UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Žiga Zupan

Primerjava razvoja spletnih aplikacij v Visual Studio z Web Forms in MVC

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Damjan Vavpotič

Ljubljana, 2013

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00380 / 2013
Datum: 2.4.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ŽIGA ZUPAN**

Naslov: **PRIMERJAVA RAZVOJA SPLETNIH APLIKACIJ V VISUAL STUDIO Z WEB FORMS IN MVC**
COMPARISON OF WEB DEVELOPMENT IN VISUAL STUDIO WEB FORMS AND MVC

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V okviru diplomskega dela najprej predstavite Microsoft-ovo arhitekturo MVC, nato pa se posvetite zlasti primerjavi MVC z Web Forms. Primerjavo podkrepite z razvojem manjše spletne aplikacije na oba načina. Kritično analizirajte in primerjajte tako potek razvoja z uporabo MVC in z uporabo Web Forms kot tudi spletni aplikaciji izdelani v obeh arhitekturah.

Mentor:

doc. dr. Damjan Vavpotič

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O AV T O R S T V U

diplomskega dela

Spodaj podpisani/-a **Žiga Zupan**
z vpisno številko **63090321**

sem avtor diplomskega dela z naslovom:

Primerjava razvoja spletnih aplikacij v Visual Studio z Web Forms in MVC

S svojim podpisom zagotavljam, da:

- sem diplomsko nalogo izdelal/a samostojno pod mentorstvom

  doc. dr. Damjana Vavpotiča

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja: _____

Zahvala

Iskreno bi se rad zahvalil mentorju doc. dr. Damjan Vavpotiču za vse nasvete in vodenje pri izdelavi diplomske naloge, ter za vse spodbude, ki sem jih bil pod njegovim mentorstvom deležen.

Zahvalil bi se rad tudi svojim staršem, brez katerih mi v nasprotnem primeru ne bi uspelo obiskovati in opravljati študija v Ljubljani.

Posebna zahvala gre tudi starima staršema, ki sta mi ves čas pisanja diplomske naloge stala ob strani, me spodbujala in nagovarjala k čim hitrejši izdelavi mojega diplomskega dela.

Prav tako pa bi se rad zahvalil vsem sošolcem, sošolkam in prijateljem, ki so mi v letih študija pomagali na kakršen koli način, ter za vso pomoč in podporo, ki ste mi jo namenili pri izdelavi diplomske naloge.

Kazalo vsebine:

Kazalo slik:

Kazalo tabel:

Razlaga kratic

MVC – Model-View-Controller. Arhitekturni model, po katerem izdelujemo spletne strani.

ASP – Active Server Pages. Microsoftov programski jezik, ki se uporablja za izdelavo spletnih strani

HTML – Hyper Text Markup Language. To je označevalni jezik, s pomočjo katerega izdelujemo spletne strani. Tvori osnovno zgradbo spletne strani.

jQuery – Napredna knižnjica vnaprej spisanih JavaScript funkcij.

SQL – Structured Query Language. To je strukturni povpraševalni jezik, ki se uporablja za delo s podatkovnimi bazami in njihovimi zapisi.

IDE – Integrated Development Environment. Skupek orodij, znotraj razvijalskega okolja, ki omogoča urejanje, prevajanje in preprečevanje napak (razhroščevanje) programske kode.

IIS – Internet Information Services. To je spletni strežnik za poganjanje aplikacij spisanih s pomočjo Microsoftovih programskih jezikov.

.NET – Gre za programsko komponento sistema Microsoft Windows. Vsebuje orodja in knjižnice, katere so namenjene za razvoj aplikacij.

API – Application Programming Interface. Skupek protokolov in orodij, ki služijo za razvijanje aplikacij.

CSS – Cascading Style Sheets. To je označevalni jezik, ki določa izgled spletne strani v brskalniku.

AJAX – Asynchronous JavaScript and XML. Asinhroni JavaScript in XML, za pridobivanje podatkov na dinamičnih spletnih straneh.

URL – Uniform Resource Locator. Spletni naslov, ki določa enolični naslov spletne aplikacije v svetovnem spletu.

WF – Okrajšava za Web Forms. To je način izdelave spletnih strani s pomočjo dinamičnih strani.

JS – Okrajšava za JavaScript. Gre za skriptni programski jezik, ki omogoča interakcijo uporabnika z gradniki na spletnih straneh in za razvoj dinamičnih spletnih strani.

Povzetek

V diplomski nalogi bom predstavil Microsoft-ovo MVC arhitekturo in razložil samo sestavo Model – View – Controller (MVC). Pokazal bom, kako se različni elementi med seboj povezujejo in na koncu tvorijo celoten model spletne aplikacije. Pobližje vas bom seznanil s posameznimi strukturami same zgradbe MVC modela, kako se je zgodovina le-tega čez čas spreminjala, ter kateri so bili ključni mejniki novih različic MVC-ja.

Opisal bom razlike med uporabo MVC modela in Web Forms modela. Hkrati pa vam bom predstavil, kako zgraditi osnovno spletno stran s pomočjo C#, HTML 5, JavaScript, jQuery tehnologij z uporabo SQL baze, ter kako se povezuje CSS s samim videzom spletne aplikacije. Za primerjavo bom izdelal podobno spletno stran s pomočjo Web Forms modela, ob tem pa bom predstavil ključne razlike, ki se pojavljajo med uporabo MVC spletne aplikacije in Web Forms spletne aplikacije. Za zaključek pa vam bom predstavil še nekaj lastnosti, ki so vidne med delovanjem obeh izdelanih aplikacij.

Ključne besede: MVC, Web Forms, izdelava, primerjava, razlike, lastnosti, uporaba, spletna stran, spletna aplikacija

Abstract

The thesis represents the Microsoft MVC arhitecture and explains its structure Model – View – Controller. I will show how the different elements interact with each other and eventually form a complete model of the web application. I will take a closer look at individual structures, which form the MVC model and show how the history of development changed over time and what the key milestones of new versions of MVC were.

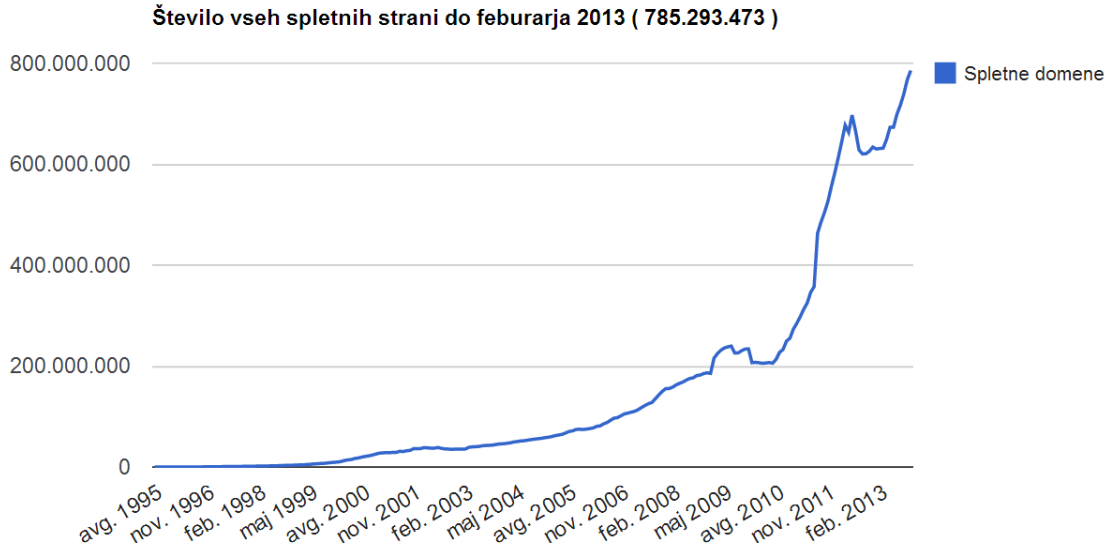
I will also describe the differences between using MVC model and Web Forms application. At the same time I am going to build a basic website using C #, HTML 5, JavaScript, jQuery technologies and a SQL database, and how the CSS styles relate to the final appearance of the web applications. For comparison I will make a similar website using Web Forms, and explain the key differences that occur while using MVC model and Web Forms web application. At the end I will introduce some features, which are visible while using both applications.

Keywords: MVC, Web Forms, making, comparison, differences, properties, usage, web page, web application

1 Uvod

Dandanes lahko že vsak povprečen računalniški uporabnik izdela preprosto spletno stran, bodisi v beležnici ali pa s pomočjo programskih orodij. Zaradi dostopnosti nasvetov in primerov uporabe izdelave spletnih strani na spletu je posledica vse večje število spletnih strani.

Na spodnji sliki (slika 1) je prikazana rast registriranih spletnih domen od leta 1995 do danes. Graf je izrisan s pomočjo pridobljenih podatkov, katera so bila pridobljena s strani podjetja Netcraft [1]. Zbrane podatke so pridobili s pomočjo postavljenih vprašalnikov in pridobljenih odgovorov na spletnih domenah.



Slika 1: Prikaz rasti spletnih strani

Zaradi povečanja izdelave spletnih strani so razvijalci prisiljeni razvijalcem ponuditi nove načine izdelave spletnih aplikacij. Nove tehnologije prispevajjo k boljši učinkovitosti spletnih aplikacij, lažjemu načinu izdelave, nove načine prikaza, poenostavijo obstoječe in omogočajo razvijanje novih funkcionalnosti aplikacije. Razvijalci programskih orodij za izdelavo spletnih strani se trudijo čim bolj olajšati delo uporabnika in tehnologija nas je pripeljala že do tega, da lahko enostavno spletno stran izdelamo že z nekaj kliki.

Zadnje čase je zelo popularen koncept izdelave aplikacij s pomočjo MVC modela. Ne gre za novejši koncept, saj je bil zasnovan že davno, a je v izraziti rabi šele zadnjih nekaj let. Razvijalcu ponuja proste roke pri vključitvi različnih tehnologij v samo aplikacijo, ne da bi vključene knjižnice kakorkoli vplivale na samo delovanje modela. Z ločenimi komponentami pa skrbi za neodvisno delovanje določenih komponent, s čimer je omogočena najboljša učinkovitost delovanja hkrati pa omogoča ločeno razvijanje funkcionalnosti za določene komponente.

Ta dejstva pa so tudi glavni temelji za izdelavo naše diplomske naloge. Namen le te je izdelati preprosto spletno stran s pomočjo sodobnih tehnologij in uporabo Microsoft MVC modela, ter prikazati, kako lahko podobno spletno stran izdelamo še s pomočjo Web Forms modela, ob tem pa primerjati razlike, ki so se pojavljale med razvojem obeh spletnih aplikacij.

Uporabljali bomo najnovejše Microsoft Visual Studio razvijalsko okolje. Najnovejša verzija pa nam tudi ponuja asinhronski programski model (angl. asynchronous programming model), kar pohitri samo izvajanje kode, saj se le ta izvaja po več delčkov hkrati. Primarno se bomo osredotočili na MVC verzijo 4 z .NET 4.5 različico v primerjavi z Web Forms aplikacijo.

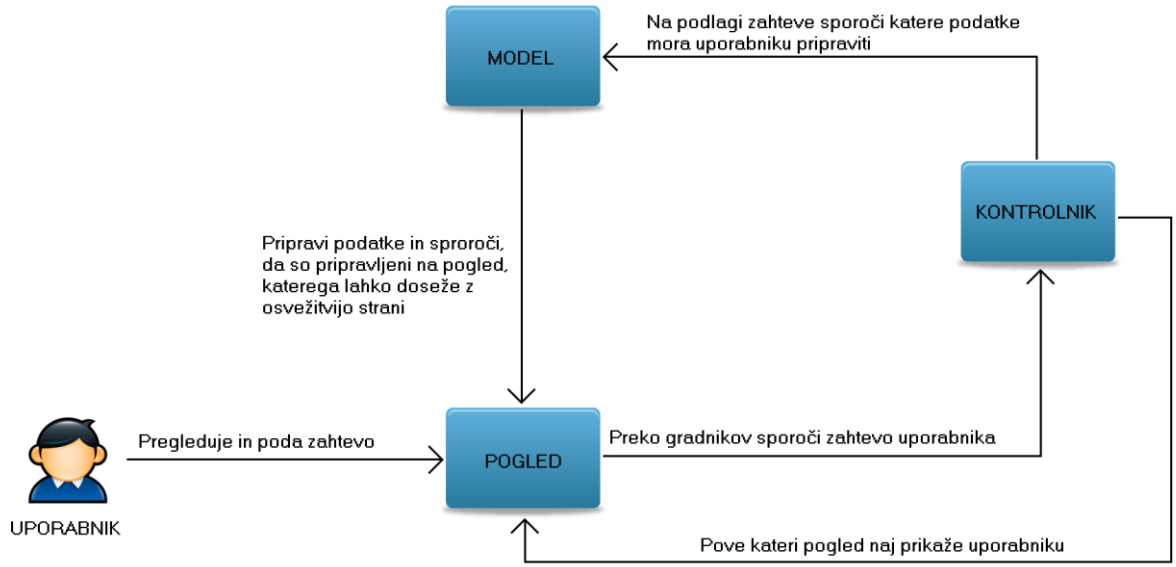
2 Model-View-Controller

Model-View-Controller ali krajše MVC je arhitekturni model, ki je sestavljen iz treh glavnih komponent:

- Model (angl. Model) – ogrodje,
- Pogled (angl. View) – oči,
- Krmilnik (angl. Controller) – možgani.

MVC je razdeljen na posamezne komponente, kjer vsaka komponenta izvaja procese in akcije, ki so prilagojene posamezni komponenti. Ločene komponente omogočajo večjo prilagodljivost pri razvoju aplikacij in pohitrijo izvajanje programske kode. MVC arhitektura omogoča tudi enostavnejše vzdrževanje kode. Koda je bolj pregledna in posamezne naloge se lahko delijo na več uporabnikov. Tako imamo lahko skupine ljudi, ki so specializirani za posamezne funkcionalnosti aplikacije. Vsaka skupina ljudi se ukvarja s svojo komponento in se tako ne meša z drugimi, kar omogoča večjo konsistentnost kode in manjše število napak. S tem dosežemo, da razvijalci posameznih komponent (modela, pogleda, krmilnika) razvijajo določene dele kode na svoj način, ne da bi bili odvisni od ostalih komponent MVC modela [2].

Na spodnji sliki (slika 2) je prikazano delovanje osnovnega MVC modela.



Slika 2: Prikaz delovanja osnovnega MVC modela

2.1 Zgodovina MVC

MVC je bil prvič predstavljen leta 1979. Idejo je predstavil Norvežan Trygve Reenskaug, ko je obiskal raziskovalni center Xerox PARC (danes samo PARC). Podjetje je imelo v lasti svoj objektno-usmerjeni programski jezik imenovan Smalltalk. MVC je bil, po nekaj letih od prvotne ideje, prvič implementiran šele v knjižnici Smalltalk-80 različice programskega jezika, pod vodstvom Jim Althoff-a. Zanimivo pa je, da ni bilo na voljo nobenih informacij o samem konceptu, niti v sami dokumentaciji Smalltalk-80 programskega jezika. Prvič je bil koncept uradno predstavljen šele leta 1988. Danes pa je koncept MVC uporabljen v večini modernejših spletnih ogrodjih in grafičnih vmesnikih (Ruby On Rails, Apple Cocoa, ASP.Net Framework, Apache Struts...) [3].

2.2 MVC model

Model je skupek:

- Podatkov,
- Logike dostopanja do teh podatkov,
- Logike za izvajanje operacij nad podatki.

V modelu so shranjeni vsi atributi, spremenljivke, funkcije in povezave med podatki (relacije). V večini primerov se objekti modela uporabljajo za pridobivanje in spreminjanje podatkov v podatkovnih skladiščih. Velikokrat pa se uporabljajo tudi za pridobivanje podatkov iz podatkovne baze ali množice podatkov (angl. dataset), ki pa se kasneje po različnih logičnih operacijah posredujejo komponenti pogled. Lahko trdimo, da je model najbolj pomembna komponenta MVC arhitekture, vendar je brez ostalih komponent nemočna.
V programiranju se uporablja tudi izraz »Thin controllers – fat models« [4], kar pomeni da je priporočeno, da se vsa logika in zahtevne operacije nahajajo v modelu, v kontrolniku pa se izvajajo samo zahteve uporabnika. Kontrolnik glede na interakcijo uporabnika, preko gradnikov na pogledu, pripravi nadaljnji postopek, kako naj se aplikacija odzove na željo uporabnika. Kontrolnik modelu sporoči, katere podatke naj pripravi in s katerimi pogoji naj te podatke omeji in jih nato prikaže uporabniku. Model pripravi podatke in jih posreduje kontrolniku, ta pa pogledu sporoči, kateri pogled naj uporabi in mu hkrati posreduje pridobljene podatke. Obenem pa lahko iz, komponente pogled, tudi dostopamo do modela. Do njega lahko dostopamo s pomočjo Razor ali pa ASPX sintakse. Na ta način lahko podatke samo beremo (ne moremo jih spreminjati). Pri uporabi omenjenih sintaks model opozori pogled, da je prišlo do sprememb podatkov in da naj osveži svoj prikaz. S tem se zagotavljata tudi konsistentnost in pravilnost podatkov, saj lahko samo kontrolnik dostopa do urejanja podatkov znotraj modela.

2.3 MVC view

Pogled skrbi za pravilno predstavitev podatkov, ki jih prejme s strani modela. Hkrati sporoča kontrolniku katero zahtevo (pritisk na gumb, vnos vrednosti, sprememba v spustnem seznamu) je uporabnik izdal in kaj pričakuje kot rezultat na podano zahtevo.

Pojavijo se tudi tako imenovani View Helpers, ki pomagajo pri prikazovanju enakih HTML predlog, katere se uporabljajo na večih pogledih, s čimer jih ni potrebno vedno znova izrisovati od začetka. S pomočjo pogleda lahko omejimo število podatkov, ki naj se prikazujejo ali pa jih glede na vrsto uporabnika tudi sploh ne dovolimo prikazati.

2.4 MVC controller

Kontrolnik skrbi za pravilno komunikacijo med modelom in pogledom. V kontrolnikih pregledujemo, če so podani podatki konsistentni (pravilnih tipov, dolžin), skrbi za avtentikacijo (prepoznavanje uporabnika) in tudi avtorizacijo uporabnikov (katere podatke lahko trenutno prijavljeni uporabnik ali pa gost vidi). Glede na zahtevo s strani pogleda se odloči, katere podatke bo model priskrbel in pogledu določi, kakšen način prikaza podatkov bo pogled za to pripravil.

Tudi tu se pojavijo tako imenovani Action Helpers, ki nam pomagajo pri procesiranju podatkov, tako da razbijajo daljšo kodo na več manjših delčkov, ki se nato posamično izvajajo, s čimer se manjša verjetnost napak pri zahtevnih, dolgih operacijah.

2.5 Orodja, ki omogočajo uporabo MVC modela

Na spletu lahko najdemo kar nekaj razvojnih orodij, ki omogočajo uporabo MVC modelske arhitekture. Med temi se najde tudi nekaj brezplačnih, ki pa so nekoliko bolj okrnjene in ne ponujajo vsega, kar bi lahko izkoristili.

Nekaj bolj poznanih orodij, ki omogočajo uporabo MVC modela:

- ASP.NET MVC,
- Joomla,
- Zend Framework,
- Django,
- Spring ...

Več o programskih orodjih za izdelavo spletnih aplikacij, pa si lahko ogledate na spletnem naslovu [5]. Tu lahko najdete spisek orodij za razvijanje spletnih aplikacij, z označenimi funkcionalnostmi, ki jih le-ta omogočajo.

