; class: identifikator koji može obilježavati više elemenata u dokumentu.
- Elemente ovisno o tome kako su postavljeni u odnosu na druge u stablu dokumenta.

ID i Class identifikatori razlikuju velika i mala slova, a mogu sadržavati i alfanumeričke znakove i podvlake. Class se može primjeniti na bilo koji broj primjeraka bilo kojeg elementa. ID se može primjeniti na samo jedan element.

Pseudo-klase koriste se u CSS selektorima kako bi se omogućilo oblikovanje na temelju podataka koji nisu sadržani u stablu dokumenta. Jedan primjer korištenja pseudo-klasa je :hover, koji identificira sadržaj samo kada korisnik „pokazuje“ na vidljivi element, obično držanjem pokazivača miša iznad njega.

Prije CSS-a, gotovo svi prezentacijski atributi HTML dokumenta bili su sadržani unutar HTML oznake. Sve boje fonta, stilovi, pozadine, poravnjanje elemenata, obrubi i veličine morali su biti

eksplicitno opisani, često u više navrata, unutar HTML-a. Time su dokumenti bili složeniji, veći i više skloni pogreškama i teško održivi. CSS omogućava autorima pomicanje većine tih podataka u drugu datoteku, stlski list, što rezultira jednostavnijim HTML-om. [5]

2.4. C#

C# je tipski siguran objektno orijentiran programski jezik koji programerima omogućava izradu sigurnih i robusnih aplikacija koje se pokreću na .NET okviru. C# se može koristiti da bi se stvorile aplikacije za Windows klijente, XML web usluge, distribuirane komponente, klijent-poslužitelj aplikacije, aplikacije za baze podataka i još mnogo toga. Visual C# nudi napredni program za uređivanje kôda, praktične dizajnerske programe za korisničko sučelje, integrirani program za ispravljanje pogrešaka i mnoge druge alate koji olakšavaju razvoj aplikacija na temelju C# jezika i .NET okvira. [6]

Sintaksa C# je vrlo izražajna, ali je također jednostavna i laka za učenje. Sintaksa C# je vrlo prepoznatljiva svima koji poznaju C, C++ ili Java-u. Razvojni programeri koji poznaju bilo koji od tih jezika, u vrlo kratkom vremenu obično mogu početi proizvoditi produktivno u C#. Sintaksa C# pojednostavljuje mnoge složenosti C++ -a i pruža moćne značajke kao što su vrste null vrijednosti, popisivanja, lambda izraze i izravni pristup memoriji, koje nisu pronađene u Javi. C# podržava generičke metode i tipove, koji pružaju povećanu sigurnost i performanse tipa i iteracije, koji omogućavaju implementatorima skupova za prikupljanje da definiraju ponašanja prilagođenih iteracija koje su jednostavnje za korištenje pomoću klijent- kôda. [6]

Kao jezik usmjeren na objekte, C# podržava pojmove kao što su enkapsulacija, nasljeđivanje i polimorfizam. Sve varijable i metode, uključujući glavnu (main) metodu: ulaznu točku aplikacije, su enkapsulirane unutar definicija klasa. Klasa može izravno nasljeđivati iz jedne roditeljske klase, ali može implementirati bilo koji broj sučelja. Metode koje nadjačavaju virtualne metode u roditeljskoj klasi zahtijevaju nadjačavanje ključne riječi kao način izbjegavanja slučajne redefinicije. [6]

Ukoliko postoji potreba za komunikacijom s drugim Windows programima, kao što su COM objekti ili izvorni Win32 DLL, to se može napraviti u C# kroz proces koji se zove „Interop“. Interop omogućava C# programima da rade gotovo sve što prijašnja C++ aplikacija može učiniti. C# čak podržava pokazivače i koncept „nesigurnog“ koda za one slučajeve u kojima je izravni pristup memoriji apsolutno kritičan. [6]

Izgradnja procesa (eng. *Build process*) u C# je jednostavnija u odnosu na C i C++ i fleksibilnija nego u Javi. Nema zasebnih datoteka zaglavlja i nema uvjet da se metode i tipovi podataka moraju deklarirati u određenom redoslijedu. C# datoteka može definirati bilo koji broj klasa, struktura, sučelja i događaja. [6]

C# programi se pokreću na .NET okviru, integralnoj komponenti sustava Windows koja uključuje virtualni sustav izvršavanja koji se naziva eng. Common language runtime (CLR) i jedinstven skup klasnih biblioteka. CLR je komercijalna implementacija tvrtke Microsoft za eng. Common language infrastructure (CLI), međunarodni standard koji je osnova za stvaranje okruženja za izvršenje i razvoj u kojima jezici i biblioteke neprimjetno rade zajedno. [6]

Kada se izvršava C# program, asembler je učitan u CLR, što može dovesti do različitih radnji na temelju podataka iz manifesta. Zatim, ako se zadovolje sigurnosni zahtjevi, CLR izvodi eng. Just in time (JIT) kako bi se kod pretvorio u izvorne strojne upute. CLR također pruža i druge usluge vezane uz automatsko „prikupljanje smeća“, upravljanje iznimkama i upravljanje resursima. Kod koji izvršava CLR ponekad se naziva „upravljanim kodom“, za razliku od „neupravljanog koda“ koji se kompajlira na strojni jezik koji cila određeni sustav. [6]

Jezična interoperabilnost ključna je značajka .NET okvira. Budući da je kod koji proizvod C# prevoditelj u skladu s eng. Common Product Specification (CTS), kod generiran iz C# može komunicirati s kodom generiranim iz .NET verzija Visual Basic, Viusal C++ ili bilo kojeg od više od 20 drugih CTS- kompatibilnih jezika. Jedna cjelina može sadržavati više modula napisanih na različitim jezicima. NET-a, a tipovi se mogu međusobno referirati kao da su napisani u istm jeziku.

