UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

DANIJEL HRVAĆANIN

Primerjava arhitektur Model-Pogled-Kontroler

DIPLOMSKO DELO NA
VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: doc. dr. Janez Demšar

Ljubljana, 2012

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: 00179/2011
Datum: 04.11.2011

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

Kandidat: **DANIJEL HRVAĆANIN**

Naslov: **PRIMERJAVA ARHITEKTUR MODEL-POGLED-KONTROLER**
COMPARISON OF MODEL-VIEW-CONTROLLER ARCHITECTURES

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Arhitektura model-pogled-kontroler je osnova večine modernih ogrodij za izdelavo dinamičnih spletnih strani. Skladno z njo zasnujemo aplikacijo v treh delih; model hrani podatke, pogled skrbi za komunikacijo z uporabnikom, v kontrolerju pa se navadno nahaja večina programske logike.

V diplomi kratko opišite osnovne značilnosti arhitekture model-pogled-kontroler, nato pa jo predstavite na izbranem vzorcu ogrodij. Pri primerjavi ogrodij bodite pozorni na različne kriterije, ki nas vodijo pri njihovem izboru, kot so čas priučevanja, skalabilnost in ekonomski vidiki uporabe posameznega ogrodja.

Mentor:

doc. dr. Janez Demšar

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Danijel Hrvaćanin

z vpisno številko 63050259

sem avtor diplomskega dela z naslovom:

Primerjava arhitektur Model-Pogled-Kontroler

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom

  doc. dr. Janeza Demšarja,
- 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 10.04.2012 Podpis avtorja: _____

ZAHVALA

Zahvalil bi se vsem, ki so mi na poti študija, na kakršenkoli način stali ob strani, me spodbujali, podpirali ter pomagali razumeti določene stvari. Zahvale gredo tudi profesorjem na Univerzi, ki so mi s svojim pristopom k poučevanju dali nova znanja ter me navdušili za določene tehnologije.

Med vsemi pa bi najbolj izpostavil oba starša, ki sta ves čas verjela vame, mi stala ob strani tako psihično kot tudi denarno.

SEZNAM KRATIC

MVC – Model-View-Controller – arhitektura oziroma vzorec, s katerim lahko zasnujemo spletne aplikacije.

RAD – Rapid Application Development – metodologija, ki se uporablja pri razvoju aplikacij, pri kateri se izognemo dolgotrajnemu planiranju.

API – Application Programming Interface – programska koda, ki predstavlja strukturo in je potrebna pri komunikaciji različnih komponent med seboj.

MAPPER – abstraktni razred, kjer definiramo abstraktne funkcije oziroma operacije po navadi so to CRUD metode.

CRUD – Create, Read, Update and Delete – štiri osnovne funkcije, ki definirajo trajno podatkovno shemo.

WAMP – Windows Apache MySQL PHP – platforma za operacijski sistem Windows, kjer se obenem namestijo Apache strežnik, podatkovni sistem MySQL ter skriptni jezik PHP.

PHP – PHP Hypertect Preprocessor (Personal Home Page Tools) – odprtokodni programski jezik, ki se uporablja za razvoj dinamičnih spletnih vsebin.

JVM – Java Virtual Machine – navidezna plast, ki izvaja operacijsko kodo prevedeno iz izvorne kode napisane v programskem jeziku Java.

CDDL – Common Development and Distribution License – brezplačna licenca, ki jo je razvil Sun Microsystems in temelji na Mozillini brezplačni in odprtokodni licenci.

GPLv2 – General Public License version 2 – brezplačna programska licenca, ki omogoča, da je program brezplačen, tudi če je skozi čas nadgrajen oziroma razširjen.

HTML – Hyper Text Markup Language – označevalni jezik za izdelavo spletnih strani.

CSS – Cascading Sytle Sheets – podloge, predstavljene v obliki preprostega slogovnega jezika, ki skrbijo za prezentacijo spletnih strani.

ASP – Active Server Pages – Microsoftovo okolje, v katerem lahko programiramo dinamične spletne strani

HTTP – HyperText Transfer Protocol – protokol za prenos informacij na spletu

HTTPS – HyperText Transfer Protocol Secure – kombinacija HTTP s protokolom SSL/TLS. Ponuja zakodirano komunikacijo in varno identifikacijo s strežnikom.

FTP – File Transfer Protocol – standarden omrežni protokol za prenos datotek z gostitelja na gostitelja.

FTPS – File Transfer Protocol Secure – k standardnemu protokolu FTP ponuja še kriptografske protokole TLS ali SSL za večjo varnost.

SMTP – Simple Mail Transfer Protocol – standard za pošiljanje elektronskih pošt preko internetnih omrežij.

NNTP – Network News Transfer Protocol – protokol za razpošiljanje Usenet novic med novičarskimi strežniki.