2.6 Microsoft MVC

Na osnovnih principih MVC arhitekture pa je zasnovan tudi Microsoftov MVC model. Osnovni princip delovanja je ta, da uporabnik, kateremu je fizično vidna samo komponenta pogled, preko določenih gradnikov pošilja zahteve na kontrolnike. Ti se nato odločijo katere podatke bo model (glede na zahtevo uporabnika) vrnil in sporoči pogledu, kateri način prikaza naj za te podatke uporabi. Model glede na kontrolnikovo posredovanje pripravi želene podatkov, jih po potrebi spremeni in shrani, pogledu pa kontrolnik javi, da je prišlo do sprememb in ga »prosi« za osvežitev prikaza. Lahko pa model podatke še dodatno spremeni (spremeni vrstni red prikaza podatkov, določi katere podatke naj prikaže (glede na izbiro uporabnika)) in nato pripravljene podatke pošlje pogledu, ki jih prikaže na takšen način, kakor mu je to sporočil kontrolnik.

2.7 Zgodovina Microsoft MVC

Na spodnji tabeli je prikazana zgodovina izdanih verzij v časovnem obdobju [6].

Datum	Verzija
10. december 2007	ASP.NET MVC CTP
13. marec 2009	ASP.NET MVC 1.0
16. december 2009	ASP.NET MVC 2 RC
4. februar 2010	ASP.NET MVC 2 RC 2
10. marec 2010	ASP.NET MVC 2
6. oktober 2010	ASP.NET MVC 3 Beta
9. november 2010	ASP.NET MVC 3 RC
10. december 2010	ASP.NET MVC 3 RC 2
13. januar 2011	ASP.NET MVC 3
20. september 2011	ASP.NET MVC 4 Developer preview
15. februar 2012	ASP.NET MVC 4 Beta
31. maj 2012	ASP.NET MVC 4 RC
15. avgust 2012	ASP.NET MVC 4
30. maj 2013	ASP.NET MVC 4 4.0.30506.0
26. junij 2013	ASP.NET MVC 5 Preview
23. avgust 2013	ASP.NET MVC 5 RC 1
17. oktober 2013	ASP.NET MVC 5

Tabela 1: Zgodovina Microsoftovih MVC verzij

Do različice MVC 3 je za izdelavo spletnih aplikacij skrbel samo Web Forms prikazni pogon (angl. Web Forms View Engine), z različico MVC 3 pa smo dobili še Razor prikazni pogon (angl. Razor View Engine), ki se je zaradi preprostosti in odlike obdržal tudi v nasledniku različice MVC 4. Microsoft je do sedaj predstavil 4 različice MVC ogrodij, z vsako pa so prišle posodobitve za že obstoječe funkcionalnosti, dodane pa so bile tudi nove.

2.8 Glavne razlike med Microsoft MVC verzijami

Prva uradna verzija ASP.NET MVC 1 je bila širši javnosti predstavljena 13. marca 2009. Vsebovala je vse glavne značilnosti MVC- ja in še danes (v novejših verzijah) predstavljajo temelje Microsoft-ovih MVC aplikacij. Občutki ob prihodu MVC koncepta so bili sprva mešani, saj je prva različica potrebovala zelo veliko arhitekturnih sprememb v podjetjih, ki so izdelovala ogromne aplikacije, če so želela dodobra izkoristiti vse funkcionalnosti, katere je nudil nov MVC model. Na srečo pa je bil ASP.NET MVC prvi Microsoftov izdelek, ki je omogočal popolno razširitev. Večina glavnih komponent samega MVC modela je bila lahko razširjena ali pa tudi na novo implementirana s strani razvijalcev. To pa je bil tudi velik korak naprej v poti proti modernejšim in bolj učinkovitimi spletnimi aplikacijami.

Glavne značilnosti verzije MVC 1:

- Sam MVC koncept,
- Koncept usmerjanja (angl. routing) [7], ki je skrbel za ohranjanje tako imenovanih čistih URL naslovov, in bolj prijaznih do samih uporabnikov. Primer: če poznamo naslove www.test.com/produkti/Default.aspx?produktID=99&pogled=podrobneje, nam usmerjanje naslovov omogoča www.test.com/produkti/podrobneje/99, kar enormno skrči URL naslove. Ob tem pa lahko namesto identifikatorjev uporabimo tudi nize, kar je bolj pregledno samemu uporabniku www.test.com/produkti/podrobneje/mleko,
- Poenostavljeni pomočnik (angl. helpers) za izrisovanje HTML značk s pomočjo pogona,
- Pomočnik za AJAX klice, ki so skrbeli za vstavljanje pridobljenih podatkov,
- Pojavijo se začetki validinacije modela [8].

Naslednja različica - MVC 2 je sledila naslednje leto. Glavni namen različice je bil izboljšati učinkovitost delovanja storitev, s tem pa tudi lažje vzdrževanje ogromnih aplikacij v podjetjih, ki so uporabljala Microsoft-ovo MVC tehnologijo.

Glavne značilnosti verzije MVC 2:

- Preverjanje atributov modela na strežniški in odjemalski strani,
- Dodatne funkcionalnosti za večje aplikacije,
- Asinhronski kontrolniki,
- Pomočniki predlog, ki so avtomatsko izrisali prikazne strani in strani za urejanje glede na model in podane podatke,
- HTML pomočniki so bili od sedaj naprej lamdbno zasnovani (usmerjeni),
- Shranjevanje meta podatkov znotraj modela (shranjevanje sredstev v paru identifikatorja in vrednosti),
- Predloge po meri [8].

Po manj kot enem letu pa je prišla nova različica - MVC 3, z ogromnimi spremembami in dodatnimi orodji, kot so NuGet (ki omogoča enostavno dodajanje razširitev knjižnic in orodij znotraj Visual Studio ogrodja), IIS Express, SQL Server Express in drugimi. MVC 3 različica za svoje delovanje potrebuje vsaj .NET 4 Framework, zaradi kompleksnosti vseh novih pridobitev.

Glavne značilnosti verzije MVC 3:

- JavaScript validinacija in boljšo združljivost s samim scriptnim jezikom,
- Ročno validinacijo (angl. remote validination), s čimer lahko preverjamo pravilnost podatkov že na samih elementih (gradnikih) v aplikaciji,
- Dependecy resolver,
- Nov Razor prikazni pogon,
- Global action filters, ki omogočajo, da se neka funkcionalnost izvaja v celotni aplikaciji (recimo za preverjanje varnosti),
- Predloga za upravljanje z uporabniki (registracija, spremembe, brisanje, dodajanje) in prijavo v sistem [8].

Trenutno pa je najbolj priljubljena in uporabljena različica MVC 4. Z najnovešjo različico je prišla tudi podpora za brskalnike na mobilnih napravah, katera uporablja jQuery Mobile tehnologijo.

Glavne značilnosti verzije MVC 4:

- ASP.NET Web API, je novo ogrodje, ki ponuja dodatne zmožnosti za izdelavo spletnih storitev in nudi podporo za namizne brskalnike in vse mobilne platforme,
- Spremenjena predloga, ki je bolj modernih oblik, hkrati pa je izboljšana tudi funkcionalnost. Uporablja tehniko adaptivnega izrisovanja (angl. adaptive rendering), ki omogoča prikaz strani na namiznih brskalnikih in brskalnikih na mobilnikih, brez dodatnega spreminjanja nastavitev.
- Mobilne predloge in podpora za mobilne aplikacije,
- Povečana podpora za asinhrono izvajanje aplikacije,
- Bundling in Minification, ki skrbi za manjše število HTTP zahtev. Bundling omogoča enostavno združevanje večih datotek v eno samo. Omogoča ustvarjanje CSS in JavaScript skupkov (Bundlov), hkrati pa tudi poljubnih File bundlov. Minification pa skrbi za optimizacijo CSS in JavaScript datotek, tako da odstranjuje nepotrebne dele kode (komentarji, presledki) in skrajša imena spremenljivk na en znak,
- Podpora (Oauth) za prijavo v aplikacijo s strani spletnih omrežij [8].

Nova verzija MVC 5 je bila na voljo širši javnosti avgusta letos. Na žalost je v projektu nisem uspel stestirati, saj je bila aplikacija izdelana s pomočjo MVC 4 modela že izdelana.

2.9 Web Forms

ASP.Net Web Forms je del ASP.Net razvojnega ogrodja. Je eden izmed treh modelov za izdelavo spletne aplikacije. Web Forms sestavljajo posamezne spletne strani, ki jih uporabnik zahteva pri uporabi spletne aplikacije. S pomočjo interakcije lahko zahteva prikaz različnih spletnih strani, spremembe podatkov na trenutno prikazani spletni strani, ali pa preko gradnikov sproža nadaljnje akcije. Posamezne strani so izdelane s pomočjo HTML tehnologije, strežniških kontrol in strežniške kode, lahko pa se uporabljajo tudi zunanje knjižnice in skriptni jezik JavaScript.

Ko uporabnik zahteva določeno stran je ta prevedena in izvedena na strani strežnika. Šele za tem se uporabniku pošlje HTML koda, ki jo brskalnik ustrezno prikaže uporabniku. S pomočjo Visual Studio orodij, lahko spletno stran v obliki modela Web Forms izdelamo z avtomatskim grajenjem HTML elementov, ki jih v aplikacijo dodajamo s pomočjo orodjarne gradnikov. Posameznim gradnikom pa lahko določamo še lastnosti prikaza in logiko ob proženju dogodkov na posameznih gradnikih. Za razvoj strežniške logike lahko uporabljamo programska jezika Visual Basic ali pa C# [9].

2.10 Razlika med ASP.NET MVC application in ASP.NET Web Forms application

MVC aplikacija ima od različice MVC 3 na voljo 2 prikazna pogona:

- Razor View engine,
- ASPX View engine.

Medtem ko pa Web Forms aplikacije uporabljajo samo ASPX View engine. Web Forms omogoča tudi vizualni predogled, ki nam kodo prikazuje na način, kot bi le ta bila videti v samem brskalniku. MVC aplikacija pa nam takšnega predogleda ne omogoča, ampak moramo za predogled našo kodo zagnati (to pa omogoča tudi Web Forms). Vizualni predogled je grafični uporabniški vmesnik in nam tudi omogoča enostavno pozicioniranje HTML elementov s pomočjo »drag and drop« funkcionalnosti, kar pomeni, da nam avtomatsko generira kodo za izbran gradnik. Hkrati lahko tem elementom določamo tudi vse lastnosti ki jih omogoča. Medtem ko moramo v MVC aplikaciji vse elemente in njihove lastnosti določiti ročno s kodo. Enako velja tudi za povezovanje do podatkovnih skladišč. V Web Forms to lahko določimo s pomočjo grafičnega vmesnika, kjer nas program vodi do vzpostavitve povezave, hkrati pa nam tudi izdela osnovni izpis s pomočjo tabel, ki pa jih nato lahko do neke mere tudi spremenimo. Dobra stran avtomatskega dodajanja elementov je tudi to, da nam v ozadju Web Forms sami skrbijo za odpravljanje razlik med različnimi brskalniki (prikaz in delovanje naj bi bil na vseh brskalnikih enak). MVC pa nam pri tem ponuja bolj odprte roke, vendar s to razliko, da je potrebno ročno določiti vse lastnosti in tudi skrbeti za pravilen prikaz in delovanje spletne aplikacije na vseh brskalnikih.

MVC model je bolj okrnjen (angl. lightweight) kot Web Forms, vendar še vedno vsebuje vse ASP.NET funkcionalnosti.Web Forms kot privzeti pogon uporablja ASPX, zato imajo spletni dokumenti na koncu tudi končnico *.aspx. MVC pa nam pred izdelavo novega projekta ponudi izbiro med obema prikaznima pogonoma. Razor prikazovalni pogon nam nudi izbiro dokumentov med končnicami *.cshtml ali *.vbhtml. Cshtml dokumenti se generirajo ob uporabi C# programskega jezika, vbhtml pa ob uporabi Visual Basic programskega jezika.

Ne moremo pa trditi, da je eden od prikaznih pogonov boljši od drugega, saj ima vsak svoje prednosti in slabosti, medtem ko pa funkcionalnost ostaja podobna ali celo enaka. Izbira pogona je odvisna predvsem od obsega aplikacije, načina uporabe aplikacije in samega dogovora med zaposlenimi znotraj podjetja ali posamezniki, kateri prikazni način jim bolj ustreza.

2.11 Razlika med Web forms view engine in Razor view engine

Starejši od obeh je Web Forms engine (ASPX). Razor ni nov jezik, ampak nudi drugačno označevalno sintakso (angl. markup language). Razor je bolj napreden pogon predstavljen šele v MVC 3 verziji, medtem ko je ASPX prisoten že od samega začetka. Razor uporablja *.cshtml in *.vbhtml končnice dokumentov za vse poglede (angl. views), delne poglede (angl. partial views), urejevalne predloge (angl. editor templates) in grafično postavitev (angl. layout). ASPX pa uporablja končnico *.aspx za poglede, končnico *.ascx za delne poglede in urejevalne predloge in končnico *.master za grafično postavitev in glavne strani [10]. Razor sintakso se hitreje naučimo, a je samo izvajanje nekoliko počasnejše kot pri ASPX [11].

Razor pa nam omogoča tudi pridobivanje in prikazovanje, podatkov neposredno iz komponente model. Uporabniku lahko na pogledu izpišemo podatke iz modelov (objektov), ki jih MVC ogrodje ustvari samo (ob ustvarjanju projekta, ki so ključni za samo delovanje ogrodja), ali pa iz modelov, ki so bili dodani s strani razvijalca [12].

Na spodnjih slikah (slika 3, slika 4) je prikazana razlika med ASPX in Razor sintakso. Na prvi pogled sta sintaksi videti dokaj podobno. Kasneje pa ugotovimo, da Razor uporablja manj označevalnih značk kot ASPX. Z Razor pogonom označimo samo začetek funkcij in atribute, ki jih želimo prikazati na strani uporabniku, medtem ko pa moramo biti pri ASPX pogonu bolj pazljivi. Pri slednjem moramo v označevalne značke vključiti tudi vse ostale dele funkcije. Razor pogon uporablja označevalno značko @, ASPX pa za začetno označevalno značko uporablja <% in za končno še %>. Znotraj teh označevalnih značk pa zapišemo dele funkcije.

```
Web Forms pogon


<% if (DateTime.Now.Hour > 20 || DateTime.Now.Hour < 6) %>
<% { %>
    <p>Čas je za spanje !</p>
<% } else { %>
    <p>Užij dan !</p>
<% } %>




Razor


@if (DateTime.Now.Hour > 20 || DateTime.Now.Hour < 6)
{
    <p>Čas je za spanje !</p>
} else {
    <p>Užij dan !</p>
}
```

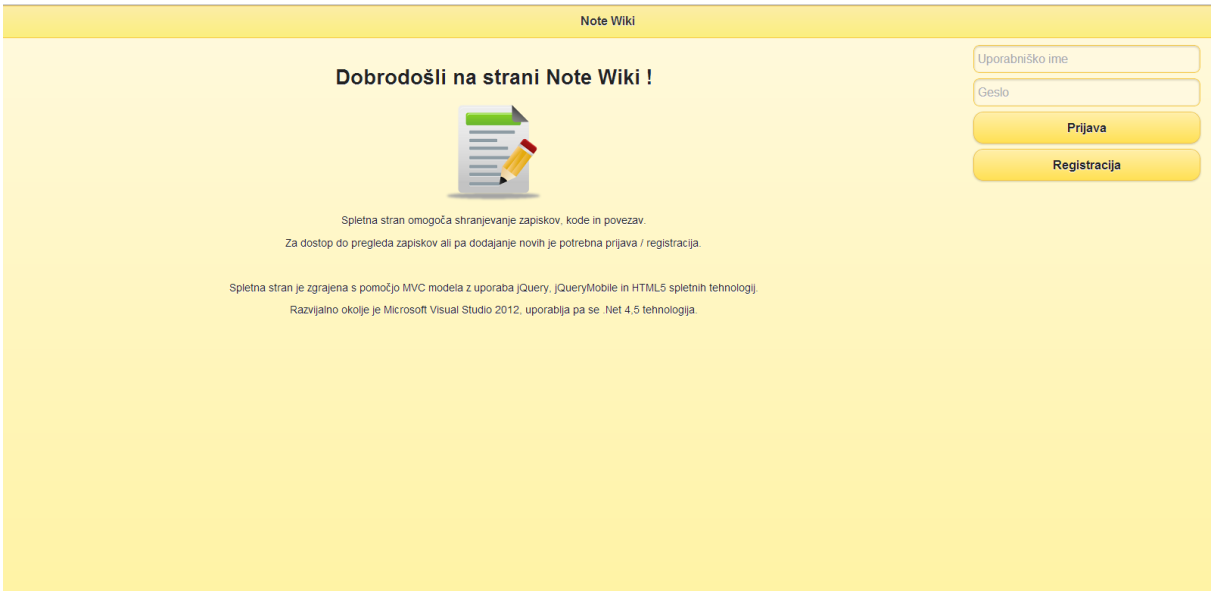
Slika 3: Primerjava Web Forms in Razor pogona na if-else stavku

```
Web Forms pogon


<% var dnevi = new List<string>() { "pon", "tor", "sre", "čet", "pet", "sob", "ned" }; %>
<ul>
    <% foreach (var dan in dnevi) %>
    <% { %>
        <li><%: dan %></li>
    <%  } %>
</ul>




Razor


@{ var dnevi = new List<string>() { "pon", "tor", "sre", "čet", "pet", "sob", "ned" }; }
<ul>
    @foreach (var dan in dnevi)
    {
        <li>@dan</li>
    }
</ul>
```

Slika 4: Primerjava Web Forms in Razor pogona na for zanki

3 Izdelava aplikacije Note Wiki

Za praktično delo diplomske naloge, smo si zadali izdelati spletno aplikacijo za shranjevanje zapiskov (angl. Notes). Aplikacijo smo izdelali na dva načina. Prvo aplikacijo smo izdelali s pomočjo MVC modela, drugo pa z Web Forms modelom. V obeh aplikacijah pa smo si pomagali še z jQuery, jQuery Mobile, jQuery UI in CSS spletnimi tehnologijami.



Slika 5: Osnovna stran spletne aplikacije Note Wiki

3.1 Namen aplikacije

Namen aplikacije je hitro in učinkovito shranjevanje različnih zapiskov (ali tudi tem), oziroma beleženje opažanj in ugotovitev. Njena uporaba je usmerjena predvsem za uporabo programerjev. Njena preprosta zgradba pa ne vzame preveč časa in ni moteča s strani končnega uporabnika, saj je dodajanje zapiskov hitro. Obenem pa je mogoča preprosta uporaba tudi z mobilnimi napravami, če se spomnimo kake ugotovitve kje na poti ali pa, če nismo trenutno pri računalniku.

Primer uporabe:

Postavimo se v vlogo programerja. Razvijamo neko aplikacijo, v kateri se določeni deli aplikacije ponavljajo v celoti ali pa samo v določeni meri (različna imena spremenljivk, dodana vrstica kode ...). Za naš primer vzemimo generiranje HTML kode znotraj JavaScripta, ki jo potem dodamo v aktivno stran. Vemo da smo to kodo že nekje uporabljali, a se trenutno ne spomnimo na katerem delu programa. Lahko bi šli preverjat vsako datoteko posebej in se premikali med vrsticami, dokler le tega dela kode ne najdemo.

V naši aplikaciji pa lahko s preprostim vnosom iskanega zapiska kodo najdemo, jo skopiramo in popravimo. S tem pridobimo na hitrosti programiranja in skrbimo za konsistenco kode, da nimamo na primer več različno napisanih funkcij za enak problem. To pa primore tudi h konsistentnosti kode in izboljšanju odpravljanja problemov kode.

3.2 Opis aplikacije

Naša aplikacija ni tako obsežna kot nekatere druge spletne aplikacije (Wikipedia, Evernote, Google Docs ...), saj je bolj usmerjena za naše potrebe. Trudili smo se izdelati aplikacijo, s katero lahko enostavno in hitro shranimo zapiske, ki nam pomagajo pri hitrejšem in kakovostnejšem programiranju. S shranjevanjem določene kode, ki se ponavlja, dosežemo konsistentnost kode, kar omogoča enostavnejše odpravljanje napak in lažjo sledljivost le-te. Trenutno omogočamo shranjevanje teksta, povezave in v računalništvu najbolj pomembnega dela – kode. Z menijem se lahko na hitro premikamo med zapiski, ki smo jih dodali sami, oziroma med zapiski, ki so jih dodali drugi uporabniki. Omogočamo pa tudi pregled zapiskov, katerih trenutni uporabnik še ni pregledal. Z iskalnikom na vsaki strani pa lahko v hitrem času dostopamo do želenega zapiska. Najden zapisek lahko nato pogledamo, vsebino zapiska pa tudi kopiramo. Seveda je za dostop do vsebine potrebna registracija in prijava. Uporabnik lahko spreminja svoje osebne podatke in določa ali naj se posamezne teme vidijo tudi drugim uporabnikom ali pa le sebi. Na voljo je tudi administrativna stran, za urejanje podatkov vseh uporabnikov in njihovih zapiskov.