Osim run-time usluga, .NET Framework također uključuje i veliku biblioteku od preko 4000 klasa organiziranih u prostore imena (eng. *Namespace*) koji pružaju široku paletu korisnih funkcija za sve, od unosa i izlaznih datoteka do manipulacije nizom od XML parsiranja, do kontrola sustava Windows formi. Tipična C# aplikacija koristi .NET okvir veoma opsežno za obradu poslova. [6]

2.5. AngularJS

AngularJS je open-source front-end web aplikacijski okvir temeljen na JavaScript programskom jeziku napravljen od strane Google-a i zajednice pojedinaca i korporacija, za rješavanje mnogih

iazova s kojima se susreću u razvoju single-page aplikacija. Svrha AngularJS-a je pojednostaviti razvoj i testiranje takvih aplikacija pružajući MVC okvir (eng. *Model-view-controller*) i MVVM okvir (eng. *Model-view-viewmodel*) na strani klijenta, zajedno s komponentama koje se obično koriste u bogatim internetskim aplikacijama. [7]

AngularJS okvir funkcionira tako da se prvo učitava u HTML stranicu, u koju su ugrađeni posebni tag atributi. Angular tumači te atribute kao smjernice za vezivanje ulaznih ili izlaznih dijelova stranice na model koji predstavlja standardne JavaScript varijable. Vrijednosti tih JavaScript varijabli mogu se ručno postaviti unutar koda ili se mogu preuzeti iz statičkih ili dinamičkih JSON resursa.

Ciljevi projekta AngularJS uključuju:

- Razdvojiti DOM (engl. *Document Object Model*) manipulaciju od logike aplikacije, što dramatično utječe na način strukturiranja koda.
- Odvojiti klijentsku stranu aplikacije od poslužiteljske strane. To omogućava paralelni razvoj ponovnu uporabu obje strane.
- Pružiti strukturu za izgradnju aplikacije: od izrade korisničkog sučelja, pisanja poslovne logike, do testiranja.

AngularJS koristi termin „*scope*“. Kao dio „MVC“ arhitekture, scope predstavlja *„Model“*, a svim varijablama definiranim u *scope-u* može se pristupiti kroz *„View“* i *„Controller“*. Scope se ponaša kao ljepilo i veže *„View“* i *„Controller“*. U AngularJS, scope je određena vrsta objekta koja sama može biti u ili van scope-a u bilo kojem dijelu programa, slijedeći uobičajena pravila varijable scope u JavaScriptu kao i svaki drugi objekt. [8]

2.6. SignalR

ASP.NET SignalR je biblioteka za ASP.NET programere koja pojednostavljuje proces dodavanja funkcionalnosti rada u stvarnom vremenu (eng. *Real-time functionality*) u aplikacijama. Funkcionalnost rada u stvarnom vremenu je mogućnost da kôd poslužitelja omogući sadržaj na povezane klijente odmah nakon što klijent postane dostupan, umjesto da poslužitelj čeka klijenta da zatraži nove podatke. [9]

SignalR se može koristiti za dodavanje bilo kakve funkcionalnosti rada u stvarnom vremenu u ASP.NET aplikaciju. Iako je aplikacija za razgovor(eng. *chat*) najčešći primjer korištenja, SignalR se može koristiti i za mnogo više stvari. Svaki put kada korisnik osvježi web stranicu da vidi nove podatke, ili kada stranica koristi Long polling (HTTP oblik transporta za komunikaciju između klijenta i poslužitelja) za dohvaćanje podataka, SignalR je kandidat za korištenje. SignalR se koristi i kod aplikacija za nadzor i praćenje, kao i kod aplikacija gdje je potrebno surađivanje (kao što je simultano uređivanje dokumenata), te ažuriranja napredovanja u radu. SignalR također omogućava izradu potpuno novih vrsta web aplikacija koje zahtijevaju ažuriranja sa servera veoma visokom frekvencijom, kao što su igrice u stvarnom vremenu.

SignalR automatski omogućava upravljanje vezama i omogućava emitiranje poruka istovremeno svim povezanim klijentima. Također, poruke se mogu slati i samo određenim klijentima. Veza između klijenta i poslužitelja je trajna, za razliku od klasične HTTP veze koja se ponovno uspostavlja za svaku komunikaciju. SignalR podržava *„server-push“* funkcionalnost pri čemu poslužiteljski kod može pozvati kod klijenta u pregledniku pomoću poziva udaljene procedure (RPC), za razliku od modela zahtjev-odgovor koji je najčešći na internetu.

SignalR koristi *WebSocket* transport tamo gdje je dostupan, ili koristi starije transporte tamo gdje je to potrebno. Iako bi se aplikacije mogle napisati direktno koristeći *WebSocket*, korištenje SignalR-a znači da su već omogućene mnoge funkcionalnosti koje bi se inače trebale implementirati. To znači da se aplikacija može kodirati za korištenje *WebSocket-a* bez brige o izradi posebnog koda za starije klijente. SignalR također olakšava korisniku brigu oko ažuriranja *WebSocket-a* jer se SignalR automatski ažurira pružajući aplikacijama sučelje u svim verzijama *WebSocket-a*. [9]

3. IZVEDBA SUSTAVA

Na početku, potrebno je kreirati novi ASP.NET Web API projekt u Visual Studio-u. ASP.NET Web API je okvir koji omogućava kreiranje internet aplikacija koristeći HTTP (engl. *Hyper text transfer protocol*) protokol. HTTP protokol potreban je za komunikaciju između korisničkog i programskog sučelja. ASP.NET Web Api omogućava i korištenje MVC (engl. *model-view-controller*) obrasca. MVC obrazac koristi se za odvajanje pojedinih dijelova aplikacije u komponente ovisno o njihovoj namjeni.

Model predstavlja podatke i logiku određene aplikacije, što u slučaju izrade aplikacije za komunikaciju porukama predstavlja bazu podataka. *View* predstavlja prikaz podataka, odnosno, HTML i CSS dokumente koji se prikazuju krajnjem korisniku. *Controller* upravlja korisničkim zahtjevima, što u slučaju aplikacije za komunikaciju porukama predstavlja dio programa koji krajnji korisnik ne vidi, a neophodan je za funkcioniranje aplikacije. *Controller* definira što će se događati kada krajnji korisnik koristi aplikaciju.