XML – eXtensible Markup Language – način za definiranje pravil, ki je berljiv človeku kakor tudi napravam.

JSON – JavaScript Object Notation – odprt način predstavitve podatkov v obliki objektov.

GIS – Geographic Information System – sistem za zajemanje, shranjevanje, obdelavo, analizo, urejanje in predstavitev vseh tipov prostorskih podatkov.

WSGI – Web Server Gateway Interface – univerzalen vmesnik med spletnimi strežniki in aplikacijami napisanimi v programskem jeziku Python.

SEO – Search Engine Optimization – postopek izboljšanja vidljivosti spletišča ali spletne strani na iskalnikih prek naravnih oz. neplačanih iskalnih rezultatov.

ORM – Object Relational Mapping – tehnika, kjer podatkovne baze predstavimo kot objekt v razredih zaradi kompatibilnosti med različnimi sistemi.

YAML – YAML Ain't Markup Language (Yet Another Markup Language) – format s katerim določimo strukturo podatkov, ki so uporabniku berljivi. Temelji na koncptih programskih jezikov C, Perl in Python ter idejah iz XML in standarda RFC 2822.

IDE – Integrated Development Environment – je aplikacija, ki ponuja razvijalcem celovita orodja za razvijanje programske opreme.

POVEZETEK

V diplomskem delu bomo predstavili arhitekturo Model-View-Controller (v nadaljevanju MVC). Poskušali bomo predstaviti sam koncept tehnologije MVC, kako so posamezne komponente povezane med seboj ter kakšni so splošni principi v današnji uporabi.

Za primerjavo med posameznimi tehnologijami, bomo izdelali testni projekt, tako da bomo enako kodo replicirali na tri različne tehnologije in sicer Zend Framework, Django ter ASP.NET 3.0. Pri primerjavah se bomo skoncentrirali na najpomembnejše stvari, kot so kolikšen del samega projekta je avtomatiziran, koliko osnovnega znanja mora imeti posameznik, predno se odloči za katero izmed teh tehnologij, koliko fleksibilnosti nam ponuja, v kolikšni meri so določene tehnologije v osnovi že združene in v kolikšni meri nam določena tehnologija omogoča Rapid application development (v nadaljevanju RAD), ki je dandanes ključnega pomena, tako z ekonomskega vidika, kakor tudi z vidika hitrosti razvoja novih tehnologij. Skušali bomo predstaviti tudi skalabilnost MVC arhitekture, ki je prav tako pomemben faktor pri razvoju aplikacij. Med samim razvojem projekta bomo poskušali povleči določene vzporednice med okolji in ugotoviti v kolikšni meri sledijo vzorcu MVC. Izpostavili bomo prednosti določenega okolja MVC, na kaj moramo biti pozorni ter ostale lastnosti določenega okolja.

Ključne besede: arhitektura aplikacij, modularen način razvoja, koncept MVC, hitri razvoj aplikacij

ABSTRACT

In this diploma thesis we will introduce Model-View-Controller architecture (MVC). We will try to introduce concept of MVC technology, how are certain components related to each other and what general principles are used nowadays.

For comparison between different technologies, we will create test project on three different frameworks Zend Framework, Django and ASP.NET 3.0. In comparisons we will concentrate on automatization of certain steps in developing project, how much knowledge must developer have before he can start using one of the described frameworks, what flexibility each framework offers to developer, to what extend are certain technologies already bundled together and most important from time development perspective as well as economic, how is Rapid application development (RAD) presented in each. As scalability is major factor in application development, we will try to touch that topic as well. During the project development, we will seek for particular parallels between the frameworks and try to figure out if they are really following the MVC pattern. We will stress out advantages of each framework and environment in which they are developed, and on which things we need to pay attention when we decide for particular one.

Keywords: architecture of applications, modular application development, MVC concept, rapid application development

KAZALO

SEZNAM SLIK

1. UVOD

V poplavi takšnih in drugačnih spletnih strani, portalov in socialnih omrežij ter hitro nastajajočih tehnologij je težko definirati način oz. strukturo, ki bi pomenila dokončno rešitev vseh problemov. Pri razvoju spletnih aplikacij je zelo pomembno, da v začetni razvoj ne vlagamo preveč, vendar si skušamo omogočiti, da po potrebi aplikacijo priredimo bodisi uporabniku bodisi neki novi tehnologiji, ki bi morebiti izboljšala njeno kakovost.

V današnjem času je uporabnik seveda kralj in vsi se borijo za obstoj le te. Veliko podjetij da na letni ravni oz. nasploh veliko sredstev za uporabniške analize. Določene analize uporabniških izkušenj kažejo, da je za normalno brskanje po spletnih aplikacijah uporabnik zadovoljen tudi z zakasnitvami do 3 sekund. Vse, kar je okrog 3 sekund ali več, pa uporabniki že občutijo kot oviro pri brskanju. Uporabniške izkušnje se tudi spreminjajo in kar je danes zelo v uporabi, ni nujno, da bo primerno čez nekaj let.