3.3 Uporabljena orodja

Pri izbiri uporabljenih orodij smo se pretežno osredotočili na Microsoftova programska orodja. Kot glavno razvojno orodje smo si izbrali Microsoft Visual Studio 2012. Za shranjevanje podatkovnih skladišč smo si izbrali Microsoft SQL Server 2008 strežnik, za upravljanje le tega pa Microsoft SQL Server Managment Studio orodje.

Razlog za izbiro le-teh orodij je v tem, da Microsoftova orodja nudijo nenehne popravke in dodatke. Z dodajanjem novih dodatkov sledijo vsem novim tehnologijam, s popravki pa skrbijo za dosego čim bolj optimalnega delovanja. Hkrati razvijalcu omogočajo, z dodatnimi orodji, hitro programiranje in programiranje brez napak, kar posledično pripomore k sami konsistentnosti kode. Gre za uporabniku prijazna orodja (angl. user-friendly), katera pripomorejo k čim lažji in učinkoviti izdelavi spletnih aplikacij. Prav tako se Microsoftova orodja dobro in zanesljivo obnesejo pri vključitvi in uporabi ostalih orodij, kar pripomore k lažjemu razvijanju. Razvijalec pa se lahko v polni meri posveti ostalim ključnim funkcionalnostim aplikacije. Za uporabo Microsoftovih orodij pa je na voljo tudi obširna zbirka literature, ki omogoča razumevanje delovanja orodij, prav tako pa na spletu lahko najdemo veliko pomoči pri reševanju problemov, ki so se nam pojavili med samim razvojem.

3.3.1 Microsoft Visual Studio

Microsoft Visual Studio je integrirano razvojno okolje (angl. integrated development environment – IDE), razvito v podjetju Microsoft. Integrirano razvojno okolje je aplikacija, ki uporabniku omogoča razvijanje aplikacijskih programov [13].
Za integrirano razvojno okolje velja, da je sestavljen iz:

- Urejevalnika kode (angl. source code editor),
- Vgrajenih orodij za avtomatizacijo (angl. build automatization tools) – prevajalnik, razporeditev za globalno uporabo,
- Razhroščevalnika (angl. debugger).

Visual Studio se uporablja za razvijanje:

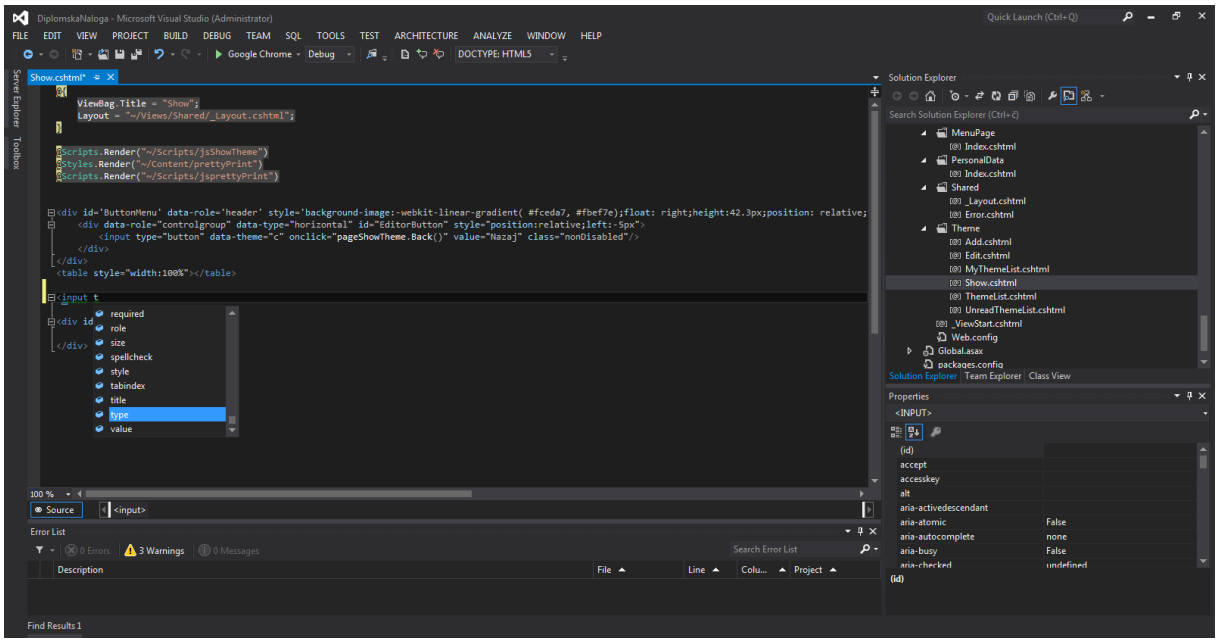
- Konzolnih aplikacij (Xbox),
- Namiznih aplikacij,
- Spletnih strani,
- Spletnih aplikacij,
- Spletnih stroritev.

Visual Studio za razvoj aplikacij uporablja .NET Framework razvojno ogrodje, ki se primarno izvaja na operacijskih sistemih Microsoft Windows [14]. .NET ogrodje vsebuje številne objektno usmerjene knjižnice, ki olajšajo samo programiranje. Upravlja z navideznimi napravami (angl.virtual machine) in pomnilnikom (angl. memory management), ter upravlja z napakami (angl. exception handling). Vsebuje tudi jezikovno združevanje (angl. language interoperability), kar pomeni, da se kodi različnih programskih jezikov združita v skupni jezikovni vmesnik, kar jima omogoča medsebojno komuniciranje [15]. Hkrati pa omogoča interakcijo med starejšimi in novejšimi verzijami programskega jezika. Omogoča pa tudi enostavno dodajanje različnih knjižnic, ki nam še dodatno olajšajo in poenostavijo razvoj aplikacij.

Razvijalci Visual Studia se trudijo, da nam bi bilo razvijanje novih aplikacij čim bolj preprosto, hitro in predvsem učinkovito. V ta namen nam v Visual Studiu služi več orodij. Na kratko bi vam predstavil šesti čut (angl. IntelliSense) in načrtovalni pregled (angl. Designer view).

IntelliSense je orodje, ki nam pomaga pri hitrosti in pravilnosti programiranja. Omogoča nam pregled seznama spremenljivk, metod, funkcij in lastnosti, ki so integrirane v knjižnicah, ali pa elemente knjižnic, ki smo jih izdelali sami. Intellisense si prav tako shranjuje in prikazuje imena uporabljenih spremenljivk JavaScripta, če le-tega uporabljamo v aplikaciji. Hkrati pa nam nudi tudi samodokončanje izbranih elementov, kar enormno pospeši izdelavo aplikacij.

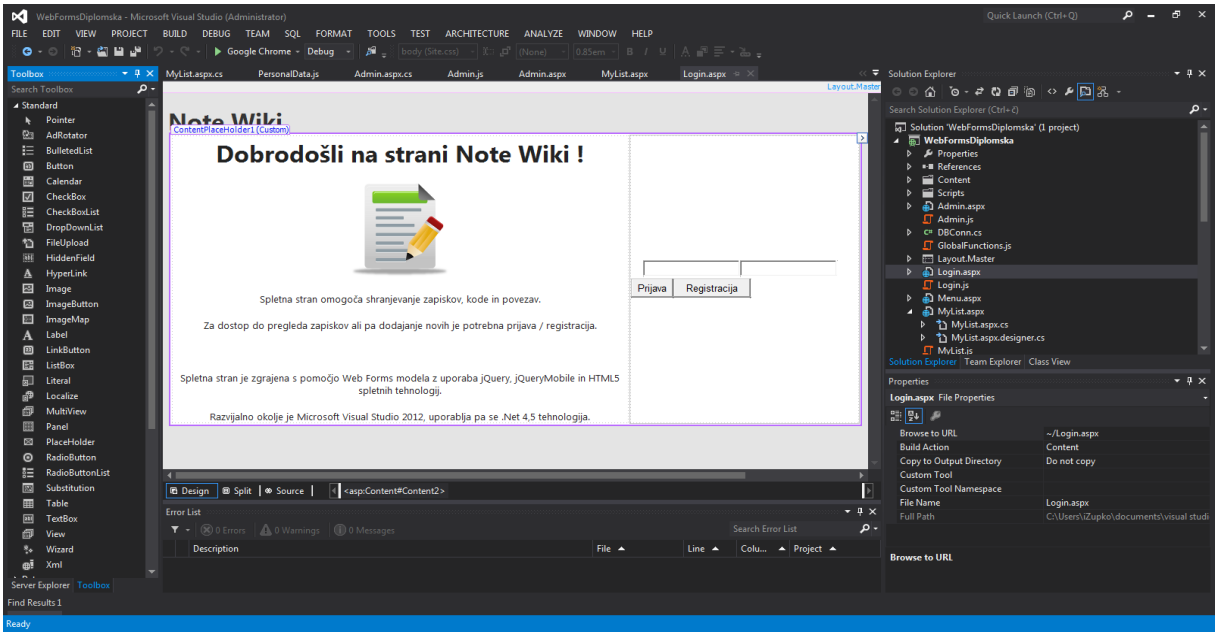
Na spodnji sliki (slika 6) je prikazan izgled razvojna orodja Microsoft Visual Studio 2012 v načinu urejanja kode s pomočjo IntelliSense orodja.



Slika 6: Prikaz pomoči IntelliSense orodja

Drugo orodje pa je tako imenovan Designer View, ki nam pomaga predvsem pri izdelavi Web Forms spletnih strani in Windows Forms namiznih aplikacij. Orodje nam omogoča enostavno postavljanje različnih kontrol na vidno okno zaslona. Prav tako omogoča upravljanje z različnimi lastnostmi posameznega elementa v delovnem okolju. Visual studio nam omogoča tudi povezovanje podatkovnih skladišč s preprosto izbiro kontrolnikov in prilagajanju lastnosti s klikanjem brez kodnega poseganja. Vsaka sprememba pa je takoj vidna v načrtovalnem pogledu, brez prevajanja in razhroščevanja aplikacije, kar nam prav tako pohitri izdelavo aplikacije.

Na spodnji sliki (slika 7) je prikazano orodje Designer View v uporabi z Web Forms modelom izdelave aplikacije.



Slika 7: Prikaz orodja Designer View pri uporabi Web Forms modela

3.3.2 Microsoft SQL Server 2008

Za shranjevanje in obdelavo podatkovnih skladišč smo si izbrali Microsoft-ov SQL Server 2008. Gre za sistem upravljanja podatkovnih skladišč, ki temeljijo na relacijskem modelu. Osnovna funkcionalnost je shranjevanje in pridobivanje podatkov glede na zahtevo aplikacije. Podatke lahko pridobivamo preko lastnega računalnika, strežnika ali pa računalnika v omrežju [16].

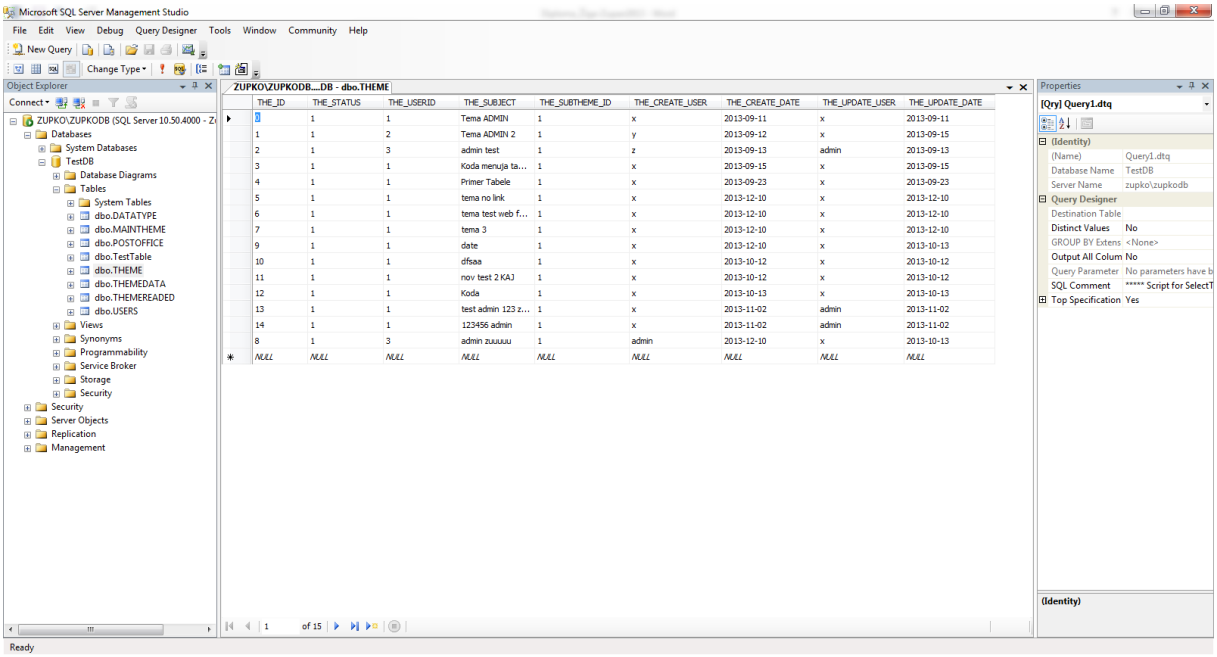
Razlog za izbiro Microsoftovega orodja za hranjenje podatkovnih skladišč je zaradi kakovostne in preproste integracije s samim Visual Studio razvojnim okoljem.

3.3.3 Microsoft SQL Server Managment Studio

Za lažje upravljanje s podatkovnimi strukturami pa bomo uporabili Microsoft SQL Server Managment Studio.

Gre za orodje, ki ga uporabljamo za bolj enostavno konfiguriranje posameznih podatkovnih struktur, upravljanje podatkovnih struktur in dodeljevanje pravic vseh komponent sistema. Omogoča način pisanja skript in grafična orodja, za izvajanje operacij nad podatkovnimi skladišči in podatki, ki so znotraj le teh. Glavno okno ogrodja je objektni raziskovalec (angl. Object Explorer), ki uporabniku omogoča raziskovanje in izbiranje posameznih podatkov znotraj podatkovnih skladišč ter samo izvajanje operacij nad njimi, obenem pa omogoča tudi nadzor nad podatki – določimo, katera podatkovna skladišča določen uporabnik lahko vidi [17].

Na spodnji sliki (slika 8) je prikazana uporaba ogrodja ob izvajanju poizvedb nad podatki v podatkovnih skladiščih.



Slika 8: Izgled in uporaba Microsoft SQL Server Managment Studio orodja

3.4 Predstavitev uporabljene tehnologije

Poleg že omenjenih orodij, za urejanje kode in upravljanje podatkovnih baz, smo v našo aplikacijo vključili še druge spletne tehnologije, ki nam olajšajo programiranje, skrbijo za boljše delovanje in poskrbijo za lepši videz aplikacije. Vse uporabljene tehnologije se v večji meri med seboj dobro razumejo, kar pomeni, da lahko z eno tehnologijo vplivamo na vse druge.

3.4.1 HTML 5

Je nov označevalni jezik za izdelavo spletnih strani. Temelji na prejšnji verziji (HTML) z mnogo dodatki. Omogoča enoten pregled strani na vseh napravah, res pa je, da vsak brskalnik nekoliko drugače generira strani. Spletno stran izdelamo s pomočjo HTML elementov, ki so sestavljeni iz značk (angl. tag). Značke po navadi pišemo v parih za začetek in konec značke. Pišemo jih s pomočjo špičastih oklepajev (<, >), lahko pa jih tudi gnezdimo. HTML elementi pa vsebujejo tudi posamezne atribute, ki skrbijo za obnašanje le-teh elementov na spletni strani. Vanje lahko vključimo tudi JavaScript funkcije in CSS lastnosti za oblikovanje prikaza gradnikov, ter izvajanje akcij ob kliku, prehodu ... Najbolj zanimiva stvar HTML 5 pa je zagotovo delovanje spletne strani tudi, ko nismo v dosegu interneta. To je mogoče s pomočjo manifesta.

3.4.2 JavaScript

JavaScript je skriptni programski jezik za interakcijo na spletnih straneh. Z JavaScriptom skrbimo za izvajanje dinamičnih spletnih strani. Podprt je v vseh novejših brskalnikih in je odprt jezik, katerega lahko uporablja vsak. Zaradi preprostosti ne potrebujemo nobenega posebnega prevajalnika in z njim lahko enostavno spreminjamo lastnosti HTML elementom in spreminjamo CSS prikaz elementov. JavaScript funkcije se izvajajo ob sprožitvi akcije. To je lahko ob kliku na gumb, ob premiku miške, ob proženju tipk, nalaganju strani, ali drugih dogodkih.

3.4.3 jQuery

jQuery je knjižnica za skriptni jezik JavaScript. Vsebuje že vnaprej spisane JavaScript funkcije, ki zmanjšajo obseg kode in enormno pospešijo samo programiranje. Za jQuery je značilen slogan »Write less, do more«, kar pomeni, da z manj pisanja naredi več.

3.4.4 jQueryUI

jQueryUI je interaktivna knjižnica, spisana s pomočjo jQuery komponente, ki skrbi za vizualizacijo strani. Knjižnica skrbi za vnaprej izdelani grafični prikaz elementov, ki pa temelji na uporabi CSS spletne tehnologije. V bistvu gre za predhodno dodeljene stilske lastnosti posameznim elementom, ki pa jih lahko po želji tudi spreminjamo.

3.4.5 jQueryMobile

jQuery Mobile je knjižnica, ki prav tako temelji na jQuery komponenti. Skrbi, da določene funkcionalnosti delujejo na vseh napravah – tako na računalnikih, kot na ostalih mobilnih napravah. S tem ni več potrebno pisati posebnih funkcij za delo na računalnikih (kjer se kot glavna interakcija uporabljajo dogodki miške) in ostalih mobilnih napravah (kjer dogodke miške nadomesti prst, oziroma dotični dogodek). S tem poskrbimo, da se naša aplikacija enako odziva na vseh (če seveda želimo) napravah enako.

3.4.6 CSS

CSS je označevalni jezik, ki določa videz spletne strani v brskalniku. Pojavil se je po zahtevi za vedno lepše in boljše spletne strani. Je ločen od kode, kar omogoča da lahko enostavno zamenjamo celoten izgled spletne strani. Omogoča vizualno spreminjanje točno enega HTML elementa s pomočjo selektorja (lojtrica - #) ali pa spremembo skupini elementov s pomočjo psevdo-razreda (pika - .).

3.5 Podatkovna baza

Naši aplikaciji uporabljata tudi podatkovno bazo. V njej se shranjujejo podatki o uporabnikih in zapiskih. Baza je sestavljena iz šestih tabel. Vsaka tabela shranjuje podatke za različne tipe (strukture). Skupno pa jim je, da vsaka od tabel vsebuje štiri skupna polja.

Skupna polja:

- Uporabniško ime uporabnika, ki je podatek ustvaril,
- Čas, ko je bil podatek ustvarjen,
- Uporabniško ime uporabnika, ki je podatek spremenil,
- Čas, ko se je podatek spremenil.

S temi štirimi polji imamo boljši vpogled na dogajanje v naši podatkovni bazi. V danem trenutku lahko izvemo, kateri uporabnik je ustvaril kateri zapis in ob kateri uri je bilo to. Prav tako lahko vidimo, v katerem času je bil podatek zadnje popravljen in kateri uporabnik je to spremenil.

Prikaz zgradbe podatkovne baze je predstavljen na sliki 9.