Prema MVC obrascu, prvo je potrebno pomoću HTML-a i CSS-a izraditi sve potrebne ekrane koji će se pojavljivati u aplikaciji. Prvi ekran je početni ekran, te sadržava formu za kreiranje novog korisnika, formu za kreiranje novog razgovora, te prema potrebi prikazuje i popis razgovora. Forma za kreiranje novog korisnika omogućava korisniku unos korisničkog imena i lozinke. Pritiskom na simbol olovke stvara se novi korisnik. Forma za kreiranje razgovora omogućava prijavljenom korisniku stvaranje novog razgovora i odabir korisnika s kojim želi uspostaviti komunikaciju. Na slici 3.1. prikazan je HTML kod za prvi ekran.

```
<div id="wrapper" ng-controller="homeController">
    <div id="menu">
        <p class="welcome">
            <!--<a> All messages</a>-->
        </p>
        <p class="welcome">
            <input ng-model="userName" />
            <input ng-model="password" type="password" />
            <span class="symbol" ng-click="login()">&#10000;</span>
            <hr>

            <input ng-model="newConversation" />
            <input ng-model="newConversationMembers" />
            <span class="symbol" ng-click="createConversation()">&#10000;</span>
            <hr>
        </p>

        <hr>
    </div>
    <div id="contact">
        <ul>
            <li ng-repeat="chat in conversations">
                <span ng-click="openConversation(chat.id)">{{chat.name}}</span>
            </li>
        </ul>
        <hr>
    </div>
```

Slika 3.1. *HTML kod za prvi ekran*

Drugi ekran je ekran koji prikazuje razgovor između korisnika. Drugi ekran sastoji se od prostora na kojemu se prikazuje razmjenjene poruke, te od forme za upisivanje i slanje nove poruke. Na slici 3.2. prikazan je HTML kod za drugi ekran.

```
<div id="wrapper2">
    <div ng-controller="chatController">

        <div id="chatbox">
            <ul>
                <li ng-repeat="chat in messages track by $index">
                    <span>{{chat}}</span>
                </li>
            </ul>

        </div>
        <form id="form">
            <input type="text" ng-model="message" name="usermsg" id="usermsg" size="62" />
            <!--  <input type="submit"  name="submitmsg" id="submitmsg" value="Send"/> -->
            <span id="envelope" ng-click="newMessage()">&#9993;</span>

        </form>
       </div>

        </div>
```

Slika 3.2. *HTML kod za drugi ekran*

Kada su ekrani realizirani u HTML-u, potrebno ih je urediti u u posebnoj CSS datoteci na koju je potrebno pozvati se u HTML dokumentu. Na slici 3.3. prikazano je pozivanje CSS datoteke u HTML dokumentu.

```
15      <link rel="stylesheet" type="text/css" href="./Templates/1ekran.css">
```

Slika 3.3. *Pozivanje CSS dokumenta u HTML dokumentu*

U CSS datoteci, elementi HTML dokumenata se uređuju prema želji programera. Svaki HTML element moguće je opisati pomoću CSS-a. Na slici 3.4. prikazano je kako su neki od HTML elemenata uređeni pomoću CSS-a.

```
#wrapper
{
    margin: 5px auto;
    padding-bottom: 25px;
    background: #f2f2f2;
    width: 600px;
    border: 4px solid #1a8cff;
    border-radius: 4px;
    height: 400px;
}
.welcome
{
    padding-left: 20px;
    font-family: "Comic Sans MS", cursive, sans serif;
    color: #1a8cff;
    padding-right: 20px;
    font-size: 20px;
    padding-top: 20px;

}
.symbol
{
    float: right;
    font-size: 25px;
}
```

Slika 3.4. *CSS dokument*

Nakon kreiranja ekrana koji će se prikazivati u aplikaciji, potrebno je omogućiti prijelaz s jednog ekrana na drugi. Prijelazi između ekrana omogućavaju se korištenjem AngularJS UI-router-a. AngularJS i UI-router potrebno je pozvati u HTML dokumentu kako bi se mogli koristiti u aplikaciji. UI-router nam omogućava prijelaz s jednog ekrana na drugi. Na slici 3.5 prikazan je JavaScript dokument u kojem je omogućen UI-router.

```
3   var routerApp = angular.module('routerApp', ['ui.router']);
4
5
6   routerApp.config(function ($stateProvider, $urlRouterProvider) {
7
8       $urlRouterProvider.otherwise('/home');
9
10      $stateProvider
11          .state('home', {
12              url: '/home',
13              templateUrl: './Templates/partial-prva.html',
14              controller: 'homeController'
15          });
16
17      $stateProvider
18          .state('write', {
19              url: '/write/:id',
20              templateUrl: './Templates/partial-druga.html',
21              controller: 'chatController'
22          });
```

Slika 3.5. *AngularJS UI-router*

U prvom redu koda navodi se kako će se u aplikaciji koristiti UI-router. UI-router podržava naredbu *State provider*. *State provider* govori koji će se ekran prikazati, što će pisati u zaglavlju preglednika kada se ekran prikaže, te na kojoj se lokaciji u projektu nalazi ekran.

Poslije kreiranja ekrana i definiranih odnosa između njih, potrebno je kreirati bazu podataka, odnosno, potrebno je definirati podatke koje koriste pojedini elementi aplikacije. Prema MVC obrascu, taj dio naziva se *Model*. Pri kreiranju aplikacije za komunikaciju kratkim porukama korišten je *Entity Framework*. *Entity Framework* je objektno- relacijski okvir koji omogućava .NET programerima rad s bazama podataka. Korištenjem *Entity Framework Code First* opcije, moguće je napraviti bazu podataka koristeći C# kod. Pri kreiranju baze podataka na takav način potrebno je programu dati do znanja da taj dio koda predstavlja bazu podataka. Na slici 3.6. prikazano je na koji način se programu daje do znanja da kreira bazu podataka.

```
3    using System;
4    using System.Collections.Generic;
5    using System.Data.Entity;
6    using System.Linq;
7
8    public class ChatApp : DbContext
```

Slika 3.6. *Slika prikazuje na koji način se programu daje do znanja da napisani kod predstavlja bazu podataka.*

Za aplikaciju za komuniciranje kratkim porukama potrebno je definirati korisnike, poruke i razgovore te pripadajuće parametre koji su potrebni za funkcioniranje. Na slici 3.7. prikazano je na koji način se kreiraju razgovori, koje parametre moraju sadržavati, te na koji način su povezani s korisnicima i porukama.

```
41   public class Conversation
42   {
43       public int Id { get; set; }
44       public string Name { get; internal set; }
45       public virtual List<User> Members { get; internal set; }
46       public virtual List<Message> Messages { get; internal set; }
47       public DateTime CreatedDate { get; internal set; }
48   }
```

Slika 3.7. *Klasa razgovor u kontekstu baze podataka s pripadajućim parametrima*

Nakon kreiranja baze podataka potrebno je prema MVC obrascu napraviti *Controller*. Jedan dio *Controllera* potrebno je napraviti na klijentskoj strani, a drugi dio na strani poslužitelja. Na klijentskoj strani potrebno je pomoću AngularJS okvira kreirati JavaScript dokument koji će omogućiti funkcionalnost ekranima. Što znači, kada krajnji korisnik pritisne tipku ili upiše tekst u formu, da se dogodi određena akcija. Svaki JavaScript dokument potrebno je inicijalizirati u HTML dokumentu u kojem se prikazuje. Kada se kreira Angular kontroler, njemu je potrebno dodijeliti ime prema kojem će HTML dokument znati da se koristi baš taj kontroler, te u kojem dijelu HTML dokumenta će se kontroler koristiti, što se može vidjeti na slici 3.8.

```
2    <div ng-controller="chatController">
```