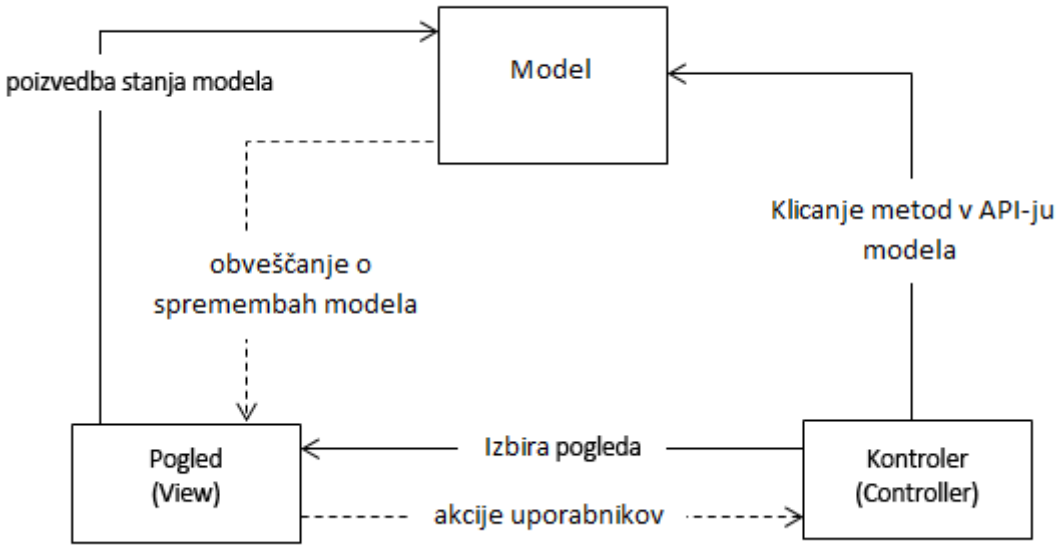
S tem, ko se povečuje računalniška pismenost in dostopnost do internetne povezave, je tudi vse več uporabnikov spletnih portalov in aplikacij. Nekatere standardne rešitve sistemov temu niso več kos in niso dovolj zmogljiva za vse večji naval uporabnikov.

Kakor hitro se veča masa uporabnikov, se s tem premosorazmerno veča tudi število novih tehnologij, ki preplavljajo trg. Nekatere tehnologije so dobrodošle in so inovativne, spet druge pa le izvedenke že obstoječe tehnologije, ki morebiti izboljšujejo obstoječo ali pa jo le prikažejo na drugačen način. Kljub vsemu temu je za arhitekta zelo pomembno, da je aplikacija, ki jo bo strukturiral, dovolj skalabilna, da ne podleže vsemu prej naštetemu.

Kateri model bo arhitekt aplikacij uporabil pri gradnji aplikacij, je v veliki meri odvisno od osebnih izkušenj. V diplomskem delu smo se osredotočili na en koncept in sicer MVC, ki je trenutno zelo popularen pri razvoju spletnih aplikacij. Sam koncept MVC je znan že nekaj časa v računalniškem svetu, vendar je v izraziti rabi šele nekaj let. Ravno modularnost, ki smo jo poprej izpostavljali, je pri MVC konceptu ključnega pomena. Pri implementaciji projektov imamo praktično proste roke kar se tiče rabe različnih tehnologij na različnih nivojih strukture. Naša naloga ni bila iskanje pomanjkljivosti in favoriziranje. Poskušali smo izpostaviti prednosti, ki nam jih tak način razvoja ponuja in prednosti samega okolja, v katerem smo razvijali. Diplomsko delo je bilo predvsem raziskovalne narave, zato smo ga razdelili na smiselne sklope za vsako od okolij, ki smo se ga odločili predstaviti.

2. KONCEPT MODEL-VIEW-CONTROLLER

Pri izdelavi samih projektov smo uporabili tehnologije, ki temeljijo na arhitekturi MVC. MVC pomeni, da sam projekt razbijemo na tri velike sklope oziroma plasti, kjer se na vsakem sklopu odvijajo procesi in operacije značilni zanj. Podrobneje o arhitekturi MVC na sliki 1. Tak koncept nam omogoča večjo fleksibilnost pri razvoju in implementaciji novih metod pri izvajanju programske kode.



Slika 1. Osnovni principi koncepta MVC.