USERS (uporabniki): Za prijavo v sistem je potrebena prijava uporabnika s predhodno registracijo. V tej tabeli hranimo podatke o registriranih uporabnikih. Poleg osnovnih podatkov o uporabniku, uporabniškega imena in gesla, se v tabeli nahajata še dve pomembni polji. Gre za vprašanje in odgovor, s katerim uporabniku zagotovimo avtorizacijo pri urejanju podatkov. Če se odgovora ujemata, ima uporabnik na voljo urejanje predhodno shranjenih vrednosti.

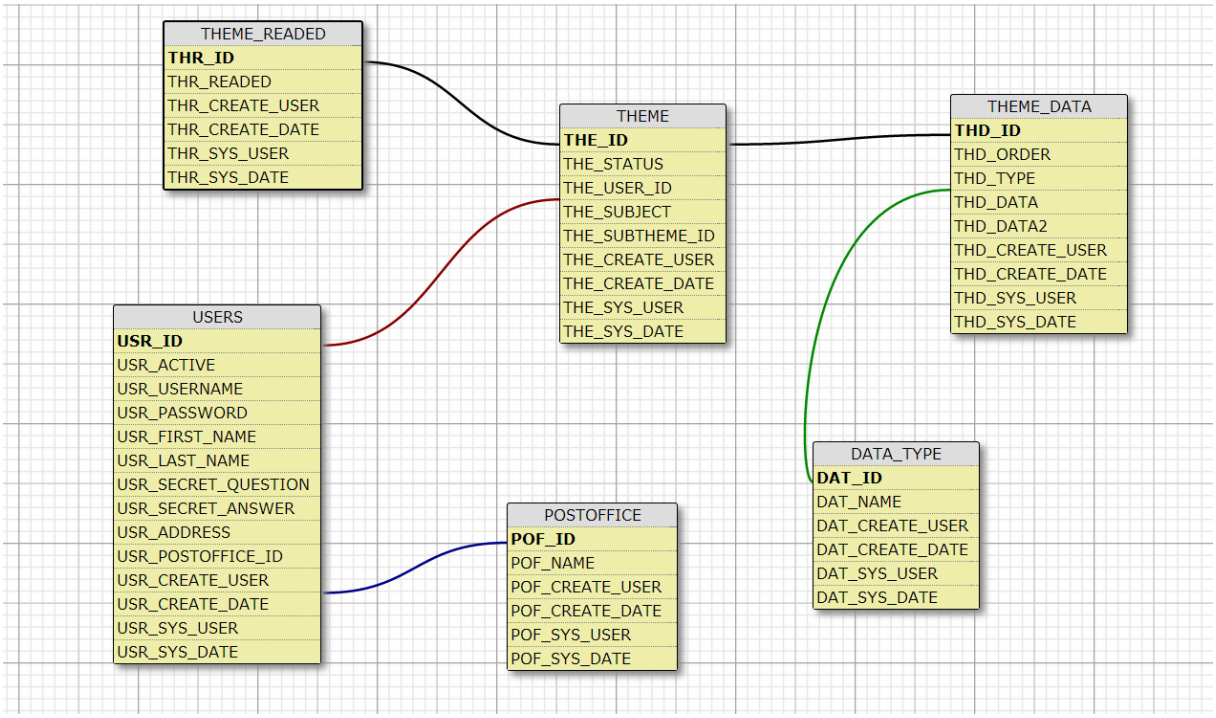
POSTOFFICE (pošte): Seznam pošt je tabela, ki združuje poštne številke in nazive pošt. Veže se na uporabnike, ki se predhodno prijavijo. Iz tabele črpamo podatke in jih ponudimo uporabniku na izbiro, s čimer zagotovimo pravilnost podatkov. S tem ne prihaja do problema, da bi recimo dva različna uporabnika za enak kraj vnesla različno poštno številko.

THEME (teme oz. zapiski): Tabela shranjuje osnovne podatke zapiska. V njen so shranjena imena zapiskov, identifikator uporabnika, ki je zapisek ustvaril, status zapiska (ali je tema aktivna in vidna vsem uporabnikom ali le uporabniku, ki je zapisek ustvaril). Tabela pa je pripravljena tudi na povezovanje določenega zapiska v nadskupino.

THEME_DATA (podatki zapiska): Tukaj pa je shranjena vsebina zapiska z določenimi tipi. Kot že rečeno trenutno omogočamo shranjevanje zapiska v tekstovni obliki, shranjevanje kode in povezave. V tabeli se pod skupnim identifikatorjem teme zapisani tipi in podatki zapisa za ta tip. Shranjuje se tudi vrstni red dodajanja posameznih tipov znotraj enega zapiska.

THEME_DATA_TYPE (vrsta elementa znotraj zapiska): Tabela združuje identifikatorje multimedijskih tipov, ki jih omogočamo in nazive le-teh tipov.

THEME_READED (prebrani zapiski): V tej tabeli beležimo, katere teme je uporabnik že pregledal. S tem omogočamo prikazovanje še neprebranih tem določenemu uporabniku. Sestavljena je iz identifikatorja teme in identifikatorjev uporabnika. Vsakič, ko dodamo novo temo se v to tabelo doda le identifikator uporabnika, ki je ta zapisek ustvaril. Vsakič, ko drug uporabnik pregleduje teme se njegov identifikator doda (če ta še ne obstaja) v tabelo, s čimer se mu neprebrana tema spremeni v prebrano. Prav tako, ko uporabnik ureja temo, se vsem ostalim uporabnikom označi spremenjen zapisek kot neprebran.



Slika 9: Prikaz podatkovne baze, uporabljene v aplikaciji Note Wiki

3.6 Vključene funkcionalnosti

Če bi želeli vključiti vse funkcionalnosti, kot jih ima recimo Wikipedia, bi se naše delo precej zavleklo. Ko pa bi končali z izdelavo recimo MVC aplikacije, bi morali zadevo ponoviti še v Web Forms aplikaciji. Zato naša aplikacija vsebuje le, za nas, ključne funkcionalosti.

Vključene funkcionalnosti:

- Registracija uporabnika,
- Prijava v aplikacijo,
- Spremembe uporabniških nastavitev,
- Vnos novega zapiska,
- Pregled zapiskov,
    - Lastni zapiski,
    - Vsi zapiski,
    - Nepregledani zapiski,
- Spremembe zapiska,
- Iskanje po zapiskih,
- Administrativna orodja,
    - Urejanje zapiskov,
    - Urejanje uporabnikov.

3.6.1 Zunanje knjižnice

Brez zunanjih knjižnic seveda ne gre, saj nam občutno pohitrijo delo in so nekatere ključne funkcije že sprogramirane, kar pomeni da se lahko posvetimo bistvu izdelave naše aplikacije.

V našem razvoju aplikacije smo poleg nekaj že omenjenih zgoraj:

- jQuery,
- jQueryUI,
- jQuery Mobile,

uporabili še knjižnico **prettifier** [18], ki skrbi za prikaz kode v brskalniku. S tem dosežemo, da se koda ne vidi samo kot navaden tekst, ampak se določeni deli kode pobarvajo, kar pripomore k boljši razumljivosti kode in hitrejši prepoznavi delovanja funkcije.

```
<table style="width:100%;border: 1px solid #fadb4e;border-radius:10px">
    <tr>
        <td style="width:150px">
            <label>Ime</label>
        </td>
        <td>
            <input type="text" placeholder="Ime" id="txtFirstnameReg"/>
        </td>
    </tr>
    <tr>
        <td style="width:150px">
            <label>Priimek</label>
        </td>
        <td>
            <input type="text" placeholder="Priimek" id="txtLastnameReg"/>
        </td>
    </tr>
    <tr>
        <td style="width:150px">
            <label>Naslov</label>
        </td>
        <td>
            <input type="text" placeholder="Naslov" id="txtAdressReg"/>
        </td>
    </tr>
    <tr>
        <td style="width:150px">
            <label>Pošta</label>
        </td>
        <td>
            <select id="selPostOfficeReg"></select>
        </td>
    </tr>
    <tr>
        <td style="width:150px">
            <label>Uporabniško ime</label>
        </td>
        <td>
            <input type="text" placeholder="Uporabniško ime" id="txtUsernameReg"/>
        </td>
    </tr>
</table>
```

Slika 10: Prikaz kode s pomočjo knjižnice prettifier v HTML pogledu

3.7 Opis delovanja

V nadaljevanju bomo predstavili delovanje spletne aplikacije. Opis delovanja temelji na osnovnem principu, ki smo ga hoteli doseči, ko smo si zadali izdelati aplikacijo za shranjevanje zapiskov. V tem delu še ne bomo predstavljali razlik med aplikacijama, izdelanama z MVC modelom, oziroma Web Forms modelom. Zagotovili smo, da se oba principa ne razlikujeta v sami funkcionalnosti in da lahko z obema pristopoma dosežemo enako delovanje aplikacije. Primerjave med obema pristopoma pa bomo razložili nekoliko kasneje. Vsem stranem je skupno, da vsebuje navigacijo za premik na prejšnjo stran oziroma preusmeritev na stran iz katere smo dostopali do trenutno aktivne strani. Enotnost videza spletnih strani smo kontrolirali s pomočjo CSS in jQueryUI predlog, s čimer smo dosegli, da si strani med seboj niso preveč različne in dajejo občutek povezanosti.

Če nekako povzamemo delovanje naše aplikacije. Pred uporabo naše aplikacije si je potrebno pridobiti uporabniški račun za vstop v spletni sistem. Po prijavi obstoječega uporabnika pa nam je na voljo delo z zapiski. Seveda lahko vsak uporabnik dodaja svoje zapiske in jih nato s pomočjo iskalnika pregleduje in jih po potrebi uporabi v svojih aplikacijah, ali pa v aplikacijah, na kateri dela več različnih programerjev. S tem lahko zagotavljamo enakost kode skozi celotno aplikacijo in hitrejše odpravljanje napak na delih kode, katere mogoče nismo izdelali sami. Pregled zapiskov je ločen na tri različne dele.

Omogočamo pregled svojih lastnih zapiskov, pregled vseh aktivnih zapiskov in pregled še neprebranih zapiskov s strani trenutnega uporabnika. Dele zapiska lahko aktivni uporabnik tudi kopira in nato uporabi po svoji želji. Pri dodani povezavi v zapisku se z enostavnim klikom na dodeljeno povezavo, odpre stran shranjeno v zapisku, bodisi za potrebe dodatnih informacij. Urejanje zapiskov je mogoče samo na zapiskih uporabnika, ki jih je ustvaril. Do urejanja lahko dostopamo s pregledom dodanih lastnih zapiskov. Uporabniku je dovoljeno tudi urejanje njegovih osebnih podatkov, ki jih je izpolnil pri registraciji, ampak le pod pogojem, da se njegov vneseni odgovor ujema z odgovorom v podatkovni bazi.

Dostop do vseh podatkov o uporabnikih in tudi vseh zapiskih v bazi pa ima administrator. Omogočen mu je pregled in tudi urejanje vseh podatkov znotraj podatkovne baze. Na voljo mu je tudi prikaz dodanih tem po posameznih uporabnikih.

V nadaljevanju so na kratko razloženi posamezni deli aplikacije, podrobneje pa so funkcionalnosti, s slikami izdelanih strani, razložene v prilogah, pri čemer pa so slike, uporabljene za prikaz, vzete iz MVC načina delovanja aplikacije.

3.7.1 Registracija

Registracija uporabnika služi za dodajanje novih uporabniških računov, s katerimi lahko dostopajo do naše aplikacije. Poleg osnovnih podatkov pri registraciji je potrebno vnesti še skrivno vprašanje in skrivni odgovor, kar se uporablja za avtorizacijo uporabnika pri spreminjanju osebnih podatkov.

3.7.2 Prijava

Prijava je ključna funkcionalnost, ki zagotavljanje možnost dela na naši aplikaciji. Za prijavo v našo aplikacijo se uporablja uporabniško ime in geslo, določeno ob registraciji.

3.7.3 Meni

Menijska stran je osnovna stran naše aplikacije. S te strani lahko izbiramo med funkcionalnostmi, ki jih naša spletna aplikacija omogoča. Z menijske strani lahko dostopamo do strani za spremembo uporabniških nastavitev, do pregleda zapiskov ter dodajanja novih zapiskov ali popravljanja naših obstoječih zapiskov.

3.7.4 Spremembe uporabniških podatkov

Na tej strani uporabniku omogočamo urejanje lastnih podatkov, ki jih je izpolnil ob registraciji. Avtorizacijo uporabnika preverjamo s pomočjo kombinacije skrivnega vprašanja in skrivnega gesla. Če se vnesena podatka ujemata, ima uporabnik na voljo spreminjanje prav vseh lastnih uporabniških podatkov.

3.7.5 Pregled zapiskov

Uporabniku omogočamo pregled treh različnih zapiskov:

- Pregled vseh zapiskov,
- Pregled lastnih zapiskov,
- Pregled še ne prebranih zapiskov.

S pomočjo teh strani lahko uporabnik pregleduje zapiske, jih ureja in tudi kopira lastne ugotovitve ter ugotovitve ostalih uporabnikov. Za lažje iskanje po zapiskih je uporabniku na voljo iskalnik tem, kateri mu dinamično prikazuje zapiske na podlagi iskane besede.

3.7.5.1 Pregled vseh zapiskov

Prikazuje vse zapiske v bazi, ki jih je možno videti. Vidni so zapisi vseh uporabnikov, če so se le te odločili za prikaz ostalim uporabnikom.

3.7.5.2 Pregled lastnih zapiskov

Pregled lastnih zapiskov omogoča pregled zapiskov, dodanih s strani trenutno prijavljenega uporabnika. V tem načinu trenutni uporabnik ne vidi zapiskov ostalih uporabnikov. S to stranjo je uporabniku dodana tudi možnost urejanja zapiska, saj lahko zapiske spreminja samo avtor le teh.

3.7.5.3 Pregled nepregledanih zapiskov

S pregledom nepregledanih zapiskov lahko uporabnik vidi samo zapiske, ki jih še ni pregledal. To smo dosegli s pomočjo tabele v podatkovni bazi, ki hrani katere teme je kateri uporabnik že pregledal. Ob pogledu teme se le ta izbriše iz seznama neprebranih zapiskov.

3.7.6 Prikaz in urejanje zapiska

Na teh straneh lahko pregledujemo ali pa urejamo dodan zapisek. Obe izmed funkcionalnosti se nekoliko razlikujeta v načinu delovanja.

Pri pregledu zapiskov nam je v veliko pomoč zunanja knjižnica prettifer, ki olepša prikaz dodane kode. Poleg kode pa lahko uporabnik pregleduje še dodan opis in se s klikom na povezavo premakne na spletni naslov, ki je zapisan v zapisku. S tem uporabniku omogočimo še kakšne dodatne informacije, kaj naj bi izbrani zapisek predstavljal.

Pri urejanju zapiska pa uporabnik lahko izbira med prikazovanjem opisa, kode in povezave. Urejanje zapiska je na voljo le avtorju zapiska in administratorju. S shranitvijo urejenega zapiska, se zapisek, tudi če je bil prej že prebran, ostalim uporabnikom doda v spisek še neprebranih zapiskov.

3.7.7 Dodajanje zapiska

Pri dodajanju zapiska lahko prav tako, kot pri urejanju zapiska, izbiramo med prikazovanjem opisa, kode in povezave na zapisku. Po shranitvi se avtorju avtomatsko doda status prebranega, ostalim uporabnikom pa se bo novo dodani izpisek prikazal v seznamu še ne prebranih zapiskov.

3.7.8 Administrativna stran

Skrbnikom strani pa sta na voljo dva osnovna pregleda.
To sta:

- Pregled uporabnikov,
- Pregled tem.

Pri pregledu uporabnikov ima skrbnik na voljo prikaz podatkov registriranega uporabnika, popravljanje podatkov registriranega uporabnika in pregled zapiskov posameznega uporabnika.

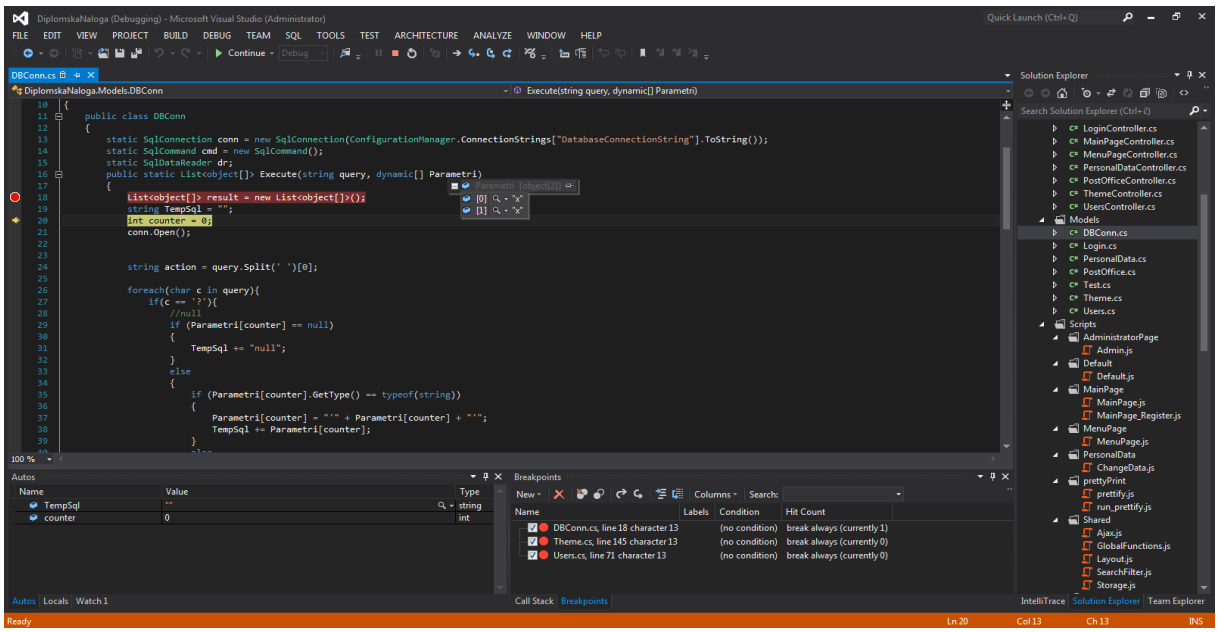
Pri pregledu zapiskov pa je skrbniku na voljo pogled teme in urejanje teme. Skrbnik strani ima dostop do vseh zapiskov, ne glede na to ali je avtor določil, da je ta zapisek zaseben ali javen. Če skrbnik spletne strani spremeni zapisek, se ta prikaže v seznamu še ne prebranih zapisov vsem uporabnikom, tudi avtorju, ki je ta zapisek ustvaril. Ob tem pa ne spremeni avtorja zapisa, kar pomeni, da ga avtor še vedno lahko spreminja.

3.8 Način odpravljanja napak

Pri razvoju aplikacij ne moremo mimo dejstva, da ne bomo ob programiranju kdaj pa kdaj naredili kake napake. Pomembno ob ugotovitvi napake je, da jo čim prej ugotovimo in odpravimo.

V Visual Studio je na voljo razhroščevalnik napak (angl. debugger), s katerim smo odpravljali napake in preverjali stanja trenutnih objektov, spremenljivk, metod ter funkcij.

Na spodnji sliki (slika 11) je prikazano odpravljanje napak s pomočjo razhroščevalnika v aplikaciji, izdelani s pomočjo MVC modela.

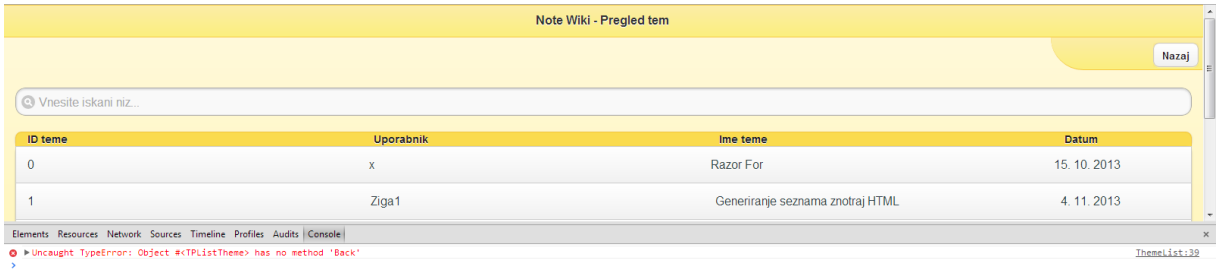


Slika 11: Odpravljanje napak s pomočjo vgrajenega razhroščevalnika

Za ugotavljanje in razreševanje napak na uporabniški strani pa smo morali uporabiti drugačen pristop. Tu nam je bil v veliko pomoč razhroščevalnik v brskalniku (za razvoj in testiranje smo uporabljali Google Chrome brskalnik). Do njega lahko znotraj brskalnika dostopamo s pomočjo kombinacije tipk **CTRL + SHIFT + I**.