Slika 3.8. *Inicijaliziranje AngularJS kontrolera u HTML dokumentu*

Isto tako, svaku akciju definiranu na Angular kontroleru potrebno je inicijalizirati u HTML dokumentu na mjestu na kojem će se akcija izvršavati, što je prikazano na slici 3.9.

```
15          <span id="envelope" ng-click="newMessage()">&#9993;</span>
```

Slika 3.9. *Inicijaliziranje akcije AngularJS kontrolera u HTML dokumentu. Pritiskom na simbol omotnice, napisana poruka prosljeđuje se korisniku*

Nakon što su AngularJS kontroleri inicijaliziranu HTML dokumentima, potrebno je u JavaScript dokumentima kreirati funkcije koje dovode do određene akcije. Prvi kontroler, imena HomeController sadrži funkcije potrebne za funkcioniranje prvog ekrana, na kojemu se upisuje novi korisnik, kreira novi razgovor, te ispisuju postojeći razgovori. Na slici 3.10 prikazana je funkcija za kreiranje novog korisnika.

```
14      $scope.login = function () {
15          var data = {
16              userName: $scope.userName,
17              password: $scope.password
18          };
```

Slika 3.10. *JavaScript funkcija koja omogućava korisniku unos korisničkog imena i lozinke.*

Nakon kreiranja korisnika, potrebno je i kreirati novi razgovor, što je prikazano na slici 3.11.

```
35      $scope.createConversation = function () {
36          if (!window.userId) return;
37
38          var data = {
39              userId: window.userId,
40              name: $scope.newConversation,
41              members: $scope.newConversationMembers
42          };
```

Slika 3.11. *JavaScript funkcija koja kreira novi razgovor*

Nakon što su kreirane sve funkcije potrebne za funkcioniranje prvog ekrana, potrebno je klijentsku stranu povezati s poslužiteljskom stranom i bazom podataka. Povezivanje klijentske i poslužiteljske strane odvija se putem HTTP protokola. U AngularJS kontrolerima potrebno je pomoću HTTP metoda pozivati metode na poslužiteljskoj strani koje će upravljati bazom podataka i vraćati klijentu tražene podatke ukoliko klijent to zatraži. Povezivanje se omogućava POST i GET metodama HTTP protokola, gdje POST metoda pokreće metode za upisivanje novih podataka u bazu podataka, a GET metoda šalje zahtijev poslužiteljskoj strani za podacima

iz baze podataka. Na slici 3.12. prikazano je kako klijentska strana pomoću POST metode šalje poslužiteljskoj strani podatke o novom korisniku.

```
19          $http({
20              method: 'POST',
21              url: 'api/Login',
22              headers: {
23                  'Content-Type': 'application/json',
24                  'Accept': 'application/json'
25              },
26              data: data
27          }).then(function successCallback(response) {
28              window.userId = response.data;
29              $scope.getConversations();
30
31          }, function errorCallback(response) {
32              alert("Error")
33          });
34      }
```

Slika 3.12. *POST metoda HTTP protokola koja poslužitelju šalje podatke o novom korisniku*

Nakon kreiranja HTTP metoda i funkcija na klijentskoj strani, potrebno je kreirati metode koje se pozivaju kada klijent zatraži uslugu. Kada klijent pomoću POST metode HTTP protokola pošalje poslužitelju podatke, poslužiteljska strana mora te podatke spremiti u bazu podataka. Način na koji poslužiteljska strana obrađuje POST zahtijev klijenta i sprema podatke u bazu podataka, prikazan je na slici 3.13.

```
26          [HttpPost]
27          [ResponseType(typeof(int))]
28          [Route("api/Login")]
29          public int Login([FromBody]LoginModel model)
30          {
31              using (var ctx = new ChatApp())
32              {
33                  var user = ctx.Users.FirstOrDefault(p => p.UserName == model.UserName && p.Password == model.Password);
34                  if (user == null)
35                  {
36                      user = ctx.Users.Add(new User
37                      {
38                          UserName = model.UserName,
39                          Password = model.Password
40                      });
41                      ctx.SaveChanges();
42                  }
43
44                  return user.UserID;
45              }
46          }
```

Slika 3.13. *POST metoda na poslužiteljskoj strani koja dohvaća podatke o novom korisniku i sprema ih u bazu podataka.*

Na isti način kreiraju se i POST metode za kreiranje novog razgovora. Nakon kreiranja razgovora, potrebno je na ekran ispisati listu razgovora između trenutno prijavljenih korisnika. GET metodom klijentska strana traži od poslužiteljske strane da joj omogući popis svih razgovora između korisnika. GET metoda na klijentskoj strani prikazana je na slici 3.14.

```
64          $http({
65              method: 'GET',
66              url: 'api/Conversations/' + window.userId
67          }).then(function succesCallback(response) {
68              $scope.conversations = $scope.conversations.concat(response.data);
69          }, function errorCallback(response) {
70              debugger;
71          });
72      }
```

Slika 3.14. *GET metoda na klijentskoj strani kojom klijent traži od poslužitelja da mu omogući popis razgovora.*

Isto tako, na poslužiteljskoj strani, potrebno je kreirati metode koje će se izvršiti kada klijent zatraži popis razgovora. Metode na poslužiteljskoj strani koje se pozivaju kada klijent zatraži postojeće razgovore prikazane su na slici 3.15.

```
81          [HttpGet]
82          [Route("api/Conversations/{userId}")]
83          public IEnumerable<object> Conversation(int userId)
84          {
85              using (var ctx = new ChatApp())
86              {
87                  var user = ctx.Users.Include("Conversations").FirstOrDefault(p => p.UserID == userId);
88
89                  return user.Conversations.Select(p => new
90                  {
91                      id = p.Id,
92                      name = p.Name
93                  }).ToList();
94              }
95          }
```

Slika 3.15. *Metoda na poslužiteljskoj strani koja na zahtijev klijenta vraća listu postojećih razgovora.*

Nakon što korisnik izabere jedan od postojećih razgovora, pokreće se drugi ekran. Na drugom ekranu moguće je vidjeti stare poruke ako postoje, te pisati nove poruke. Komunikacija između drugog ekrana i poslužiteljske strane odvija se pomoću SignalR-a. SignalR omogućava krajnjim korisnicima komunikaciju u stvarnom vremenu, što znači da kada jedan korisnik pošalje poruku drugom korisniku, ona će se istog trena prikazati na ekranu. Kako bi se otvorila komunikacija, potrebno je pokrenuti SignalR i na klijentskoj i na poslužiteljskoj strani.