2.1 Zgodovina

Sama ideja seže že v zgodnja osemdeseta leta, natančneje 1979. Prvič je ta koncept opisal Tryqve Reenskaug, ko je obiskal znanstvenike v laboratorijih podjetja Xerox. Poglavitni problem, ki je botroval nastanku MVC, je bil, kako ponuditi uporabnikom večjo kontrolo nad podatki z različnih perspektiv.

Prva implementacija se je zgodila leta 1983, ko so Jim Althoff in drugi napisali knjižnico za Smalltalk-80. Sama implementacija je bila opisana šele leta 1992 s strani Steva Burbecka.

Vzorec MVC je šele zadnja leta v razmahu in se uporablja praktično v vseh novejših spletnih ogrodjih in grafičnih vmesnikih. Za podrobnejšo razlago sledite spletni povezavi [1].

2.2 Razčlenitev posameznih komponent

2.2.1 Model

Model predstavlja konstrukcijo podatkov in logiko za dostopanje do teh podatkov in manipulacijo nad temi podatki. Z enim izrazom lahko rečemo, da je to API (v nadaljevanju Application programming interface), kjer so opisani vsi atributi, relacije in funkcije. En model oziroma razred v modelu, če le ta vsebuje več razredov, po navadi predstavlja fizično podatkovno tabelo v neki bazi oziroma zbirki podatkov.

Obstaja več načinov, kako lahko ustvarimo ogrodje modela, in sicer, tako da programer najprej definira razrede in znotraj teh razredov relacije in atribute ter preko njih ustvari podatkovno zbirko ali pa kot kreira podatkovno zbirko in pusti samemu procesu, da avtomatično generira razrede.

Samo strukturo modelov lahko razčlenimo še na nadaljnje tri nivoje. Na osnovni model z atributi, na razširjeni model tako imenovani Mapper in še tretji nivo, ki predstavlja poslovno logiko aplikacije. Taka razporeditev nam omogoča hitrejšo in bolj fleksibilno strukturo za prilagajanje raznim spremembam poslovnega modela.

Le kontroler lahko dostopa ali posodablja stanje v modelu, ta pa spremembe o stanju posreduje naprej do pogleda.

Pri realizaciji modelov se je v praksi prijela metodologija »Fat Models, Skinny Controllers«, kar pomeni, da je vsa logika shranjena v modelih, v kontrolerjih pa je le tisti del, ki nadzira samo pravilnost in konsistentnost podatkov, ki so poslani v procesiranje modelu.

2.2.2 Pogled

Pogled je zadolžen za prikaz stanja modela. Pomen podatkov, ki jih prikazuje pogled, lahko ločimo na različne skupine končnih uporabnikov in jim v skladu z njihovim pooblastilom bodisi omejimo bodisi omogočimo vpogled v podatke in uporabo le teh. Komunikacija od pogleda do modela poteka izključno preko kontrolerja in obratno.

Pogledi so rezervirani za front-end razvijalce, zato mora biti v pogledih čim manj programske kode. V kolikor pogosto uporabljamo en in isti segment kode skozi več različnih pogledov se lahko poslužimo uporabe helperjev. Helperji so funkcije oz sklopi funkcij, ki obdelajo določeno skupino podatkov in vrnejo rezultat.

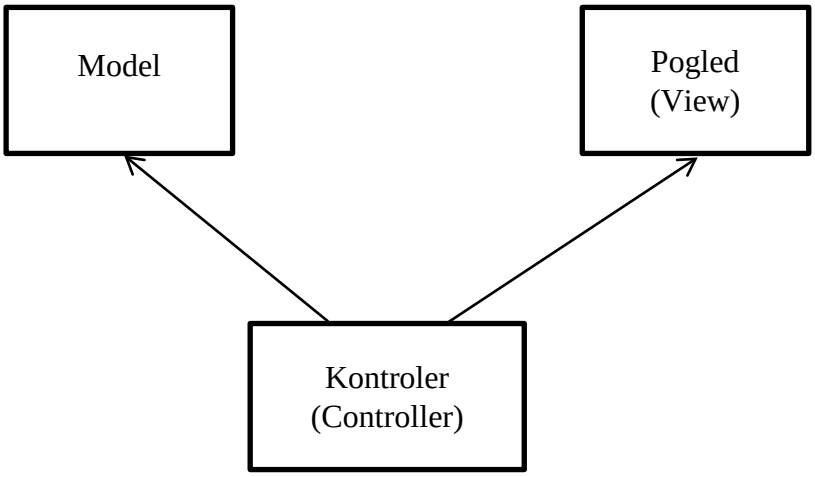
2.2.3 Kontroler

V kontrolerju se prav tako poslužujemo metode »Fat Models, Skinny Controller«, kar pomeni, da vso logiko prenesemo na model in s tem razbremenimo kontroler. Vse, kar dovolimo kontrolerju z vidika procesiranja podatkov, je preverjanje konsistentnosti podatkov (se pravi pravilna oblika oziroma format, podatkovni tip) ter preverjanje avtentikacije in avtorizacije uporabnikov. Na podlagi pridobljenih vhodnih parametrov in podatkov, ki jih pridobi od modela, se odloča, kateri pogled in podatki bodo prikazani.