Tu imamo na voljo pregled HTML elementov, dostop do virov strani (Local Storage, Session Storage, Cookies), ter dostop do JavaScript datotek, kjer lahko na poljubnem delu kode izvajanje JavaScripta ustavimo in se postopoma premikamo po vrsticah kode. V primeru napak se nam v konzolnem zavihku (angl. console), prikaže vrsta napak in vrstica kode, kjer se je ta napaka pripetila.

Primer uporabe razhroščevalnika znotraj brskalnika lahko vidite na spodnji sliki (slika 12).



Slika 12: Prikaz odpravljanja napak z uporabo Google Chrome razhroščevalnika

4 Razlike med razvojem z obema pristopoma

Razumljivo in pričakovano je, da so se med razvojem obeh načinov spletnih aplikacij porajale razlike, ki smo jih bili primorani tudi odpraviti in vsako izmed spletnih aplikacij prilagoditi na zastavljeno delovanje le-te. Razlike se pojavljajo zaradi drugačne zasnove delovanja obeh načinov izdelave spletne aplikacije. Ker smo najprej razvili spletno aplikacijo s pomočjo MVC modela, smo bili zatem primorani oblikovati Web Forms delovanje aplikacije do takšne mere, da funkcionalnosti ne bi bile preveč različne.

Razlike so se pojavile med našim izdelovanjem aplikacije Note Wiki. Nastale so na podlagi našega razvoja aplikacije, kar ne pomeni da se podobne razlike porajajo tudi pri drugih razvijalcih. Pred izdelavo aplikacije smo imeli nekoliko več znanja pri razvijanju s pomočjo MVC modela, zato je tudi možno, da se nekaterih stvari pri Web Formsih nismo lotili najbolj učinkovito. Verjetno se pri nekaterih rešitvah pozna tudi pomanjkanje izkušenj, s čimer smo bili primorani napako odpraviti z znanjem, ki smo ga v tistem trenutku imeli. Prav tako smo se odločili, da pri uporabi Web Forms ne bomo uporabljali AJAX tehnologije, ampak običajen »post back«, kar pomeni, da se vsaka sprememba gradnika s strani uporabnika pošlje na strežnik, ta pa glede na zahtevo odgovori in spremeni prikaz uporabniku.

V nadaljevanju so prestavljene nekatere razlike, s katerimi smo se skozi razvoj aplikacij soočili in jih tudi uspešno odpravili.

	Naziv razlike	**Opis**
1	Session proti sessionStorage	Shranjevanje podatkov v sejo
2	Samodejni obrazci	Avtomatski obrazec
3	HTML podpora	Različna načina HTML elementov
4	ASP tipka proti klasični	Kako se razlikujeta HTML gradnika za tipko
5	Integracija JavaScript z ASP elementi	Klicanje ASP elementov z JavaScript
6	Popup	Realizacija popup-a pri Web Forms
7	Kopiranje CSS sloga	Razlika med oblikovnimi lastnostmi
8	Problem z iskalnikom	Problem pri implementaciji enake funkcionalnosti iskanja
9	IntelliSense pomoč pri slogih	Slabša podpora pri ASP elementih
10	Forms in submit	Razlika v izvajanju s »post back«
11	Povezava zunanjih knjižnic	Razlika vključevanja zunanjih knjižnic
12	Pridobivanje podatkov	Kako smo drugače pridobivali podatke v Web Formsih
13	Testiranje strani	Razlika pri testiranju razvoja aplikacij
14	Skupna funkcija za prikaz zapiskov	Razlika med skupnimi funkcijami
15	Izmenjava podatkov med JavaScript in strežnikom	Kako smo realizirali izmenjavo podatkov med JS in strežnikom
16	Navigacija med stranmi	Kaj smo uporabljali pri navigaciji med stranmi aplikacije

Tabela 2: Spisek in kratek opis razlik, ki so se pojavile med razvojem obeh aplikacij

4.1 Session proti sessionStorage

Pri razvoju naše aplikacije, smo se nekajkrat zatekli tudi po shranjevanju določenih uporabniških nastavitev v tako imenovane seje. (angl. Session). Seje omogočajo shranjevanje preprostih spremenljivk, ali pa celo celih tabel podatkov. S pomočjo sej lahko različne podatke prenašamo med različnimi stranmi znotraj naše aplikacije in poskrbimo, da se glede na dane podatke v seji stran drugače generira, oziroma pokaže druge podatke kot ostalim uporabnikom.

V MVC modelu smo uporabljali samo sejno shrambo podatkov (angl. Session Storage), saj se je celotna stran generirala pri samem uporabniku. Podatke smo s pomočjo AJAX klicev pridobili in shranjevali s klici na bazo, prikaz in delo s samimi podatki pa se generira pri samem uporabniku, s čimer sprostimo delovanje strežnika. Sejna shramba podatkov se nahaja v novejših brskalnikih (chrome, opera, safari). Brskalnik za vsako spletno aplikacijo vsebuje svojo sejno shrambo. Podatki v sejni shrambi z drugih spletnih strani so nevidne in ni možno dostopanje do njih. Ta se generira glede na naslov strani strežnika, s čimer je omogočeno katere seje naj se vidijo in katere ne. Do sejne shrambe lahko enostavno dostopamo s pomočjo JavaScripta.

V Web Formsih pa smo uporabili kombinacijo sejne hrambe in shranjevanje seje na strani strežnika. Tu smo imeli nekoliko težav pri prenosu seje iz same aplikacije v vsako stran, saj smo za nekatere JavaScript funkcije potrebovali podatke iz seje, ki pa jih v lokalni sejni shrambi ni. Ker pa Web Forms deluje tako, da za vsako spremembo oziroma potrditev kliče strežnik, je dostop direktno iz JavaScripta skoraj nemogoč. S tem pristopom smo poskrbeli da so se strani generirale pravilno.

Session in sessionStorage sta dva različna načina hrambe sej. Z JavaScript ne moremo neposredno dostopati do Session shrambe in obratno iz strežnika ne moremo neposredno dostopati do sessionStorage sejne shrambe. S temi težavi se srečujejo vsi razvijalci in zato raje uporabljajo piškotke (angl. cookies).

4.2 Samodejni obrazci

Pri izdelavi Web Forms aplikacije imamo na možnost nekaj že vnaprej pripravljenih obrazcev. Tako nam je na voljo samodejni obrazec za prijavo v sistem, zamenjavo gesla, pridobivanje pozabljenega gesla in registracijo uporabnika. Tega pa lahko dodamo s pomočjo orodjarne pri izdelavi Web Forms aplikacije. Tega nam MVC model ne omogoča in je potrebno vse izdelati ročno. Pri Web Formsih pa je potrebno samo povezati avtomatsko generiran obrazec s podatkovno bazo, in določiti možnosti za prikaz. Če smo zadovoljni z načinom prijave, nam avtomatska dodaja prihrani kar nekaj časa. A ker smo želeli, da sta aplikaciji videti v večji meri enako, smo morali tudi v Web Formsih izdelati lasten sistem prijave, določiti stilske predloge in spisati algoritem za prijavo uporabnika v aplikacijo.

Lahko pa bi uporabili že vnaprej pripravljene osnovno izdelane strani. Problem je, da se začetni predlogi v obeh načinih razlikujeta in bi tako morali popravljati obe predlogi hkrati, da bi dosegli podoben videz. Problem pri predlogah pa je tudi v tem, da lahko dobimo kar nekaj datotek, ki jih sploh ne potrebuje, kar samo poveča in upočasni delovanje aplikacije. Zato smo se raje odločili za izdelavo vsake spletne strani od začetka do konca, s čimer smo imeli večji nadzor in vključili samo datoteke, ki smo jih potrebovali.

4.3 HTML podpora

Velika razlika med obema pristopoma je tudi v načinu izdelava HTML elementov (gradnikov). Web Forms uporablja aspx različico HTML strani. V bistvu gre za spremenjeno tehnologijo HTML, z nekaj dodanimi funkcionalnostmi. Na primer vsak gradnik ima dodano na začetku še značko asp. Seveda uporaba tega ni nujno potrebna, a če želimo dodajati nove elemente v našo spletno stran s pomočjo orodjarne, se vsem izbranim elementom avtomatsko dodajo asp značke. Pri Web Forms se tudi cela stran nahaja znotraj form HTML značk. S tem se ob vsaki potrditvi (angl. submit) forma shrani in pošlje v obdelavo na strežnik.

MVC model pa uporablja klasične HTML elemente, kjer lahko enostavneje kontroliramo ali naj se ob neki spremembi aplikacija obrne po pomoč k strežniku ali ne.

4.4 ASP tipka proti klasični

S pomočjo orodjarne v Web Formsih se nam avtomatsko dodajo ASP tipke. Tem tipkam lahko določimo dva različna dogodka ob proženju. Na voljo nam je OnClick dogodek, ki sproži zahtevo na strežniku, in OnClientClick, ki izvede operacijo pri uporabniku, za tem pa se sproži še zahteva na strežnik. Pri avtomatskem dodajanju se mu doda tudi poseben atribut runat="server", kar pomeni, da tipka kliče strežnik za nadaljnje akcije. Če želimo, da se forma po operaciji pri uporabniku ne potrdi in se »post back« ne izvede, je potrebno s pomočjo JavaScript funkcije vračanje preklica s pomočjo kode return false.

Pri MVC pa je zadeva enostavnejša, saj nam je na voljo samo onclick atribut, ki proži naslednjo JavaScript funkcijo. V tej JavaScript funkciji pa lahko nato kličemo kontrolnik in akcijo znotraj tega. S pomočjo akcije povemo strežniku, s pomočjo katerega modela, naj nam pripravi podatke in jih nato vrne.

4.5 Integracija JavaScript z ASP elementi

Če želimo v Web Formsih kakšen ASP gumb uporabljati v JavaScript funkciji, je dostop do njega nekoliko otežen. Predhodno moramo vedeti ime prostora, ki vsebuje HTML elemente (angl. ContentPlaceHolder). Vsebovalniki prostora se pojavijo, ko želimo imeti spletno aplikacijo zgrajeno na neki osnovni maski (Master page). Ta stran je deljenja z ostalimi stranmi, kjer se odločimo za uporabo osnovne maske. Na njej pa imamo lahko določen videz strani in povezave do zunanjih knjižnic, ki jih uporabljamo. Recimo, da imamo polje za vnos besedila. Njegov identifikator je vnosnoPolje. V navadnem HTML formatu lahko do njegove vrednosti dostopamo s pomočjo jQuery funkcije `$("#vnosnoPolje").val()`. Pri ASP elementih pa je, za dostop, potrebno vpisati še ime vsebovalnika prostora. Tako moramo funkcijo popraviti na `$("# ContentPlaceHolder1_vnosPolje").val()`. Problem je, da imamo včasih več delitev vsebovalnika prostora, tako moramo v danem času vedno vedeti, do katerega polja, v katerem vsebovalniku dostopamo in ustrezno dopišemo ime le-tega. Tudi odpravljanje težav je nekoliko problematično, saj se v HTML-ju element vidi samo kot vnosPolje brez imena vsebovalnika, kar pripelje do zmede.

4.6 Popup

jQuery UI omogoča prikaz tako imenovanih pojavnih oken. Popup smo uporabili pri avtorizaciji uporabnika za spremembo lastnih podatkov. S prikazom pojavnega okna smo uporabniku ponudili prikaz podatkov, ki jih mora izpolniti, da se mu možnosti urejanja prikažejo.

Z MVC modelom smo enostavno klicali pojavno okno, ki je uporabniku ponudil vpis podatkov.

Pri Web Forms pa smo naleteli na problem neprikazovanja pojavnega okna. Ta funkcionalnost znotraj ASP strani ni mogoča. To smo rešili s pomočjo ASP plošče (angl. panel). Na ploščo smo postavili potrebne elemente za avtentikacijo uporabnika in ga ob zagonu strani skrili. Ko smo ga potrebovali, smo ploščo z ASP elementi prikazali in po zaključku skrili, ter uporabniku omogočili urejanje podatkov. Seveda smo plošči bili primorani spremeniti videz, tako da je bil čim bolj podoben pojavnemu oknu v MVC načinu.

4.7 Kopiranje CSS sloga

Prva aplikacija, ki smo izdelali, je bila zgrajena s pomočjo MVC modela. Ker smo že vložili kar nekaj truda v videz same strani, smo si želeli prihraniti nekoliko časa pri oblikovanju strani v načinu izdelave z Web Forms. Pri kopiranju smo ugotovili, da se vsi HTML elementi v obeh načinih ne obnašajo enako. Tako so bili elementi v Web Forms nekoliko prestavljeni v primerjavi z MVC modelom. Sledilo je ročno popravljanje vseh elementov s pomočjo CSS sloga znotraj HTML elementov. Obenem pa smo morali popraviti tudi nekaj elementov in klasične HTML oblike v obliko za ASP strani.

4.8 Problem z iskalnikom

Za namene aplikacije smo razvili funkcionalnost za iskanje po zapiskih. Funkcionalnost smo razvili s pomočjo JavaScript in jQuery funkcij. Ker smo uporabniku želeli ob ponovni vrnitvi na stran prikazati zadnjo vrednost iskanja, smo zato na vsako stran vezali dogodek, kaj naj se zgodi pred prikazom strani (angl. pagebeforeshow). Za pridobitev podatka zadnje vrednosti pa smo uporabljali sejno shrambo. Ker je bila funkcionalnost algoritma že rešena smo jo hoteli implementirati v oba načina spletne aplikacije. S tem tudi lažje popravljamo napako, ki se ponovi. Tako lahko popravimo napako v eni aplikaciji in jo prekopiramo v drugo. Če pa bi imeli dve različni funkcionalnosti, bi morali algoritem popravljati dvakrat – vsakič na drugačen način.

Pri gradnji z MVC načinom nismo imeli težav, saj uporabljamo dogodek »pagebeforeshow« na vsaki strani. Medtem, ko pa pri spletni aplikaciji grajeni s pomočjo Web Forms tega dogodka nismo imeli nikjer, saj aplikacija deluje na drugačen način. Zato smo morali na vsaki Web Forms strani, kjer iskalnik uporabljamo, dodati dogodek, ki je iz zadnje shranjenega vnosa generiral seznam, ki ustreza vnesenemu nizu, ter ga prikazati v našem iskalniku.

Vsaki strani z iskalnikom smo dodali dogodek ob naložitvi strani:

```
$(window).bind("pagebeforeshow", function () {});
```

4.9 IntelliSense pomoč pri slogih

Veliko različnih slogov smo dodajali tudi posameznim HTML elementom. Z njimi smo spreminjali postavitve elementov, videz, velikost, prosojnost ... Pri uporabi klasičnih HTML elementov s tem nismo imeli težav. Tudi Visual Studio nam s posebnim orodjem IntelliSense svetuje, katere lastnosti bomo uporabili ter kateri atributi so znotraj lastnosti mogoče. To generira na podlagi vnesenega niza s strani uporabnika. Pri dodajanju HTML atributa »style"«, medtem pa na ASP elementih pomoči s strani IntelliSensa nismo bili deležni. To nam je nekoliko podaljšalo čas oblikovanja elementa, težje pa je bilo tudi ugotavljanje napak, saj se je večkrat zgodilo, da smo kakšen atribut napačno zapisali ali pa pozabili kakšne ključne dele kode.

4.10 Forms in submit

Kar nekaj problemov nam je povzročala pravilna uporaba Web Forms načina s »post back« načinom, saj smo prej izdelali aplikacijo z MVC modelom, ki deluje drugače. Pri Web Formsih se celotna stran nahaja znotraj obrazca, ki skrbi, da se vsaka sprememba ASP elementov takoj potrdi in pošlje zahtevo strežniku, ki nato glede na akcijo odgovori uporabniku in mu pripravi prikaz zahteve. To pa je včasih moteče, saj recimo izbira možnosti iz seznama še ne pomeni, da želimo vrnitev akcije s strani strežnika. To smo odpravili s tem, da smo ob dogodkih, katerih ne želimo akcij s strani strežnika, v JavaScript funkciji vračali naj se zahteva na strežnik ne zgodi.

4.11 Povezava zunanjih knjižnic

Kot že rečeno smo v naši aplikaciji uporabili tudi nekaj zunanjih knjižnic. Tudi tu je bila potrebna pazljivost, ki mi je bila prej neznana in nam je povzročala kar nekaj težav. Težave smo imeli s povezavo zunanjih knjižnic za jQuery, jQuery Mobile in jQuery UI. Te so se pojavile samo pri gradnji aplikacije z uporabo Web Forms modela. Ko si prenesemo zunanje knjižnice s spleta, so te po navadi zapakirane skupaj s končnicami *.min in *.js. V datotekah s končnico *.min gre za skrčeno verzijo datoteke *.js. Skrčene verzije se uporabljajo z namenom zmanjšanja velikosti datotek in posledično hitrejše delovanje spletne strani. V Web Formsih smo povezavo zunanjih knjižnic lahko dosegli le tako, da smo v projekt vključili datoteke s končnico *.min, medtem ko pri MVC modelu ni pomembno katero od datotek uporabljamo.

4.12 Pridobivanje podatkov

Pri pridobivanju podatkov in generiranju prikaza smo prav tako naleteli na dva različna načina razvoja. Vsak od pristopov izdelave spletne aplikacije ima drugačen način pridobivanja podatkov. Skupno pa smo uporabili JavaScript funkcionalnost za grajenje HTML elementov s podatki, ki smo jih pridobili iz podatkovne baze.

Pri MVC modelu smo s pomočjo AJAX klicev na kontrolnik in njegovo akcijo prisilili strežnik, naj nam vrne podatke iz modela, ki skrbi za pridobivanje podatkov iz podatkovne baze. Po uspešno izvedenem AJAX klicu smo nato dobljene podatke vključili v gradnjo HTML strani in jih nato pripeli na stran, s katere je uporabnik zahtevo klical.

Pri Web Forms načinu pa smo strežniku poslali zahtevo po želenih podatkih. Ta jih je pridobil iz podatkovne baze in vrnil uporabniku. Po uspešno vrnjeni zahtevi strežnika smo nato podatke s pomočjo spodnje funkcije poslali v izvajanje JavaScript funkcije.

```
ClientScript.RegisterStartupScript(Page.GetType(), "command1", "<script
language='javascript'>startMyThemeList(" + serializer.Serialize(sqlResult) +
")</script>");
```

Podatke smo pred pošiljanjem v JavaScript funkcijo ustrezno prilagodili za prenos s pomočjo serializacije podatkov.

Zakaj smo se potem pravzaprav odločili za uporabo JavaScript funkcij?

Za to smo se odločili z namenom, da bi si prikrajšali čas z razvojem že obstoječih funkcionalnosti na nov drugačen način. S tem so se pojavili različni načini predvsem pri Web Forms med pošiljanjem podatkov med JavaScriptom in C#. Zagotoviti smo morali pravilno pošiljanje podatkov z uporabnikove na strežniško stran in obratno – iz strežniške k uporabniku, da bi se strani generirale pri njem samem.

S tem smo zagotovili enakost funkcije v obeh načinih izdelava aplikacije, kar prihrani čas pri morebitnem spreminjanju funkcionalnosti.

4.13 Testiranje strani

Pri razvojnem testiranju projekta so prav tako zelo opazne razlike. MVC model ob zagonu aplikacije vedno začne s strani, ki je podana preko kontrolnika in akcije, ne glede s katere aktivne strani smo zagnali aplikacijo. Medtem ko pa pri Web Formsih se prikaže stran s katere smo pognali pogled. To je recimo koristno pri testiranju HTML videza strani, ko v ozadju še ni aktivnih funkcij. Problem pa se pojavi, ko neka funkcija potrebuje podatke, ki pa jih dobi iz predhodne strani. Ob takem dogodku aplikacija javi napako, za kar je potrebno ponovno zagnati aplikacijo z ustrezne strani. To smo rešili tako, da smo v naših nastavitvah Web Forms aplikacije določili začetno stran, s katere naj aplikacija vedno začne.