Prije svega, SignalR je potrebno pozvati u HTML dokumentu isto kao i CSS dokument te AngularJS okvir. To je prikazano na slici 3.16.

```
7   <script src="./Scripts/jquery-1.10.2.js"></script>
8   <script src="./Scripts/jquery.signalR-2.2.2.js"></script>
9   <script src="signalr/hubs"></script>
10  <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.1/angular.min.js"></script>
11  <script src="https://cdnjs.cloudflare.com/ajax/libs/angular-ui-router/0.4.2/angular-ui-router.js"></script>
12  <script src="./App/app.js"></script>
13  <script src="./App/ChatController.js"></script>
14  <script src="~/App/HomeController.js"></script>
15  <link rel="stylesheet" type="text/css" href="./Templates/1ekran.css">
```

Slika 3.16. *Pozivanje AngularJS, SignalR, CSS, i ostalih skripti koje se koriste u aplikaciji*

Poslije pozivanja SignalR skripte u HTML dokumentu, potrebno je napraviti AngularJS kontroler za SignalR na klijentskoj i *Hub* na poslužiteljskoj strani.

Na klijentskoj strani potrebno je otvoriti SignalR *Hub* povezivanje i kreirati funkcije koje će slati i prikazivati poruke. SignalR također funkcionira na HTTP protokolu, ali kao primaran transport koristi *Web Socket.* Funkcija za otvaranje *Hub* povezivanja prikazana je na slici 3.17.

```
12      $scope.chatHub = $.connection.chatHub;
```

Slika 3.17. *otvaranje Hub povezivanja na klijentskoj strani*

Kada je konekcija otvorena, krajnji korisnici mogu komunicirati nesmetano u stvarnom vremenu. Na klijentskoj strani potrebno je napisati JavaScript funkcije koje će slati i prikazivati poruke na ekranu, što je pokzano na slici 3.18.

```
13  $scope.chatHub.client.broadcastMessage = function (name, message) {
14      var newMessage = name + ':' + message;
15
16      $scope.messages.push(newMessage);
17      $scope.$apply();
18  };
19  $scope.newMessage = function () {
20      $scope.chatHub.server.newMessage(window.userId, $stateParams.id, $scope.message);
21      $scope.message = '';
22  };
```

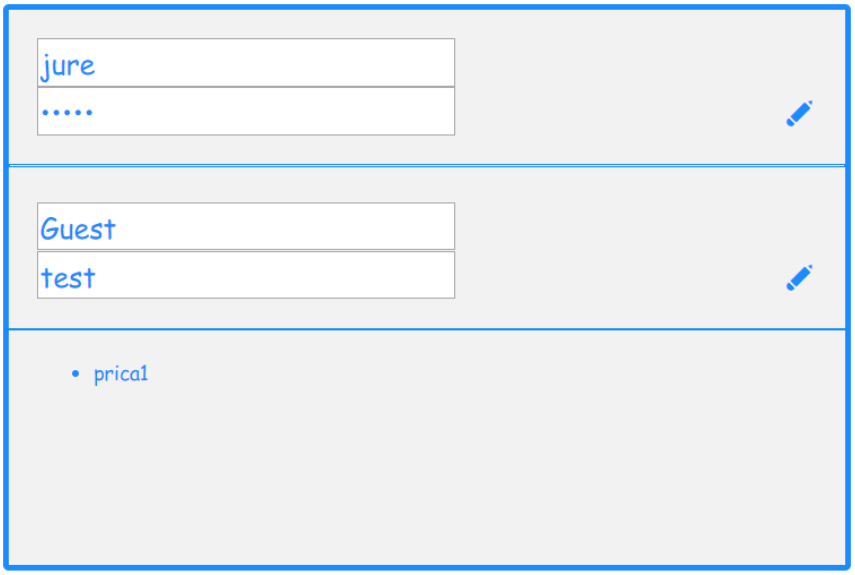
Slika 3.18. *Funkcije za slanje nove poruke.*

Nakon što su kreirane sve funkcije za komunikaciju na strani klijenta, potrebno je kreirati i metode na poslužiteljskoj strani koje će se pokretati kada je SignalR *Hub* konekcija pokrenuta. Kod koji se pokreće na poslužiteljskoj strani kada postoji SignalR konekcija prikazan je na slici 3.19. Poslužiteljska strana omogućava kreiranje nove poruke te spremanje iste u bazu podataka.

```
18      public class ChatHub : Hub
19      {
20          public void Send(string name, string message)
21          {
22              Clients.All.broadcastMessage(name, message);
23          }
24
25          public void NewMessage(int userId, int conversationId, string message)
26          {
27              using (var ctx = new ChatApp())
28              {
29                  var user = ctx.Users.Find(userId);
30                  var conversation = ctx.Conversations.Include("Members").Include("Messages").FirstOrDefault
31                      (p => p.Id == conversationId);
32
33                  if (conversation == null)
34                      return;
35
36                  conversation.Messages.Add(new Message {
37                      CreatingDate = DateTime.Now,
38                      Text = message,
39                      User = user
40                  });
41                  ctx.SaveChanges();
```

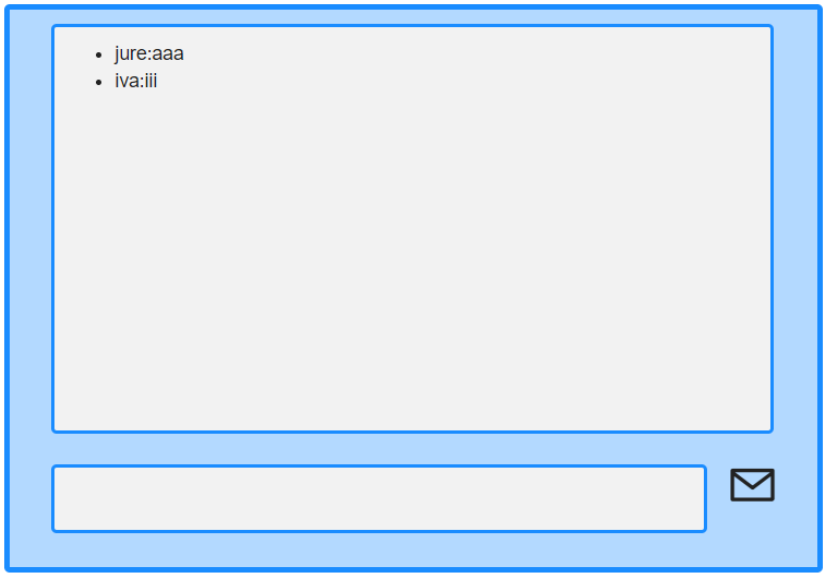
Slika 3.19. *Dio koda koji se izvršava na poslužiteljskoj strani nakon što je SignalR konekcija pokrenuta.*