Tako kot pri pogledih tudi tukaj poznamo helperje, ki nam pomagajo pri procesiranju podatkov in s tem razbijejo daljšo kodo na več manjših segmentov, ki so preglednejši in bolj učinkoviti pri posodabljanju. S tem obdržimo tudi konsistenco skozi vse kontrolerje, kadar pride do sprememb v sami kodi.

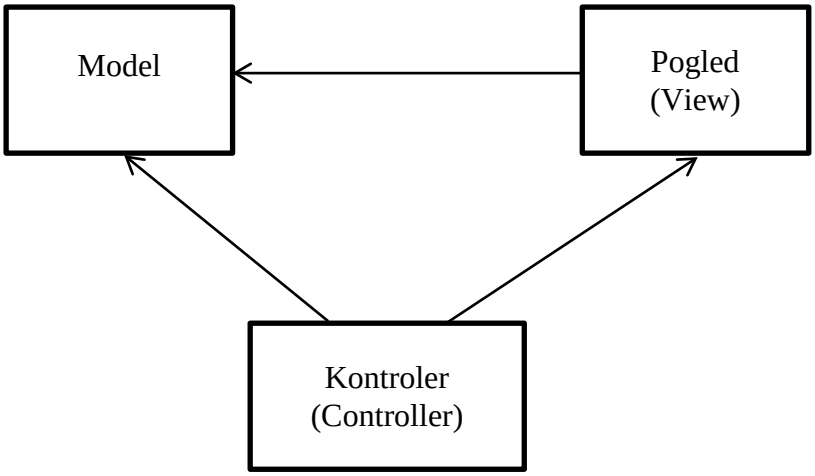
2.3 Delovanje

Obstajata dve različni interpretaciji vzorca MVC. V prvem primeru sta model in pogled popolnoma ločeni entiteti, ki med seboj ne komunicirata neposredno. Vse podatke med njima obdela kontroler, npr. slika 2.



Slika 2. Prva interpretacija vzorca MVC.

V drugem primeru pa lahko kontroler nadzira pogled in s tem pridobi potrebne podatke s strani modela, ti pa so potem poslani neposredno do pogleda brez interakcije kontrolerja, npr. slika 3. Drugače povedano, sam pogled nima funkcionalnosti v tolikšni meri, le toliko da lahko komunicira z modelom. Več o tem si lahko preberete na spletnem naslovu [2].



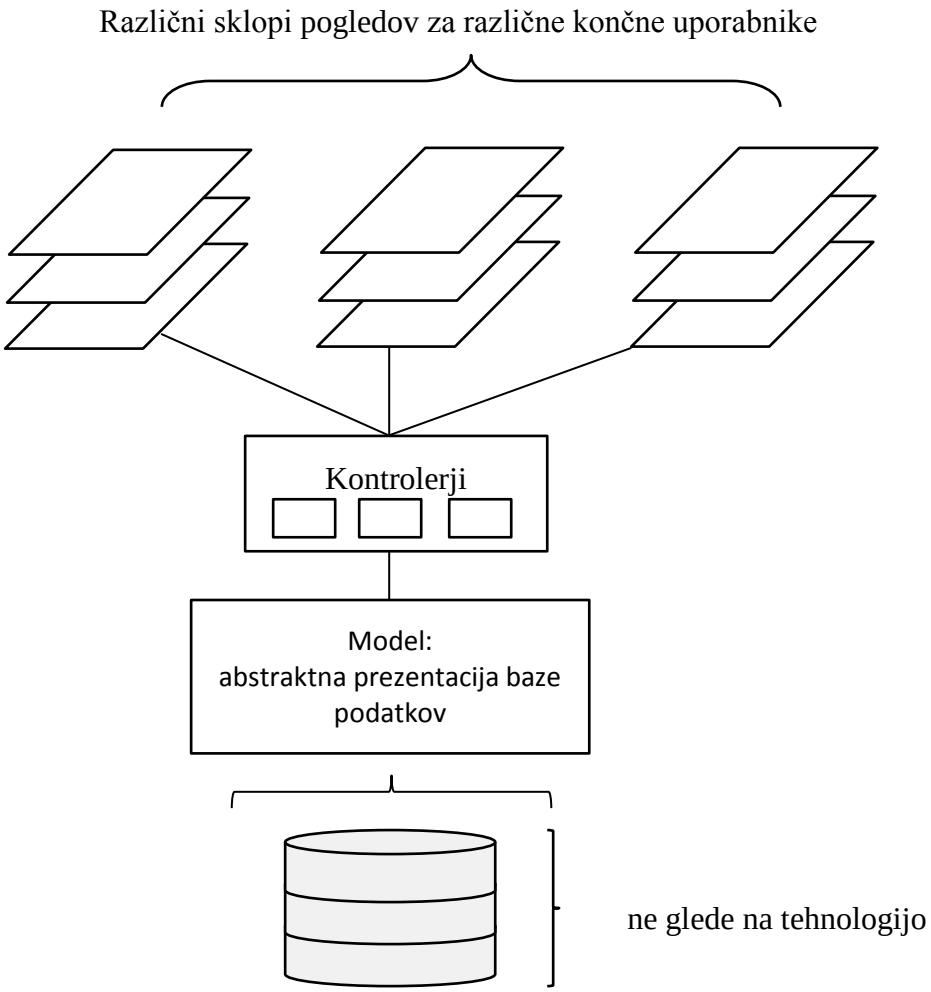
Slika 3. Druga interpretacija vzorca MVC.