4.14 Skupna funkcija za prikaz zapiskov

Ker naša aplikacija vsebuje tri zelo podobne načine prikazovanja seznama zapiskov, se določeni algoritmi v ozadju ponavljajo. V mislih imamo predvsem preverjanje statusa prebranosti zapiska, s čimer določamo ali je uporabnik neki zapisek že prebral in če ga je nato še potrebno prikazovati v seznamu še neprebranih zapiskov za določenega uporabnika. Hoteli smo da se ob vsakem izbranem zapisku pri pregledu preveri in po potrebi doda status branja.

To nam je z MVC modelom uspešno uspelo, saj smo skupni algoritem dodali v deljeno JavaScript datoteko, ter na vsak element seznama dodali dogodek, ki proži to funkcijo. Po pregledu in dodajanju statusa pa nas algoritem še prestavi na stran za prikaz zapiska, kjer se prikaže zapisek glede na shranjene podatke znotraj sejne shrambe.

Pri Web Formsih pa nam je skupni algoritem povzročal preveč težav, saj ni možnosti dostopati do sejne shrambe s strani strežnika preko C# kode. Preverjanje statusa prebranosti zapiska smo zato na vsaki od treh strani implementirali samostojno. To pa ni najbolj optimalna rešitev, saj bi v primeru kakega popravka pri prikazu ali branju statusa prebranosti morali kodo spreminjati na treh različnih koncih, kar podaljša čas programiranja.

4.15 Izmenjava podatkov med JavaScript in strežnikom

Z modelom MVC smo podatke izmenjevali s pomočjo AJAX klicev na kontrolnikove akcije. Po želji smo lahko z AJAX klicem na akcijo kontrolnika poslali tudi dodatne parametre, ki so izoblikovali, kateri podatki naj bodo vrnjeni, ali pa kateri naj se shranijo. Ker smo celotno aplikacijo izdelali s pomočjo JavaScripta in zunanjih knjižnic, nismo imeli problemov s pridobivanjem podatkov.

Pri izdelavi s pomočjo Web Forms, kot je bilo že omenjeno zgoraj, smo pridobljene podatke s strani strežnika s pomočjo serializacije poslali v JavaScript funkcijo, kjer smo pri uporabniku zgenerirali ustrezno stran s pridobljenimi podatki in mu jo nato prikazali. Problem pa se je pojavil pri pošiljanju določenih parametrov za pridobitev ali shranjevanje podatkov, saj je Web Forms način strežniškega dela in izvajanja zahtev.

Vračanje podatkov s strani strežnika, ki je nato glede na te podatke izpolnil zahtevo, smo izvedli s pomočjo skritih polj (angl. hidden value).

Najprej smo na vsaki strani, kjer smo potrebovali parametre za pridobitev ali shranjevanje podatkov v HTML kodi dodali skrito polje:

```
<input type="hidden" id="HiddenValue" name="HiddenValue" />
```

Ob izvedbi določenega dogodka, smo klicali JavaScript funkcijo, ki je v skrito polje, katero ni vidno samemu uporabniku, ki stran pregleduje, dodala podatek ali pa skupino parametrov, ki jih potrebujemo za izvedbo zahteve. Zatem smo ročno klicali potrditev forme:

```
var hidden = document.getElementById("HiddenValue");
    hidden.value = themeID;
    var frm = document.getElementById("form1");
    frm.submit();
```

Po potrditvi forme se stran ponovno naloži in skrivno vrednost lahko pridobimo v dogodku ob nalaganju strani:

```
protected void Page_Load(object sender, EventArgs e)
{
    //preverjanje vrednosti skritega polja
    if (Request.Form["HiddenValue"] != null)
    {
        //pridobimo vrednost iz skritega polja
        string HiddenFiledValue = Request.Form["HiddenValue"].ToString();

        //nadalna izvedba s pridobljenimi parametri
    }
    else
    {
        //običajen razplet ob naložitvi strani
    }
}
```

Če strežnik ugotovi, da skrivno polje obstaja, lahko prilagodimo potek dogodkov in uporabniku ugodimo po zahtevi, kaj naj aplikacija naredi, glede na njegove zahteve.

4.16 Navigacija med stranmi

Ker je naša spletna stran sestavljena iz več različnih strani, je bilo potrebno izdelati tudi navigacijo med stranmi. Navigacija je lahko avtomatska, recimo ko se uporabnik odloči pregledati izbran zapisek in ga algoritem v ozadju samodejno prestavi na to stran iz seznama zapiskov ali pa odločitvena, ko se uporabnik sam odloči, da bi recimo zamenjal pregled iz vseh zapiskov na do zdaj še neprebrane zapiske. Pri MVC modelu smo za navigacijo uporabljali JavaScript kodo `window.location.assign("");` pri kateri uporabnika preusmerimo na stran zapisano znotraj narekovajev. Pri MVC-ju je bilo potrebno vpisati kontrolnik in akcijo, na katero se bomo prestavili `window.location.assign("../Theme/MyList");`. Pri Web Forms načinu pa smo uporabljali `Response.Redirect("ThemeShow.aspx");`, kjer smo navedli točno povezavo na stran. Tu se pojavi problem navigacije s pomočjo Response.Redirect-a.

Če smo za navigacijo uporabili le tega, smo se učinkovito prestavili na želeno stran, vendar v brskalnikovi vrstici za povezavo, se URL naslov ne spremeni. Kar pomeni, da če se uporabnik odloči osvežiti stran, ne bo ostal na strani, katera mu je trenutno prikazana, ampak se mu bo prikazala stran, katere URL naslov je še vedno vpisan. Ta problem smo rešili s pomočjo klicev JavaScript funkcij in uporabo navigacije kot pri MVC modelu, kjer je bilo to potrebno.

5 Primerjava lastnosti obeh pristopov

Ker je primerjava lastnosti obeh pristopov bolj ali manj subjektivne narave in se marsikateri razvijalec bodisi Web Forms aplikacije bodisi MVC aplikacije s spodnjimi trditvami ne bi strinjal se za kakršnekoli »napačne« ugotovitve opravičujem. Ugotovitve so spisane na podlagi izdelovanja obeh aplikacij, ki sta bili izdelani za namene diplomske naloge. Prav tako sta aplikaciji spisani na način, ki sem se ga zamislil in se verjetno kje pojavijo odstopanja od običajnega razvoja. S tem pa je mogoča tudi slabša učinkovitost delovanja spletne aplikacije in bi se lahko marsikatera funkcionalnost izdelala boljše in na drugačen način. Poudaril bi tudi to, da v načinu Web Forms ne uporabljamo AJAX-a klicev za pridobivanje podatkov, ampak le-te pridobivamo z osnovnimi »post back« metodami. Pri MVC modelu pa je tako ali tako na voljo pridobivanje podatkov samo s pomočjo AJAX klicev. Lahko bi se, za boljšo primerjavo odločili, da uporabljamo AJAX klice na obeh izdelanih aplikacijah, a ker smo obe aplikaciji izdelali na način, kot smo si ga sami izbrali, smo prišli do naslednjih razlik.

Na spodnji tabeli so opisane ugotovljene lastnosti s krajšim opisom, ter kateri od obeh pristopov je po mojem mnenju boljši in zakaj.

	Lastnost	**Opis**	**Pristop**	**Razlog**
1	Dolžina kode	Več kode	Web Forms	Pomanjkanje izkušenj
2	Velikost projekta	Večja zasedenost	MVC	Kompleksnost
3	Delo s podatki	Enostavnejše	MVC	Uporaba AJAXA
4	Način programiranja	Enostavnejše	MVC	Izkušenj
5	Hitrost odzivnosti	Počasnejši	Web Forms	Zaradi »post back«
6	Zahtevnost programiranja	Odvisno	Web Forms	Pomoči na spletu
7	Usmerjenost aplikacije	Odvisno	Deljeno	Glede na zahtevo
8	Hitrost programiranja	Odvisno	Deljeno	Izkušnje, potreba
9	HTML	Enostavnejše	MVC	Osnovni gradniki
10	Urejenost projekta	Lepša urejenost	MVC	Ločenih direktorijev

Tabela 3: Prikaz lastnosti med obema pristopoma izdelave spletne aplikacije

V nadaljevanju pa so podrobneje predstavljene zgoraj naštete lastnosti, ki so bile ugotovljene med razvojem aplikacije na oba načina.

5.1 Dolžina kode

Dolžina celotne spisane kode se med obema aplikacijama krepko razlikuje. Več kode je bilo potrebno za dosego enakega delovanja v aplikaciji izdelani s pomočjo Web Forms. Verjetno bi bilo potrebno tu upoštevati tudi to, da smo imeli nekoliko prednosti pri znanju izdelava aplikacije s pomočjo MVC modela. Razlika se potem nekoliko zmanjša, ko ugotovimo, da smo začeli z MVC načinom izdelave aplikacije, kjer še nismo imeli popolno začrtanih vseh funkcionalnosti in smo jih po potrebi dodajali in spreminjali obstoječe funkcije, medtem ko pa smo pri izdelavi aplikacije s pomočjo Web Forms že imeli dokončno sliko, kaj vse mora aplikacija vsebovati in ni bilo potrebnega poseganja v obstoječe funkcije, saj smo točno vedeli, kako morajo posamezne funkcije delovati.

5.2 Velikost končnega projekta

Po uspešno zaključeni izdelavi obeh aplikaciji smo se odločili preveriti zasedenost aplikacije na trdem disku. S presenetljivostjo ugotovimo, da veliko manj prostora zasede aplikacija izdelana z Web Forms modelom. Aplikacija izdelana s pomočjo MVC modela porabi **31 MB** prostora, medtem ko Web Forms le **7 MB** prostora. Taka razlika nastaja zaradi veliko več vključenih sistemskih knjižnic v MVC modelu, ki skrbijo za pravilno delovanje Model-View-Controller arhitekture, medtem ko Web Forms model samo prikazuje strani, vse pa se več ali manj dogaja na strežniku.

5.3 Pridobivanje in shranjevanje podatkov

Če primerjamo pridobivanje in shranjevanje podatkov znotraj naših rešitev, ugotovimo, da je delo s podatki veliko lažje s pomočjo AJAX klicev znotraj MVC modela. Znotraj komponente model, lahko uporabljamo enake prikaze podatkov tudi na drugih straneh. S čimer se izognemo ponovnemu pisanju funkcij za pridobitev ali pa shranjevanje podatkov. Medtem pa je pri Web Formsih potrebno pošiljanje parametrov prek skrivnih polj in obvezno potrjevanje obrazcev. Nato pa je ob začetku dogodka nalaganja strani pomembno preverjanje, če skrivno polje vsebuje kakšno vrednost. Glede na to se nato izvede pravilni potek dogodkov.

5.4 Način programiranje

Oba načina se razlikujeta tudi po načinu programiranja. Način programiranja je odvisen od zasnove modela, ki ga uporabljamo. Web Forms način dela zahteva izdelavo posamezne strani znotraj obrazca. Vsaka sprememba potrdi obrazec in zahteva strežniku, da se na to zahtevo pravilno odzove. V primeru, da želimo, da se neki dogodek ne odraža kot potrditev, je zato potrebno kodo posebej prirediti, s čimer povemo strežniku, naj zahteve ne obravnava. Ker je Web Forms nastal veliko pred MVC modelom, je njegov način dela veliko enostavnejši.

MVC pa ima v ozadju veliko bolj napredno arhitekturo. Z ločenimi komponentami, čigar ima vsaka komponenta določene naloge, ki jih mora opraviti, razporedimo breme in zagotovimo učinkovitejše delovanje. Na enem kontrolniku imamo lahko tudi več različnih pogledov, kar nam omogoča grupiranje strani. Glede na izbiro kontrolnika in pogleda se nato model odloči, katere podatke bo glede na uporabnikovo zahtevo vrnil. S tem imamo večjo preglednost naše aplikacije in v primeru nadgradenj lahko spremenimo samo eno komponento. Uporabnik preko gradnikov na pogledu sporoči kontrolerju kaj naj mu zagotovi. Kontroler te podatke posreduje modelu. Model pripravi podatke in jih vrne kontrolniku. Kontrolnik nato sporoči pogledu, da je nova stran s podatki pripravljena in naj se osveži. S tem modelom imamo določene naloge kaj katera od komponent naredi, medtem ko je v Web Forms vse združeno znotraj ene strani, kar naredi kodo nepregledno.

5.5 Hitrost delovanja

Pri hitrosti delovanje imamo v mislih odzivnost aplikacije na zahteve uporabnika. Hitrejšo odzivnost smo bili deležni z uporabo MVC modela, saj imamo komponente prilagodljive in bolj ali manj izpopolnjene do podrobnosti za posamezne naloge. S tem nam ni potrebno skrbeti, da bi recimo ena izmed komponent delovala bolje kot druga, saj je vsaka razvita neodvisno in nudi najboljše rezultate.

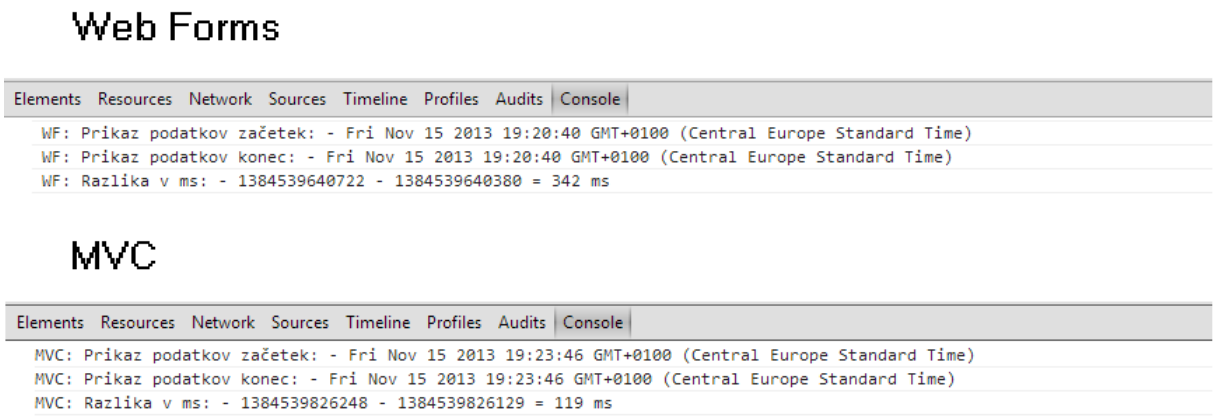
Pri Web Formsih pa te delitve ni oziroma niso tako učinkovite, saj je potrebno zagotavljanje enakovrednosti delovanja vseh funkcionalnosti zaradi ne ločenosti hkrati. Z ločenimi komponentami zmanjšamo tudi pogovarjanje uporabnika s strežnikom, kar pohitri delovanje spletne aplikacije.

Test smo izvedli s preprostim merjenjem časa ob navigaciji med stranmi. Pred spreminjanjem strani smo si shranili začetni datum s časom v sessionStorage, zatem izvedli navigacijo na drugo stran in pridobivanje podatkov za prikaz na izbrani strani. Ob prikazu strani smo shranili še končni čas in nato v brskalnikovi konzoli prikazali začetni in končni datum s časom. Zatem smo oba datuma datuma spremenili v milisekunde in odšteli razliko začetnega časa od končnega. S tem smo pridobili čas navigacije med stranmi v obliki milisekund. Test smo izvedli na treh različnih podatkovno zahtevnih straneh.

Pod pojmom podatkovno zahtevna stran je mišljena stran za prikaz, ki mora poleg osnovnega videza strani prikazati še pridobljene podatke in jih dodati strani za prikaz uporabniku. Merjenje časa smo izvajali izmenično.

Izbrali smo si stran in izmerili odzivnost z uporabo MVC ogrodja in nato še z uporabo Web Forms ogrodja. Ta postopek smo na vsaki strani ponovili trikrat. S tem smo zagotovili čim bolj podobne okoliščine za obe aplikaciji v danem času. Testiranje odzivnosti smo ugotavljali na lokalni uporabi (angl. localhost). Če se je, kateri od izmerjenih časov preveč razlikoval od povprečja drugih, smo test ponovili. Povprečni čas navigacije med stranmi v Web Forms je bil **343 ms**, pri uporabi MVC pa **126 ms**.

Na spodnji sliki (slika 13) sta prikazana naključna odzivna časa, med merjenjem hitrosti delovanja aplikacije, pri uporabi Web Forms in MVC ogrodja.



Slika 13: Razlika v hitrosti delovanja

5.6 Zahtevnost programiranja

Zahtevnost programiranja lahko merimo v različnih dejavnikih.
Mednje spadajo:

- Začetno znanje razvijalca,
- Število pomoči na medmrežju.

Začetno znanje razvijalca zelo vpliva na zahtevnost in učinkovitost programiranja. Razvijalec, ki je že uporabljal vsaj eno od tehnologij in še ni poizkusil druge, bo gotovo imel več težav pri programiranju še neznanega področja. Zahtevnost razvoja med obema aplikacijama je odvisna tudi od preteklih izkušenj. Če je razvijalec pred tem izdeloval namizne aplikacije, se bo verjetno bolje znašel z Web Formami, saj mu nudijo enostavno postavitev elementov. Če pa so predhodna znanja začetnika osnovne spletne izdelane s pomočjo beležnice, pa bo veliko bolj navdušen nad uporabo MVC modela, ki ohranja standardne HTML elemente in uporabo JavaScript spletne tehnologije.

Zahtevnost programiranja pa je odvisna tudi od števila pomoči, ki jih najdemo na spletu. Glede na to, da je Web Forms veliko starejši od MVC modela, lahko na spletu najdemo več pomoči in tudi rešitev za izdelavo aplikacije s pomočjo Web Forms. Splošna navada razvijalcev je, da če naletijo na problem, poiščejo na spletu če je že kdo predhodno imel podobne težave in če mu je kdo pomagal ali pa je sam prišel do rešitev. Zato pod zahtevnost programiranja uvrščamo tudi število omenjenih problemov na spletu.

5.7 Strežniško usmerjena aplikacija proti odjemalško usmerjeni

Pred razvijanjem projekta je pomembno, da se odločimo, kako usmerjeno aplikacijo želimo. Web Forms aplikacije je bolj strežniško usmerjena, medtem ko imamo pri MVC modelu zelo dobro podporo za odjemalniško usmerjeno aplikacijo. Tudi naša aplikacija je bol odjemalniško usmerjenja in nam izdelava za strežniško usmerjenost porabila veliko več časa. Z odjemalniško usmerjenostjo aplikacije, lahko končnim uporabnikom ponudimo bolj dinamične spletne strani, z uporabo JavaScripta pa dosežemo, da se aplikacija skorajda v celoti izvaja pri odjemalcu, s čimer sprostimo strežniške zahteve in pohitrimo izvajanje aplikacije. Strežniško usmerjena aplikacija pa skrbi, da končni uporabnik dobi samo izdelano stran, s čimer mu koda v ozadju ostaja skrita.

5.8 Hitrost programiranja

Hitrost programiranje je težko deliti, na kateri način programiranja je hitrejši in kateri ne. Na hitrost programiranja vpliva več dejavnikov. Od predhodnega znanja uporabnika, do izkušenj uporabnika ter tudi od načina razmišljanja. Seveda pa imamo včasih tudi kdaj slab dan in se produktivnost nekoliko zmanjša. Z gotovostjo pa lahko trdimo, da uporaba nekaterih funkcionalnosti, ki jih Visual Studio ponuja pri gradnji Web Forms aplikacij, občutno pospešijo programiranje. Visual Studio nam v Web Forms-ih ponuja enostavno dodajanje gradnikov na stran s pomočjo drag in drop vmesnika. Gradnik izberemo iz orodjarne in ga postavimo na stran. Omogoča nam tudi urejanje postavitve in videza dodanega gradnika. Do teh lahko pridemo z urejanjem lastnosti gradnika, obenem pa se nam spremembe aktivno osvežujejo in so takoj vidne razvijalcu. Na voljo nam je tudi dodajanje povezave do podatkovne baze in izbira podatkov z nekaj miškinimi kliki. Visual Studio nato na podlagi zabeleženih nastavitev ustrezno generira kodo. Zelo zanimiva pa je tudi funkcionalnost, ki razvijalcu s pomočjo »dvoklika« na gradnik izdela osnovni dogodek. Zatem samo še sledi dodajanje funkcije, kaj naj se ob tem dogodku zgodilo, kar enormno pospeši izdelavo spletnih aplikacij.