Prilikom pokretanja aplikacije prikazuje se prvi ekran na kojem je omogućeno registriranje novog korisnika, kreiranje novog razgovora, te pristup jednom od već kreiranih razgovora. Kada se pokrene jedan od razgovora, prikazuje se drugi ekran na kojem se odvija razgovor. Na slici 3.20. prikazan je izgled prvog ekrana, gdje je prikazan prijavljeni korisnik, forma za kreiranje novog razgovora, te popis razgovora s prijavljenim korisnikom.



Slika 3.20. *Izgled prvog ekrana*

Nakon odabira razgovora, ili prilikom kreiranja novog razgovora, pokreće se drugi ekran na kojem je moguće pisati nove poruke i vidjeti prethodne. Izgled drugog ekrana može se vidjeti na slici 3.21.



Slika 3.21. *Izgled drugog ekrana*

4. ZAKLJUČAK

U ovom diplomskom radu prikazan je postupak izrade internet aplikacije za komunikaciju kratkim porukama. Cijeli postupak je izrađen u programu Visual Studio. Za izradu aplikacije potrebno je poznavanje višerazinske klijent - poslužitelj arhitekture, .NET okvira, C# programskog jezika, te rad s bazom podataka. Također, potrebno je i poznavanje HTML-a, CSS-a te AngularJS okvira, odnosno JavaScript funkcija.

Prednost izrade ovakvog sustava je korištenje Visual Studio platforme koja omogućava programiranje u svim navedenim programskim jezicima, njihovo međusobno povezivanje, te veoma dobru preglednost koda. Isto tako Visual Studio sadrži IntelliSense koji na vrlo jednostavan način omogućava programeru ispravak i pronalazak pogrešaka. Osim u navedenim programskim jezicima, Visual Studio omogućava programiranje i u mnogim drugim programskim jezicima kao što su C, C++ i drugi.

Velika prednost izrade ovakve internet aplikacije je mogućnost učenja raznih tehnologija koje su veoma popularne u poslovnom svijetu, te poznavanje istih povećava konkurentnost programera na tržištu rada. Izrada ovakve internet aplikacije zahtijeva od programera dobro poznavanje navedenih tehnologija te načine njihovog povezivanja.

U ovom radu napravljena je internet aplikacija za komunikaciju kratkim porukama koja omogućava različitim korisnicima međusobno povezivanje i komunikaciju. S obzirom da je komunikacija između korisnika ostvarena, zadatak se može smatrati uspješno obavljenim. Funkcionalnosti kao što su prijava korisnika, izrada novog razgovora, otvaranje postojećeg razgovora su ostvareni. Uvid u poruke iz postojećih razgovora također je implementirano.

LITERATURA

[1] Visual Studio razvojno okruženje, https://msdn.microsoft.com/en-us/library/fx6bk1f4(v=vs.90).aspx , pristup ostvaren 17.3.2017.

[2] Visual Studio razvojno okruženje, https://en.wikipedia.org/wiki/Microsoft_Visual_Studio , pristup ostvaren 17.3.2017

[3] HTML prezentacijski jezik, https://en.wikipedia.org/wiki/HTML, pristup ostvaren 12.4.2017.

[4] CSS stilski jezik, https://hr.wikipedia.org/wiki/CSS , pristup ostvaren 12.4.2017.

[5] CSS stilski jezik, https://en.wikipedia.org/wiki/Cascading_Style_Sheets , pristup ostvaren 12.4.2017.

[6] C# programski jezik, https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework , pristup ostvaren 29.5.2017.

[7] AngularJS okvir, https://docs.angularjs.org/guide/introduction , pristup ostvaren 1.6.2017.

[8] AngularJS okvir, https://en.wikipedia.org/wiki/AngularJS , pristup ostvaren 1.6.2017.

[9] SignalR biblioteka, https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr , 15.6.2017

SAŽETAK

Naslov: Izrada internet aplikacije za komunikaciju kratkim porukama

U ovom radu, prikazana je izvedba i rad jednostavne internet aplikacije za komunikaciju kratkim porukama. Glavna zadaća je slanje poruka između korisnika, kreiranje novih razgovora te pokretanje postojećih razgovora. Komunikacija između klijentske i poslužiteljske strane odvija se putem HTTP protokola. Za slanje poruka, koristi se SignalR biblioteka. Cijela aplikacija izrađuje se u Visual Studio-u korištenjem sljedećih programskih jezika, biblioteki i okvira: HTML, CSS, C#, AngularJS, SignalR.

Ključne riječi:

Internet aplikacija, kratke poruke, Visual Studio, HTML, CSS, C#, AngularJS, SignalR

ABSTRACT

Title: Chat Web application

This paper shows the construction and operation of a simple web application for communication with short messages. Main task is sending messages between users, creating new conversations and opening existing conversations. The communication between client side and server side is using HTTP protocol. SignalR library was used for sending messages. The whole application is made in Visual Studio, using next programming languages, libraries, and frameworks: HTML, CSS, C#, AngularJS, SignalR.

Keywords:

Web application, short messages, Visual Studio, HTML, CSS, C#, AngularJS, SignalR

ŽIVOTOPIS

Matej Merkaš rođen je u Osijeku, 4. Studenog 1991. Godine. U Petrijevcima je završio Osnovnu školu Petrijevci. Nakon Osnovne škole upisuje Prirodoslovno- matematičku gimnaziju u Osijeku. Nakon završene Srednje škole, upisuje Preddiplomski studij Računarstva u Osijeku. Po završetku Preddiplomskog studija, upisuje Diplomski studij Elektrotehnike, smjer Procesno računarstvo.