Prenosljivost aplikacij na različne platforme in združevanje z drugimi tehnologijami je danes zelo pomembno pri razvoju aplikacij. V delovnem in poslovnem procesu podjetij je lahko veliko že obstoječih tehnologij, ki so ključne za samo delovanje. Od same strukture aplikacije je odvisno, kako hitro lahko razčlenimo in izločimo oziroma uporabimo že obstoječe sklope in koliko je potrebno, da se določen del aplikacije prilagodi potrebam specifičnega podjetja.

Razvijanje aplikacije v vzorcu MVC ima veliko prednosti kot tudi slabosti. Največja prednost je vsekakor razčlenitev aplikacije na več različnih sklopov. S takim pristopom lahko razvijalci, ki razvijajo aplikacijo na več različnih področjih (front-end – tisto, kar uporabnik dejansko vidi in s čem uporabnik interaktivno upravlja aplikacijo, back-end – del, ki skrbi za procesno logiko aplikacije ter sama podatkovna shema, ki hrani podatke in relacije med podatki). Zaradi ločene strukture nam vzorec MVC omogoča tudi večjo skalabilnost (razvijanje večjih aplikacij) ter lažje in učinkovitejše spremembe in vzdrževanja (ostro ločena opravila).

Poleg strukture nam vzorec MVC ponuja tudi več različnih tako imenovanih načinov načrtovanja aplikacije. V ta sklop spadajo Observer [3], Composite [4], Strategy [5] in Factory [6] način načrtovanja. Podrobnejši vpogled v različne vzorce načrtovanja, ustvarjanja strukture in odziva aplikacije je v knjigi [7].

Kot največjo slabost takemu vzorcu lahko štejemo preobsežnost in veliko obdelavo podatkov, kar za seboj povleče večje in zmogljivejše vire, npr. slika 4. Velika večina aplikacij potrebuje namenski strežnik za optimalno delovanje, kjer se lahko določene stvari shranjujejo v predpomnilniku in izvajajo hitreje. Ravno zaradi tega moramo razmotriti smiselnost uporabe vzorca MVC.



Slika 4. Groba shema arhitekture MVC.

3. ORODJA

Pri izdelavi projektov smo uporabili brezplačne in prosto dostopne rešitve, tehnologije in programsko opremo. Vsekakor so brezplačne verzije okrnjene in ne ponujajo vsega, kar bi lahko izkoristili, vendar za naše projekte ravno dovolj, da lahko primerjamo različne implementacije in določimo prednosti ter slabosti enega ali drugega.

Vsa orodja so bila nameščena na 64-bitnem operacijskem sistemu Windows 7. Program NetBeans lahko uporabljamo tudi na ostalih sistemih kot so operacijski sistemi UNIX/Linux, prav tako pa najdemo različico WAMP za Linux operacijski sistem in sicer LAMP. Za razvoj ASP.NET MVC na sistemih Linux je treba poiskati primerne rešitve (na primer MonoDevelop). Pri inštalaciji Pythona in Djanga je treba paziti na kompatibilnost med enim in drugim, v nasprotnem primeru je treba verzijo Pythona bodisi znižati oziroma posodobiti.

3.1 Programska oprema

3.1.1 NetBeans

NetBeans (takrat znanega kot Xelfi) so leta 1996 začeli razvijati študenti na Univerzi Charles v Pragi. Leta 1997 je Roman Stanek zasnoval podjetje, ki se je ukvarjalo s komercialnim razvojem, vse dokler ni leta 1999 ameriško podjetje Sun Microsystems odkupilo pravice.