V MVC modelu ni na voljo nič od zgoraj naštetega, a to še ne pomeni, da je v kakršnem koli zaostanku v primerjavi z Web Forms. Namreč večina programerjev se kljub možnosti olajšanja dodajanja gradnikov raje zateka k tradicionalnim metodam programiranja, kar pomeni, da vse gradnike in dogodke raje vpisuje ročno, s čimer ima večji pregled nad kodo in dogajanjem z njo.

5.9 HTML

HTML elementi se v obeh načinih razvoja nekoliko razlikujejo. Web Forms gradniki so podobni običajnim HTML gradnikom, z dodatkom atributa asp na začetku. Mogoče je urejanje videza in postavitve gradnike s pomočjo CSS sloga, vendar nam ta ni ponujen z uporabo tehnologije šestega čuta (IntelliSense) prav tako pa se pojavljajo tudi odstopanja pri obravnavi sloga med klasičnim in Web Forms HTML gradniki. Urejamo pa jih lahko tudi s posebej prirejenimi lastnostmi gradnikov, zato tudi razlika med klasičnimi gradniki. Web Forms uporabljajo nadgrajeno različico HTML elementov, katera na prvi pogled deluje nekoliko čudno kot običajni HTMl elementi. Pri avtomatskem dodajanju gradnikov s pomočjo Visual Studio orodjarne HTML elementi dobijo na začetku značko asp, dodajanje klasičnih gradnikov pa ni mogoče, saj je razvoj prilagojen gradnji Web Form obrazcev. Razvijalci se lahko odločijo ali bodo te gradnike uporabljali ali pa bodo uporabljali klasične. Vendar tu moramo biti previdni, saj nam klasični gradniki pri gradnji strani lahko ponagajajo in z njihovo postavitvijo ne dosežemo želenih rezultatov. Razvijalci običajnih spletnih strani so bolj navajeni klasičnih HTML gradnikov in z gotovostjo lahko trdim, da bi jim ASP gradniki povzročali nemalo težav. Pri pregledu strani pa se ASP gradniki prevedejo v običajen HTML jezik. Lahko pa se zgodi, da nekateri elementi dobijo pred identifikatorji gradnikov še dodatek. MVC model pa omogoča uporabo klasičnih HTML elementov, s katerimi so bolje seznanjeni vsi razvijalci spletnih strani in so tudi osnova razumevanje ASP gradnikov.

5.10 Urejenost projekta

Na spodnji sliki (slika 14) je prikazana razlika med urejenostjo projekta v obeh aplikacijah. S slike je razvidno, da je urejenost projekta lepše organizirana pri uporabi MVC modela, saj imamo več različnih pogledov združenih znotraj iste mape, ki se kreira sama z imenom kontrolnika, na katerega so te pogledi vezani. Ko dodajamo pogled znotraj kontrolnika, lahko na enostaven način dodamo pogled znotraj matične mape. Z desnim klikom na pogled izberemo dodaj pogled, ki se kreira znotraj mape. Na ta način lahko vidimo vse strani, ki uporabljajo skupni model za upravljanje podatkov, ter kateri kontrolnik skrbi za njihovo pravilno delovanje. Ob tem smo še združili JavaScript datoteke, ki se uporabljajo v določenih pogledih znotraj istega kontrolnika, s čimer prihranimo čas iskanja datotek, ki so vezane na določeno stran. V Web Forms aplikaciji pa so strani razpršene skozi celoten projekt. Na novo dodana datoteka se avtomatsko razvrsti glede na zaporedje imen ostalih datotek. Nelogično bi bilo tudi dodajanje map za posamezne strani, saj v povprečju ena stran vsebuje eno JavaScript datoteko za delovanje. Če jo poimenujemo podobno kot samo ASPX stran, se ta avtomatsko doda v projektu zanjo.

Z urejenostjo projekta močno olajšamo iskanje obstoječih strani in hitrejše dodajanje nadgradenj. Prav tako tudi ob dodajanju novih strani lahko predvidevamo, kam se bo nova datoteka shranila, pri čemer nam ni potrebno pregledovati po zaporedju imen datotek. Z urejenostjo projekta tudi zagotovimo sodelavcu ali svojemu nasledniku preprostejšo uporabo projekta.

Slika 14: Primerjava urejenosti map med obema načinoma izdelave spletne stran

6 Zaključek

V diplomskem delu sem predstavil MVC model za izdelavo spletnih strani. Opisal sem, kako vsaka od komponent opravlja svoje delo, ter kako se med seboj povezujejo. Prikazal in opisal sem tudi razlike, ki se pojavljajo z uporabo Web Forms aplikacij. Za namene prikaza MVC modela sem izdelal tudi aplikacijo z Microsoftovimi ogrodji, v kateri sem uporabil še novejše spletne tehnologije, ki se uporabljajo za izdelavo modernih spletnih strani. Za primerjavo MVC modela z drugimi načini izdelave spletnih strani pa sem izdelal še podobno aplikacijo s pomočjo Web Forms tehnologije. Na primerih sem prikazal, kje se pojavljajo največje razlike pri razvoju spletnih aplikacij na oba omenjena načina, ter v katerih lastnostih je načina uporabe med obema aplikacijama najbolj viden.

Dejstvo je, da lahko z obema načinoma izdelave spletne strani dosežemo enake rezultate, le če se za to dovolj potrudimo, se prilagodimo in izrazimo željo po izdelavi kvalitetne in lepe spletne aplikacije. Vsak od načinov prinaša svoje prednosti in slabosti, a z vloženim trudom in pridobljenim znanjem lahko končni rezultat izdelave aplikacije pripeljemo do te mere, da običajni uporabnik ne bo zaznal razlik med eno ali drugo spletno aplikacijo.

Če strnemo celotno diplomsko delo in bi si morali za naslednjo spletno aplikacijo izbrati, v katerem od obeh načinov bomo razvili novo spletno aplikacijo, bi se na podlagi vsega napisanega odločil za ponovni razvoj s pomočjo MVC modela, saj učinkovito združuje vse novejše spletne tehnologije, ki poskrbijo za atraktivne spletne strani, hkrati pa zagotavlja uporabo vseh osnovnih ključnih gradnikov spletnih strani in boljše povezovanje s HTML 5 tehnologijo ter njenimi funkcionalnostmi, ki še prihajajo in so že na voljo vsem razvijalcem.

7 Priloge

1 Opis funkcionalnosti: Registracija

Preden želimo uporabljati našo aplikacijo, je za to potrebno imeti račun za uporabo. Registracija uporabnika je dostopna na začetni strani. Pri ustvarjanju računa je potrebno vnesti kar nekaj podatkov o uporabniku. Poleg osnovnih podatkov uporabnika – ime, priimek, naslov in pošte, si uporabnik izbere še uporabniško ime za prikaz in geslo za prijavo v sistem. Za zagotovitev spremembe uporabniških podatkov pa je potrebno vnesti še tako imenovano skrivno vprašanje in skrivni odgovor na to vprašanje. Pošto določimo iz seznama pošt, ki v bazi nastopa v paru številka pošte – naziv pošte. S tem, kot smo že omenili, preprečujemo neenakost podatkov. Po vnešenih podatkih se pregleda pravilnost izpolnjenih podatkov. To pomeni, da algoritem preveri, ali so vsi podatki izpolnjeni in se oba obrazca za geslo ujemata, s čimer se uporabniku zagotovi, da je izbrano geslo pravilno vpisano. Po uspešno opravljeni registraciji, aplikacija preusmeri uporabnika na začetno stran, kjer mu je sedaj na voljo prijava v sistem.

Slika 15: Stran za registracijo uporabnika v aplikaciji Note Wiki

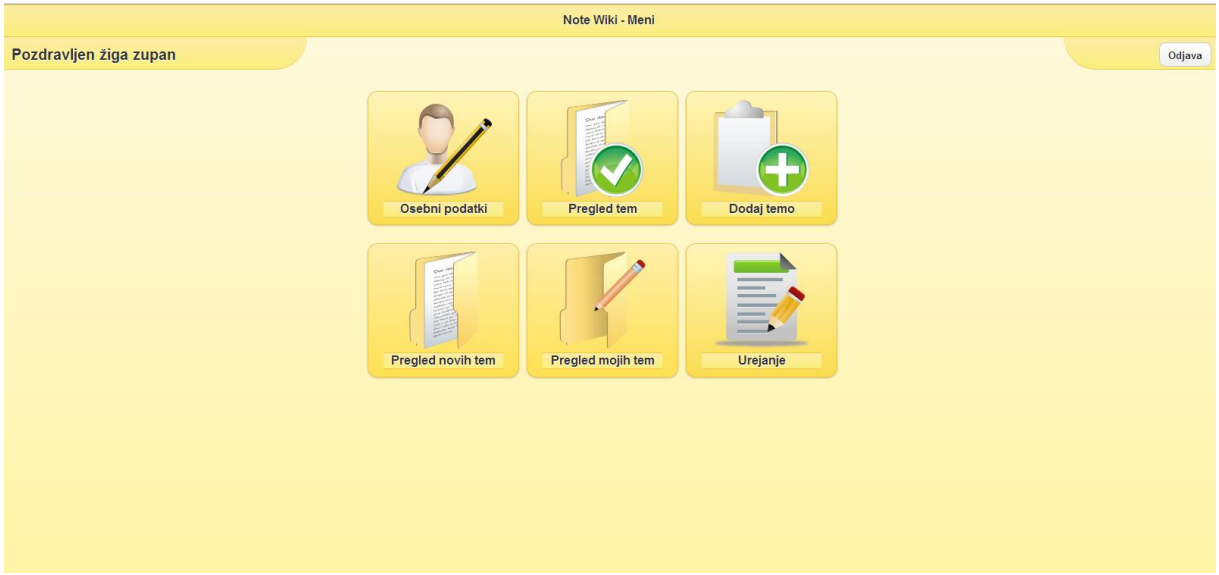
2 Opis funkcionalnosti: Prijava

Po uspešni registracija uporabniškega računa, ali ob že obstoječem računu, je uporabniku na voljo prijava v sistem. Uporabnikova naloga je vpis lastnega uporabniškega imena in gesla. Algoritem v ozadju pa v podatkovni bazi preveri, ali se vnosa ujemata. V primeru neujemanja se uporabnika obvesti o napačno vpisanem uporabniškem računu. V primeru uspešne prijave pa je uporabnik uspešno prijavljen v sistem in ga aplikacija preusmeri na osnovno stran aplikacije.

Slika 16: Stran za prijavo uporabnika ob neuspešni prijavi v aplikaciji Note Wiki

3 Opis funkcionalnosti: Meni

Menijska stran je tudi osnovna stran naše aplikacije. Z enostavnim menijem lahko na hiter način izbiramo med različnimi moduli aplikacije. Z menijske strani lahko dostopamo do strani za spremembo uporabniških nastavitev, do pregleda zapiskov – bodisi lastnih zapiskov, zapiskov ostalih uporabnikov, ali pa do še do sedaj nepregledanih zapiskov s strani trenutno prijavljenega uporabnika. Menijska stran nam tudi omogoča hitro dodajanje novega zapiska in dostop do administratorske strani, s katere lahko spreminjamo vse obstoječe teme in registrirane uporabniške račune. Na voljo pa je tudi odjava s sistema, katera pobriše trenutno shranjene uporabniške podatke in preusmeri uporabnika na začetno stran.



Slika 17: Menijska stran aplikacije Note Wiki

4 Opis funkcionalnosti: Spremembe uporabniških podatkov

Z menijske strani lahko dostopamo do sprememb uporabniških podatkov trenutno prijavljenega uporabnika. Na začetku imamo predogled nekaterih svojih predhodno izpolnjenih podatkov, ki smo jih dodali pri registraciji in jih trenutno še ne moremo urejati. Skriti podatki uporabniku so seveda geslo in skrivno vprašanje ter odgovor nanj. Če želimo podatke trenutnega uporabnika spreminjati, moramo zagotoviti ustrezno avtorizacijo uporabnika. To pregledujemo s pomočjo skritega vprašanja in skrivnega odgovora. Ko uporabnik želi urejati trenutne podatke ga najprej povprašamo o odgovoru na skrivno vprašanje. Če se odgovor na skrivno vprašanje ujema z odgovoram, določenim pri registraciji, se uporabniku pokažejo še ostali podatki in se mu omogoči urejanje vseh podatkov. Če pa uporabnik ne vpiše točnega odgovora, je njegov dostop do urejanja podatkov še vedno zaščiten in nima pravic za urejanje le teh.

Slika 18: Stran za urejanje osebnih podatkov v aplikaciji Note Wiki

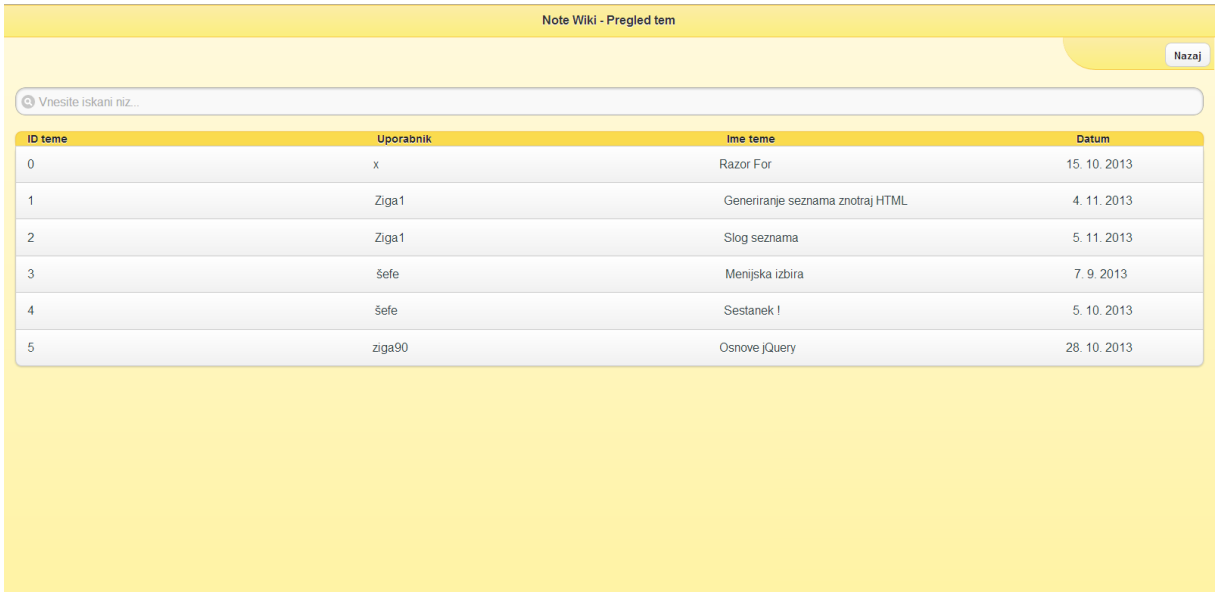
5 Opis funkcionalnosti: Pregled zapiskov

Z menijske strani pa lahko dostopamo tudi do treh različnih načinov pregledovanja zapiskov. Vsi trije načini se nekoliko razlikujejo po načinu prikaza zapiskov, a nekaj funkcionalnosti ostaja enakih. Vsak od prikazov zapiskov ima na vrhu strani iskalnik zapisov, ki išče po podatkih, ki so prikazani v seznamu trenutnih zapisov. Z vsako vneseno črko se generira nov seznam zapisov, kateri podatki vsebujejo zapisan niz. Če noben zapis ne ustreza vnesenemu nizu v iskalniku, se nam izpiše, da ne obstaja noben zapis s temi vrednostmi. Iščemo lahko po datumu vnosa zapiska, uporabniku ki je vnesel zapis, po številki zapisa ali pa po naslovu zapiska. Z izbiro izbrane teme pa se sprožita še dva dogodka. Prvi preveri ali je bila izbrana tema že pregledana s strani trenutnega uporabnika. Če strežnik s pomočjo podatkovne baze ugotovi, da tema še ni bila pregledana, jo le ta doda uporabniku v bazo prebranih zapiskov. S tem dejanjem vplivamo na prikaz še neprebranih zapiskov, če se le odločimo za tak prikaz. Za tem sledi drugi dogodek, ki preusmeri uporabnika na prikaz izbranega zapiska.

5.1 Opis funkcionalnosti: Pregled vseh zapiskov

Gre za prikaz vseh zapiskov v bazi ne glede na uporabnika, saj lahko v nekem podjetju dela na istem projektu več ljudi, s čimer si pomagajo na že ugotovljenih rešitvah. Prikazujejo se torej vsi aktivni zapisi. Določeni zapisi so lahko uporabniku tudi skriti, če se je za to odločil avtor zapiska.

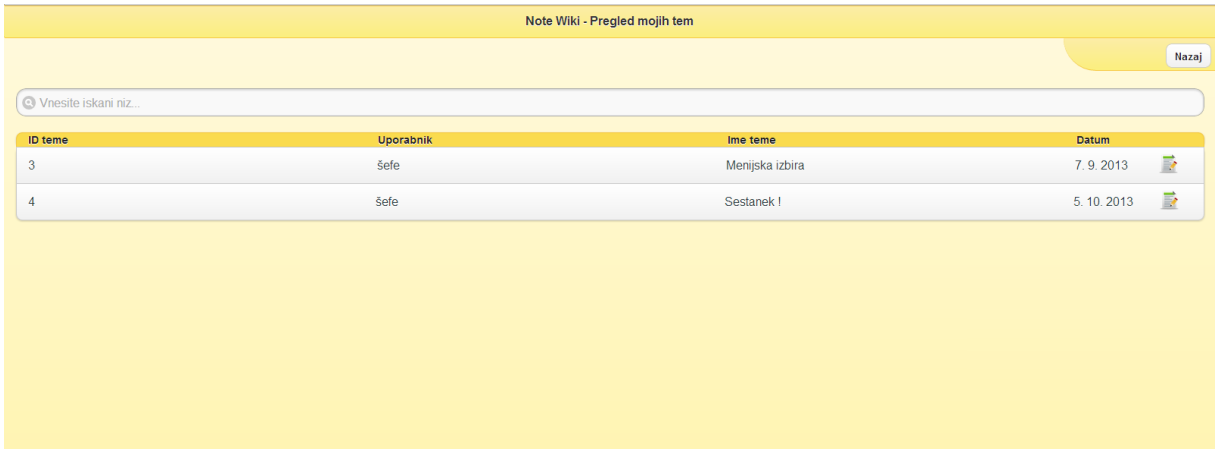
Primer sodelavec nas povpraša o problemu, zanima ga, kako smo ga rešili. Veliko več časa porabimo, če mu razložimo postopek in poiščemo ter pokažemo primer v kodi. Lahko pa mu samo omenimo, da smo ta problem že rešili in da ga lahko poišče v Note Wiki spletni aplikaciji, kjer je primer rešitve že vpisan. Povemo mu še ključno besedo po kateri naj zapisek poišče. Ko ga najde lahko v opisu, če smo ga dodali, prebere kaj ta koda naredi in kakšen je postopek uporabe. Če se z ugotovitvijo strinja lahko uporabi del kode, ki smo ga zabeležili in pregleda ugotovitve na povezavi, ki smo jo dodali. S tem lahko uporabnik ponovi del kode, ki se je že uporabljal na drugih delih programa, kar pripomore h konsistentnost kode.