PRILOG A. Osnovni HTML program

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
    <script src="./Scripts/jquery-1.10.2.js"></script>
    <script src="./Scripts/jquery.signalR-2.2.2.js"></script>
    <script src="signalr/hubs"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.1/angular.min.js"></script
>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular-ui-
router/0.4.2/angular-ui-router.js"></script>
    <script src="./App/app.js"></script>
    <script src="./App/ChatController.js"></script>
    <script src="~/App/HomeController.js"></script>
    <link rel="stylesheet" type="text/css" href="./Templates/1ekran.css">
</head>
<body ng-app="routerApp">
    <nav class="navbar navbar-inverse" role="navigation">
        <div class="navbar-header">
            <a class="navbar-brand" ui-sref="#">Chat app</a>
        </div>
        <ul class="nav navbar-nav">
            <li><a ui-sref="home">Home</a></li>
            <li><a ui-sref="write">Write</a></li>
        </ul>
    </nav>


    <div ui-view></div>


</body>
</html>
```

PRILOG B. HTML program za ekran za prijavu

```
<div id="wrapper" ng-controller="homeController">
    <div id="menu">
        <p class="welcome">
            <!--<a> All messages</a>-->
        </p>
        <p class="welcome">
            <input ng-model="userName" />
            <input ng-model="password" type="password" />
            <span class="symbol" ng-click="login()">&#10000;</span>
            <hr>
        </p>
        <p class="welcome2">
            <input ng-model="newConversation" />
            <input ng-model="newConversationMembers" />
            <span class="symbol" ng-click="createConversation()">&#10000;</span>
            <hr>
            </p>


</div>
    <div id="contact">
        <ul>
            <li ng-repeat="chat in conversations">
                <span class="list" ng-
click="openConversation(chat.id)">{{chat.name}}</span>
            </li>
        </ul>

    </div>


</div>
```

PRILOG C. HTML program za ekran za razgovor

```
<div id="wrapper2">
    <div ng-controller="chatController">

        <div id="chatbox">
            <ul class="conver">
                <li ng-repeat="chat in messages track by $index">
                    <span>{{chat}}</span>
                </li>
            </ul>

        </div>
        <form id="form">
            <input type="text" ng-model="message" name="usermsg" id="usermsg"
size="62" />
            <!--  <input type="submit"  name="submitmsg" id="submitmsg"
value="Send"/> -->
            <span id="envelope" ng-click="newMessage()">&#9993;</span>

        </form>
       </div>

        </div>
```

PRILOG D. CSS program

```
body
 {
       color: #222;



}

#wrapper
{
       margin: 5px auto;
       padding-bottom: 2px;
       background: #f2f2f2;
       width: 600px;
       border: 4px solid #1a8cff;
       border-radius: 4px;
       height: 400px;
}
.welcome
{
       padding-left: 20px;
       font-family: "Comic Sans MS", cursive, sans serif;
       color: #1a8cff;
       padding-right: 20px;
    font-size: 20px;
    padding-top: 5px;

}
.welcome2
{
       padding-left: 20px;
       font-family: "Comic Sans MS", cursive, sans serif;
       color: #1a8cff;
       padding-right: 20px;
    font-size: 20px;
    padding-top: 5px;

}
.symbol
{
       float: right;
       font-size: 25px;
    cursor: pointer;

}

hr
{

       border: 1px solid #1a8cff;
}

#contact
{
```

```
        padding-left: 20px;
        font-family: "Comic Sans MS", cursive, sans serif;
    color: #1a8cff;

}

h4
{
        margin: 0em;
}

/*//partial-druga*/

#wrapper2 {
    margin: 5px auto;
    padding-bottom: 25px;
    background: #b3d9ff;
    width: 600px;
    border: 4px solid #1a8cff;
    border-radius: 4px;
}

#chatbox {
    margin: 0px auto;
    margin-bottom: 10px;
    margin-left: 30px;
    margin-top: 10px;
    padding-bottom: 5px;
    padding-left: 5px;
    padding-top: 10px;
    background: #fff;
    height: 300px;
    width: 530px;
    border: 2px solid #1a8cff;
    border-radius: 4px;
    overflow: auto;
    background-color: #f2f2f2;
}

#form
{
        padding-left: 30px;


}

#usermsg
{
        border: 2px solid #1a8cff;
        height: 50px;
        border-radius: 4px;
        background-color: #f2f2f2;
}
#envelope
{
        font-size: 40px;
        margin-left: 10px;
    cursor: pointer;
}
```

```
#forma
{
	padding-left: 30px;
	margin: 20px auto;

}

#usernm
{
	border: 2px solid #1a8cff;
	height: 20px;
	border-radius: 4px;
	background-color: #f2f2f2;
}

/*partial-treca*/

.welcome1
{
	padding-left: 40px;
	font-family: "Comic Sans MS", cursive, sans serif;
    padding-top: 20px;
}
.list
{
    cursor: pointer;
}
```

PRILOG E. JavaScript program za funkcioniranje razgovora

```
var routerApp = angular.module('routerApp');

routerApp.controller('chatController', ['$scope', '$http', '$stateParams', function
($scope, $http, $stateParams) {

    $scope.password = 'Guest';

    $scope.groupName = 'Guest';
    $scope.message = '';

    $scope.messages = [];

    $scope.chatHub = $.connection.chatHub;
    $scope.chatHub.client.broadcastMessage = function (name, message) {
        var newMessage = name + ':' + message;

        $scope.messages.push(newMessage);
        $scope.$apply();
    };
    $scope.newMessage = function () {
        $scope.chatHub.server.newMessage(window.userId, $stateParams.id,
$scope.message);
        $scope.message = '';
    };

    $scope.getConversations = function () {
        if (!window.userId) return;

        $http({
            method: 'GET',
            url: 'api/Messages/' + $stateParams.id
        }).then(function succesCallback(response) {
            response.data.forEach(function (d) {
                $scope.messages.push(d.user + ':' + d.text);
            });

            $.connection.hub.start().done(function () {
                $scope.chatHub.server.connect(window.userId, $stateParams.id);
            });

        }, function errorCallback(response) {
            debugger;
        });
    }
    $scope.getConversations();
}]);
```

PRILOG F. JavaScript program za prijavljivanje korisnika

```
var routerApp = angular.module('routerApp');

routerApp.controller('homeController', ['$scope', '$http', '$location', function
($scope, $http, $location) {

    $scope.userName = 'Guest';
    $scope.password = 'Guest';

    $scope.newConversation = 'Guest';
    $scope.newConversationMembers = 'test';

    $scope.conversations = [];


    $scope.login = function () {
        var data = {
            userName: $scope.userName,
            password: $scope.password
        };
        $http({
            method: 'POST',
            url: 'api/Login',
            headers: {
                'Content-Type': 'application/json',
                'Accept': 'application/json'
            },
            data: data
        }).then(function successCallback(response) {
            window.userId = response.data;
            $scope.getConversations();