NetBeans je odprtokodna programska oprema (pod licencama CDDL in GPLv2), v kateri lahko razvijamo aplikacije na različnih platformah na primer Java, JavaScript, PHP, Python (verzija 7+ ne podpira Pythona, razvija pa se dodatek neodvisno od razvoja NetBeansa), Groovy, C, C++, Scala, Clojure itd. Spisan je v programskem jeziku Java in ga lahko poganjamo na operacijskih sistemih, kot so Windows, Mac OS, Linux, Solaris in ostalih, ki so kompatibilni z JVM. Platforma NetBeansa je zgrajena tako, da omogoča razvijanje aplikacij iz različnih modulov. NetBeans je razširljiv preko različnih dodatkov, ki so razviti s strani skupnosti uporabnikov.

NetBeans lahko dobimo v več različnih paketih, ki jim pravimo Bundle. Vsak paket vsebuje tista orodja, ki so specifična in potrebna za razvoj v določenem programskem jeziku. Najdemo pa tudi tak paket, kjer lahko inštaliramo celoten paket za programske jezike, ki jih NetBeans podpira.

Kakor vsi standardni vmesniki za razvoj programske kode tudi NetBeans ponuja ključna orodja, kot so sistem za nadzor različic, orodja za razhroščevanje, namigi pri uporabi funkcij, opozorila pri strukturiranju kode, iskanje funkcij in razredov znotraj projekta itd.

Za razvoj aplikacij v ogrodju Zend u moramo v nastavitvah, pod sekcijo PHP > Zend nastaviti pot do datoteke zf.bat. Ukaze uporabljamo tako, da s klikom na desni miškin gumb izberemo Zend in nato Run command. Če hočemo ukaze uporabljati v konzoli, moramo najprej v operacijskem sistemu Windows pod Environment variables nastaviti pot do skripte. S pomočjo skripte zf.bat lahko preko konzole kreiramo projekte in ostale komponente. Nekaj osnovnih ukazov:

- zf show version
- zf –help
- zf create project
- zf create controller –name [ime_controller-ja]

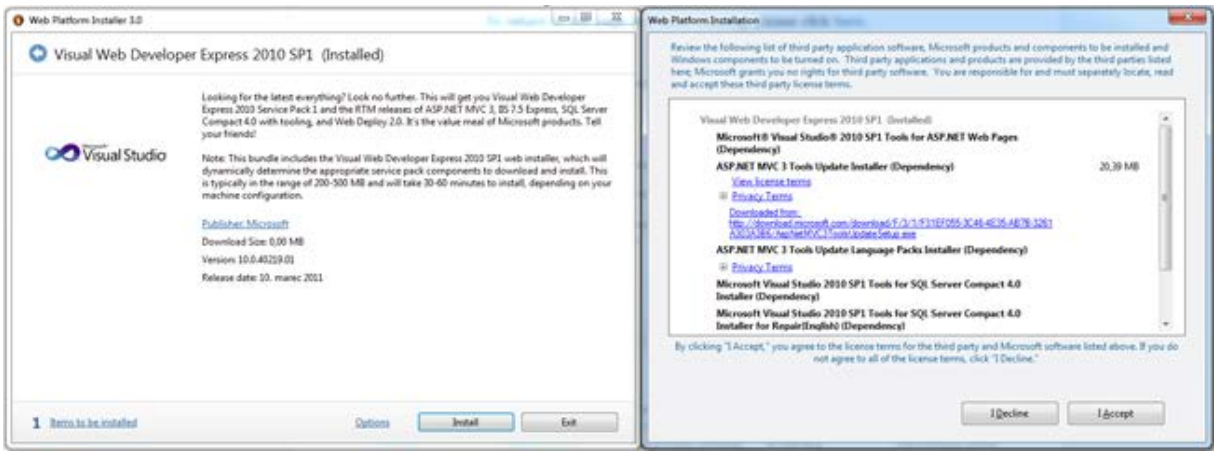
Za razvoj aplikacij v okolju Django moramo namestiti potrebne dodatke, ki nam omogočajo prepoznavo in ustvarjanje Pythonovih datotek. Sam dodatek je še zmeraj v razvojni fazi, zato ni podpore za Djangove funkcije in razrede, prav tako pa ne moremo kreirati projekta preko Djangovih ukazov, ki jih sicer uporabljamo v konzoli.

3.1.2 Visual Web Developer 2010 Express

Visual Web Developer 2010 Express je brezplačen skupek orodij, npr. slika 5, s katerim lahko začnemo razvijati aplikacije v ASP.NET MVC okolju ali drugih kot je na primer SilverLight. V paketu dobimo inštalacijo spletnega strežnika, orodja za delo s podatkovnimi bazami in orodja za razvoj.