Slika 19: Pregled vseh aktivnih zapiskov v aplikaciji Note Wiki

5.2 Opis funkcionalnosti: Pregled lastnih zapiskov

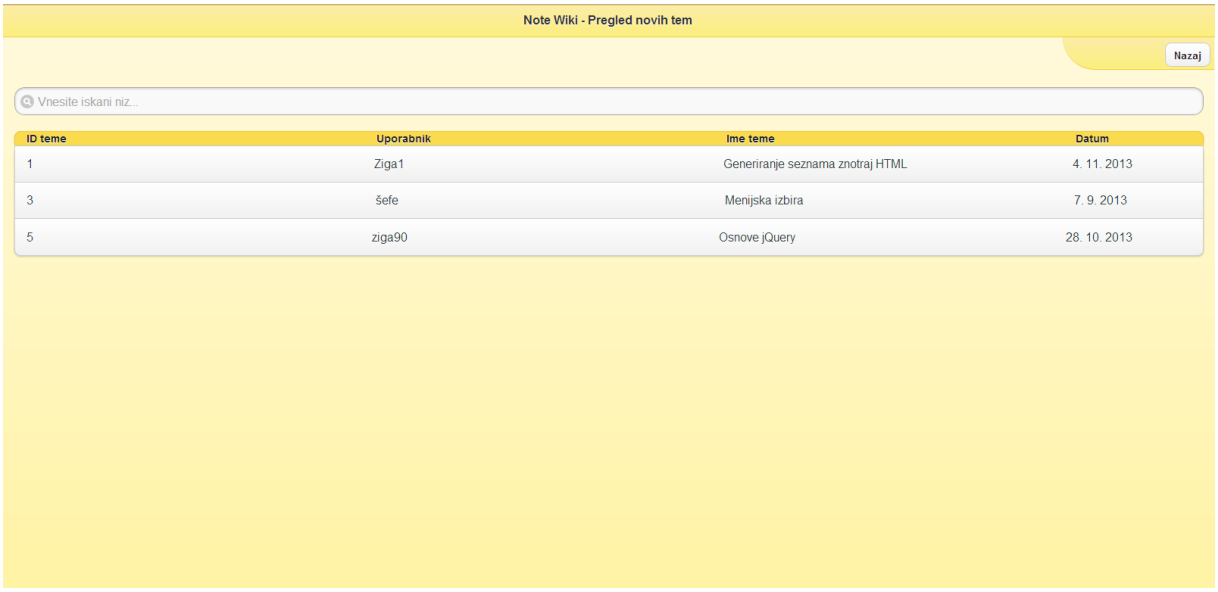
Pregled lastnih zapiskov nam omogoča pregled zapiskov le enega uporabnika. Tu se nam prikažejo zapiski, ki jih je zapisal trenutno aktivni uporabnik – mi sami. S tega pregleda nam je dodana, poleg pregleda, tudi možnost urejanja že zapisanih zapiskov. To pomeni, da vse zapiske, ki smo jih ustvarili sami, lahko tudi popravljamo. Z izbiro urejanja zapiska se prestavimo na novo stran, kjer lahko izbrani zapisek popolnoma spremenimo. Spremenimo mu lahko naslov in izberemo, kateri izbrani tipi se bodo po novem prikazali v pregledu. Podatke določenega tipa lahko tudi popolnoma spremenimo. Shranjen urejen zapisek se nato shrani in se ponovno pokaže kot neprebran vsem ostalim uporabnikom aplikacije.



Slika 20: Pregled vseh zapiskov aktivnega uporabnika v aplikaciji Note Wiki

5.3 Opis funkcionalnosti: Pregled nepregledanih zapiskov

S pregledom nepregledanih zapiskov uporabniku omogočamo vpogled zapiskov s strani vseh uporabnikov z razliko, da trenutni uporabnik ne vidi svojih zapiskov, saj se mu je status prebranega zapiska dodal že ob kreiranju, hkrati pa ne vidi niti zapiskov, ki jih je trenutni uporabnik že pregledal. Če povzamemo, uporabnik lahko vidi zapiske vseh uporabnikov, ki jih še ni pregledal, ali pa tistih zapiskov, ki so bili ponovno spremenjeni, čeprav jih je lahko že pregledal.



Slika 21: Pregled še neprebranih zapiskov aktivnega uporabnika v apikaciji Note Wiki

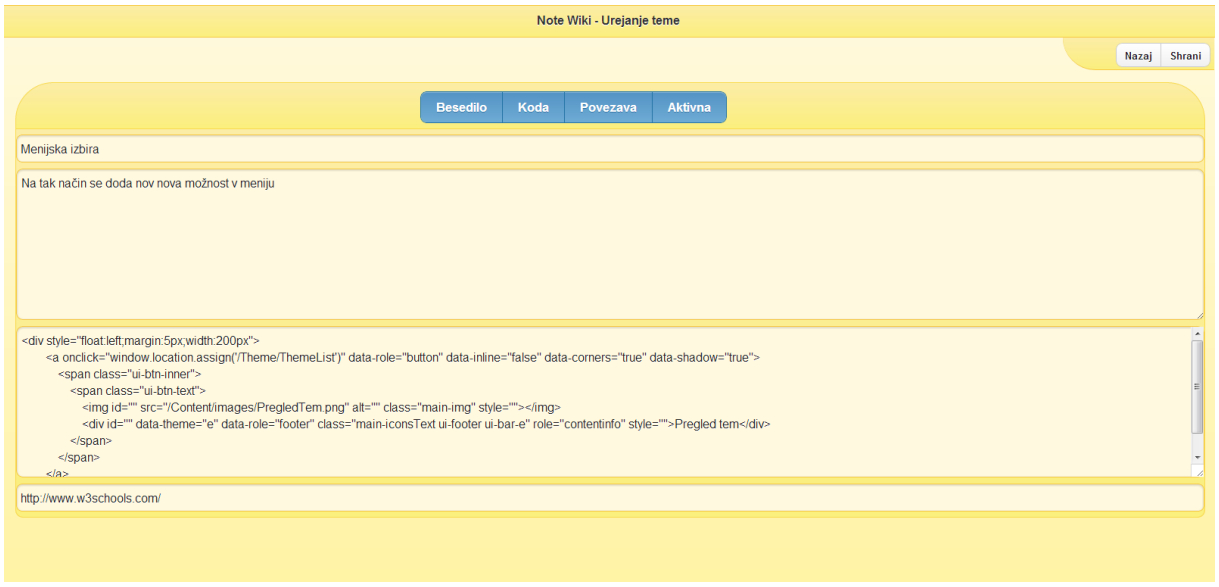
6 Opis funkcionalnosti: Prikaz in urejanje zapiska

Gre za dokaj podobni strani, ki uporabniku omogočata pregled oziroma urejanje obstoječega zapiska. Obe strani prikazujeta podatke o zapisku, čigar podatki so shranjeni v podatkovni bazi. Razlika med stranema je v različnih funkcionalnostih.

Pri pregledu zapiska uporabnik nima dostopa do urejanja zapiska, vsebuje pa zunanjo knjižnico prettifier, ki olepša kodo in jo naredi ključnemu uporabniku bolj razumljivo in lepše prikazano. Prav tako pa se s klikom na povezavo v novem zavihku prikaže stran, ki je bila zapisku dodana, v novem zavihku.

Pri urejanju pa ima uporabnik, ki je ta zapisek kreiral možnost urejanja, vendar so vsi podatki prikazani v navadni tekstovni obliki s pomočjo input HTML elementov. Urejanje zapiska je mogoče samo uporabniku, ki je ta zapisek ustvaril, s čimer preprečujemo zlo namerna dejanja, da bi kakšen drug uporabnik iz zavisti želel določene podatke prikriti in otežiti nalogo uporabniku, ki želi uporabljati že obstoječe zapiske. Po spremembi zapiska se vsem ostalim uporabnikom v bazi doda status neprebranega zapiska.

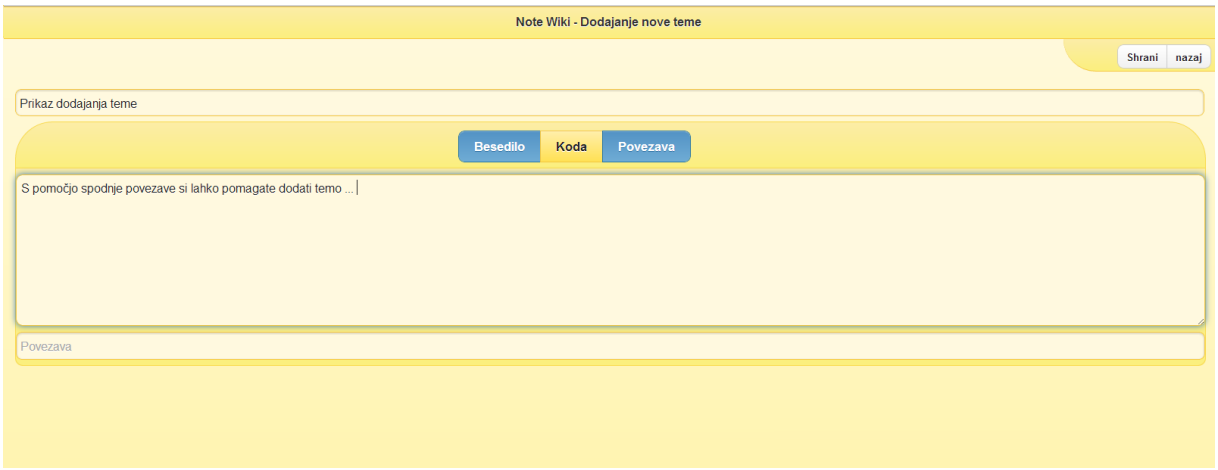
Slika 22: Stran za prikaz izbranega zapiska znotraj v aplikaciji Note Wiki



Slika 23: Stran za urejanje izbranega zapiska znotraj v aplikaciji Note Wiki

7 Opis funkcionalnosti: Dodajanje zapiska

S pomočjo forme za dodajanje zapiska lahko hitro in učinkovito dodamo nov zapisek. Dodelimo naslov zapiska, vnesemo opis zapiska, s čimer razložimo, bodisi postopek uporabe, bodisi kaj ta funkcija sploh naredi ali pa kaj drugega. Na voljo imamo tudi dodajanje kode. Priporočljivo je, da kodo v urejevalniku prej uredimo. To pomeni, da poskrbimo za pravilnost gnezdenja funkcije, s čimer je koda veliko lepša pri samem prikazu zapiska. Lahko pa dodamo tudi povezavo na drugo stran. S tem ugodimo uporabniku, ki bi to uporabljal, a ni povsem prepričan zakaj je koda spisana na tak način, če se želi prepričati o učinkovitosti delovanja podane kode. Po shranitvi zapiska se avtorju zapiska doda status prebranosti zapiska, ostalim uporabnikom pa se status spremeni v neprebranega, zaradi katerega ga lahko prikazujemo pod neprebranim seznamom zapiskov. Prav tako je urejanje zapiska omogočena samo uporabniku, ki je ta zapisek ustvaril.

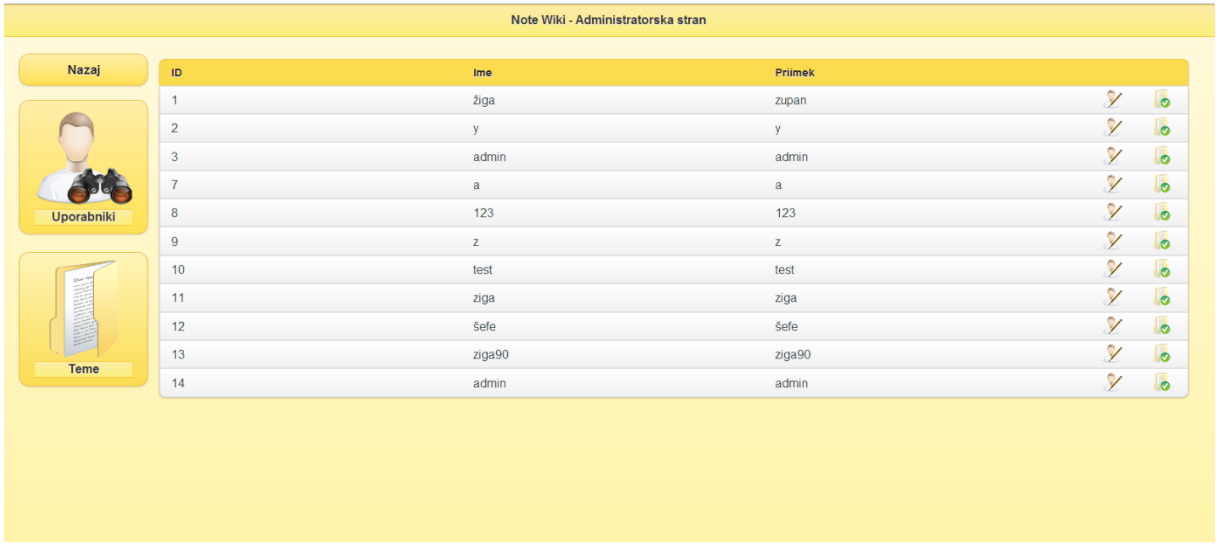


Slika 24: Primer dodajanja zapiska v aplikaciji Note Wiki

8 Opis funkcionalnosti: Administrativna stran

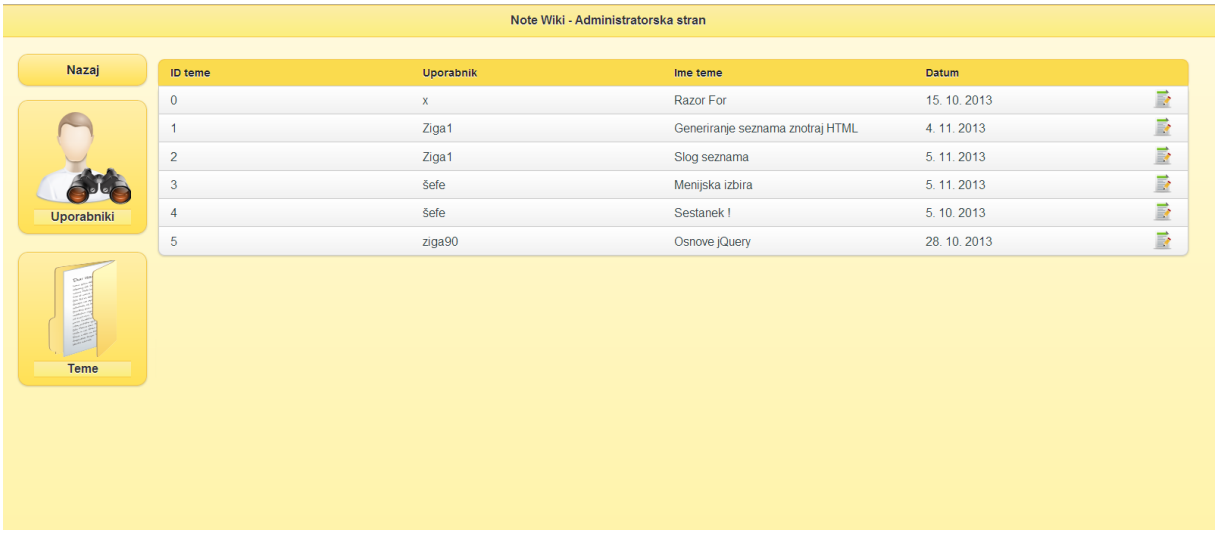
Glavni urejevalci strani (administratorji) imajo na voljo dva osnovna pregleda. Omogočamo jim pregled registriranih uporabnikov in pregled vseh zapiskov.

Pri pregledu registriranih uporabnikov omogočamo pregled osnovnih podatkov uporabnika, urejanje uporabnika in prikaz seznama zapiskov, ki jih je ustvaril izbran uporabnik. Urejanje in prikaz uporabnikov je zelo podobno prikazu osnovnih podatkov aktivnega uporabnika, le da tu urejevalec strani lahko izbira, za katerega uporabnika želi pregled podatkov ali pa urejanje podatkov. Z izbiro pregled zapiskov izbranega uporabnika pa se mu izpišejo vsi zapiski, ki jih je ta uporabnik do tega trenutka shranil. Ni pomembno ali jih je delil z drugimi uporabniki ali ne, saj ima administrator pogoje, da vidi vse zapiske. Nad zapiski lahko nato administrator izvaja dve akciji. Pregled le-teh zapiskov ali pa urejanje zapiskov izbranega uporabnika.



Slika 25: Administratorski pregled uporabnikov v aplikaciji Note Wiki

Pri pregledu vseh zapiskov pa so administratorju na voljo prav vsi zapiski shranjeni v bazi. Ni pomembno ali je neki zapisek uporabnika aktiven ali ne, v vsakem primeru se administratorju ta zapisek prikaže in mu dovoli, da popolnoma spremeni zapisek, ali pa ga samo naredi vidnega ostalim uporabnikom. Administratorske pravice so v tem primeru enake kot pravice uporabnika, ki je ta zapisek ustvaril. Če administrator zapisek spremeni, se vsem uporabnikom prikaže spremenjen zapisek v seznamu še neprebranih zapiskov (tudi uporabniku, ki je avtor tega zapiska). Ob tem pa se ne spremeni avtor zapiska, kar pomeni, da je zapisek še vedno viden pod pregledom lastnih zapiskov uporabnika, ki je ta zapisek kreiral, le da se mu, dokler je ponovne ne pregleda, prikazuje v seznamu še neprebranih zapiskov.



Slika 26: Administratorski pregled zapiskov v aplikaciji Note Wiki

8 Literatura in viri

[1] (2013) Netcraft, »Web Server Survey«
Dostopno na: http://news.netcraft.com/archives/2013/11/01/november-2013-web-server-survey.html

[2] (2013) Wikipedia, »Model-View-Controller«
Dostopno na: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[3] (2013) amix, »Model View Controller: History, theory and usage«
Dostopno na: http://amix.dk/blog/post/19615

[4] (2013) slideshare, »Thin controllers-fat models«
Dostopno na: http://www.slideshare.net/damiansromek/thin-controllers-fat-models-proper-code-structure-for-mvc

[5] (2013) Wikipedia, »Comparison of web application frameworks«
Dostopno na:
http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks#Comparison_of_features

[6] (2013) Wikipedia, »ASP.NET MVC Framework«
Dostopno na: http://en.wikipedia.org/wiki/ASP.NET_MVC_Framework

[7] (2013) dotnetcurry, »Understanding Routing in ASP:NET MVC«
Dostopno na: http://www.dotnetcurry.com/ShowArticle.aspx?ID=814

[8] (2013) dotnet.dotzone, »The History of ASP.NET MVC, So Far«
Dostopno na: http://dotnet.dzone.com/articles/history-aspnet-mvc-so-far

[9] Jose Rolando Guay Paz, Beginning ASP.NET MVC 4: *Introducing ASP.NET MVC 4*, New York, 2013, str. 6

[10] (2013) dotnet-tricks, »Difference between Razor View Engine and ASPX View Engine«
Dostopno na: http://www.dotnet-tricks.com/Tutorial/mvc/91JM151212-Difference-Between-Razor-View-Engine-and-ASPX-View-Engine.html

[11] (2013) tutoriz, »RAZOR VIEW VS ASPX VIEW ENGINE | DIFFERENCE«
Dostopno na: http://tutoriz.com/Thread-RAZOR-VIEW-VS-ASPX-VIEW-ENGINE-DIFFERENCE

[12] Adam Freeman, Pro ASP.NET MVC 4: *Working with Razor*, New York, 2012, str. 104

[13] (2013) Wikipedia, »Microsoft Visual Studio«
Dostopno na: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio

[14] (2013) Wikipedia, ».NET Framework«
Dostopno na: http://en.wikipedia.org/wiki/.NET_Framework

[15] (2013) MSDN, »Language Interoperability«
Dostopno na: http://msdn.microsoft.com/en-us/library/a2c7tshk(v=vs.71).aspx

[16] (2013) Wikipedia, »Microsoft SQL Server«
Dostopno na: http://en.wikipedia.org/wiki/Microsoft_SQL_Server

[17] (2013) Wikipedia, »Microsoft SQL Server Managment Studio«
Dostopno na: http://en.wikipedia.org/wiki/SQL_Server_Management_Studio

[18] (2013) googlecode, »JavaScript code prettifier«
Knjižnica dostopna na: http://google-code-prettify.googlecode.com/svn/trunk/README.html