        }, function errorCallback(response) {
            alert("Error")
        });
    }
    $scope.createConversation = function () {
        if (!window.userId) return;

        var data = {
            userId: window.userId,
            name: $scope.newConversation,
            members: $scope.newConversationMembers
        };
        $http({
            method: 'POST',
            url: 'api/Conversation',
            data: data
        }).then(function successCallback(response) {
            window.conversationId = response.data;

            $scope.conversations.push({ id: response.data, name:
$scope.newConversation });
            $scope.openConversation(response.data);
        }, function errorCallback(response) {
```

```
            alert("Error")
        });
    }

    $scope.openConversation = function (id) {
        $location.path('/write/'+id);
    }

    $scope.getConversations = function () {
        if (!window.userId) return;

        $http({
            method: 'GET',
            url: 'api/Conversations/' + window.userId
        }).then(function succesCallback(response) {
            $scope.conversations = $scope.conversations.concat(response.data);
        }, function errorCallback(response) {
            debugger;
        });
    }
}]);
```

PRILOG G. C# program za prijavu korisnika i kreiranje razgovora

```
using App.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using System.Web.Http.Description;

namespace App.Controllers
{
    public class ValuesController : ApiController
    {
        // GET api/values
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }

        public class LoginModel
        {
            public string UserName { get; set; }
            public string Password { get; set; }
        }

        [HttpPost]
        [ResponseType(typeof(int))]
        [Route("api/Login")]
        public int Login([FromBody]LoginModel model)
        {
            using (var ctx = new ChatApp())
            {
                var user = ctx.Users.FirstOrDefault(p => p.UserName == model.UserName
&& p.Password == model.Password);
                if (user == null)
                {
                    user = ctx.Users.Add(new User
                    {
                        UserName = model.UserName,
                        Password = model.Password
                    });
                    ctx.SaveChanges();
                }

                return user.UserID;
            }
        }


        public class ConversationModel
        {
            public string Name { get; set; }
            public int UserId { get; set; }
            public string Members { get; set; }
```

```
        }

        [HttpPost]
        [Route("api/Conversation")]
        public int Conversation([FromBody]ConversationModel model)
        {
            using (var ctx = new ChatApp())
            {
                var memberNames = model.Members.Split(';');
                var newConversation = new Conversation
                {
                    Name = model.Name,
                    Members = new List<User>(),
                    CreatedDate = DateTime.Now
                };
                var members = ctx.Users.Where(p =>
memberNames.Contains(p.UserName)).ToList();
                var user = ctx.Users.Find(model.UserId);
                members.Add(user);
                newConversation.Members = members;

                ctx.Conversations.Add(newConversation);
                ctx.SaveChanges();

                return newConversation.Id;
            }
        }

        [HttpGet]
        [Route("api/Conversations/{userId}")]
        public IEnumerable<object> Conversation(int userId)
        {
            using (var ctx = new ChatApp())
            {
                var user = ctx.Users.Include("Conversations").FirstOrDefault(p =>
p.UserID == userId);

                return user.Conversations.Select(p => new
                {
                    id = p.Id,
                    name = p.Name
                }).ToList();
            }
        }

        [HttpGet]
        [Route("api/Messages/{conversationId}")]
        public IEnumerable<object> Message(int conversationId)
        {
            using (var ctx = new ChatApp())
            {
                var conversation =
ctx.Conversations.Include("Messages").Include("Messages.User").FirstOrDefault(p =>
p.Id == conversationId);

                return conversation.Messages.Select(p => new
                {
                    id = p.MessageID,
                    text = p.Text,
```

```
                    user = p.User.UserName
                }).ToList();
            }
        }
    }
}
```

PRILOG H. C# program za funkcioniranje razgovora

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using Microsoft.AspNet.SignalR;
using App.Models;

namespace App.Hubs
{
    public class ConnectedUsers {
        public string ConnectionId { get; set; }
        public string Name { get; set; }
        public List<User> Recipients { get; set; }
        public int UserId { get; internal set; }
    }

    public class ChatHub : Hub
    {
        public void Send(string name, string message)
        {
            Clients.All.broadcastMessage(name, message);
        }

        public void NewMessage(int userId, int conversationId, string message)
        {
            using (var ctx = new ChatApp())
            {
                var user = ctx.Users.Find(userId);
                var conversation =
ctx.Conversations.Include("Members").Include("Messages").FirstOrDefault(p => p.Id ==
conversationId);

                if (conversation == null)
                    return;

                conversation.Messages.Add(new Message {
                    CreatingDate = DateTime.Now,
                    Text = message,
                    User = user
                });
                ctx.SaveChanges();

                Clients.Group(conversation.Name).broadcastMessage(user.UserName,
message);
            }
        }

        public void Connect(int userId, int conversationId) {
            using (var ctx = new ChatApp()) {
                var conversation = ctx.Conversations.Where(p => p.Members.Any(d =>
d.UserID == userId)).FirstOrDefault(p => p.Id == conversationId);

                if (conversation == null)
```

```
                    return;

                Groups.Add(Context.ConnectionId, conversation.Name);
            }
        }
    }
}
```

PRILOG I. C# program za bazu podataka

```
namespace App.Models
{
    using System;
    using System.Collections.Generic;
    using System.Data.Entity;
    using System.Linq;

    public class ChatApp : DbContext
    {

        public ChatApp()
            : base("name=ChatApp")
        {
        }

        public DbSet<User> Users { get; set; }
        public DbSet<Conversation> Conversations { get; set; }
        public DbSet<Message> Messages { get; set; }
    }

    public class User
    {
        public int UserID { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string IsActive { get; set; }

        public virtual List<Conversation> Conversations { get; set; }
        public virtual List<Message> Messages { get; set; }
        public string UserName { get; internal set; }
        public string Password { get; internal set; }

        public User()
        {
            this.Conversations = new List<Conversation>();
            this.Messages = new List<Message>();
        }

    }

    public class Conversation
    {
        public int Id { get; set; }
        public string Name { get; internal set; }
        public virtual List<User> Members { get; internal set; }
        public virtual List<Message> Messages { get; internal set; }
        public DateTime CreatedDate { get; internal set; }
    }

    public class Message
    {
        public int MessageID { get; set; }
        public string Subject { get; set; }
        public string MessageBody { get; set; }
```

```
        public DateTime CreatingDate { get; set; }

        public virtual User User { get; set; }
        public string Text { get; internal set; }
    }

}
```