Poleg orodij za razvoj vsebuje tudi orodja za nadzor nad pravilno uporabo standardov HTML in CSS, če razvijamo aplikacijo za različne brskalnike in naprave. Vsebuje tudi podporo za jezik JavaScript pri katerem nam ponuja namige, kot so vgrajene funkcije in atributi, če je spremenljivka objekt. Vmesnik nam ob kreiranju projekta ponudi dve opciji in sicer razvoj aplikacij ASP.NET MVC v programskem jeziku Visual Basic ali C#. Namesti nam knjižnice jQuery, ustvari ogrodje in potrebne datoteke za začetek dela s projektom. Preko Visual Web Developerja lahko kreiramo podatkovne baze in jih ravno tako urejamo ter ustvarjamo poizvedbe bodisi preko vnosnega polja ali pa vizualno preko diagrama.

Za lažji razvoj nam ponuja različne postavitve oken, ki si ga lahko prilagodimo po lastnih potrebah in željah. Za pravilno delovanje kode nam ponuja pregledovanje vrednosti posameznih spremenljivk, pregledovanje kode korak za korakom, postavimo lahko točke, kjer naj se izvajanje kode prekine, shranimo lahko procedure napisane v jeziku SQL itd. Preko vmesnika lahko tudi neposredno postavimo aplikacijo na spletne strežnike.



Slika 5. Vmesnik za namestitev okolja Visual Web Developer.

3.2 Tehnologije

3.2.1 WAMP

WAMP je akronim za skupek med seboj neodvisnih tehnologij, ki tečejo oziroma so namenjene operacijskemu sistemu Windows. Paket vsebuje strežnik Apache, sistem za urejanje podatkovnih zbirk MySQL in enega od sledečih programskih jezikov PHP, Perl ali Python. Poleg naštetih vsebuje tudi vmesnik phpMyAdmin, preko katerega lahko kreiramo in urejamo podatkovne zbirke. Vse to nam omogoča, da v paketu dobimo sistem, ki zna upravljati z dinamičnimi spletnimi stranmi tako imenovani spletni strežnik.

3.2.2 Internet Information Services

Internet Inforamtion Services znan tudi kot IIS, je spletni strežnik in sklop modulov za poganjanje aplikacij napisanih v Microsoftovih programskih jezikih. Med strežniškimi sistemi, ki se trenutno uporabljajo, zaseda tretje mesto z 14% deležem in 12% deležem vseh zahtev, ki se zgodijo na spletu. IIS 7.5 podpira HTTP, HTTPS, FTP, FTPS, SMTP in NNTP. Dobimo ga ob inštalaciji vseh verzij Windows Server in nekaterih verzijah Windows XP, Windows Vista ter Windows 7. Ob inštalaciji operacijskega sistema Windows IIS ni samodejno vključen, zato ga je treba predhodno vklopiti.

Predvsem v zgodnjih verzijah je bil IIS izpostavljen mnogim spletnim napadom. Najbolj znan je bil napad s črvom Code Red, vendar so te pomanjkljivost odpravili v verzijah 6.0 in 7.0. V verziji 6.0 so onemogočili samodejno zaganjanje spletnih servisov in skrbnikom strežnika omogočili, da določijo, kateri imajo dovoljenja. V verziji 7.0 so popolnoma spremenili koncept IIS in ga napisali na novo. V strukturo so vključili modularen vzorec, ki je povečal zmogljivost in zmanjšal izpostavljenost napadom. Hierarhičen pristop je omogočil poganjanje bolj enostavnih spletnih strani, nov način urejanja aplikacij Windows Forms, nov način urejanja preko ukazne vrstice in večjo podporo ogrodju .NET. IIS 7.0 nima omejenega števila vseh povezav, kot jih je imel na operacijskem sistemu Windows XP, vendar ima omejitev sočasnih povezav. Bolj podroben opis in vsebino modulov, ki jih verzije od 6.0 naprej vsebujejo, si lahko ogledate na [8].

3.2.3 Zend Framework

Zend Framework je objektno orientirano okolje implementirano v programskem jeziku PHP 5. Spada pod odprtokodne rešitve. Ustanovitelja sta Andi Gutmans in Zeev Suraski, ki sta ob enem tudi soustvarjalca programskega jezika PHP. Zend Framework podpirajo največja informacijska podjetja, kot so IBM, Google, Microsoft, Adobe Systems in Strikelron.

Za testiranje aplikacij v okolju Zend potrebujemo PHPUnit verzijo 3.0 ali višjo. V sklop okolja Zend so vključene sledeče komponente in značilnosti:

- vse komponente so objektno orientirane in se izvajajo v načinu E_STRICT,
- ohlapno implementirana arhitektura (minimalna soodvisnost komponent),
- razširljiva implementacija MVC-a za podporo predlogam PHP,
- podpora različnih podatkovnih sistemov kot so MariaDB, MySQL, Oracle, IBM DB2, Microsoft SQL Server, PostgreSQL, SQLite in Informix Dynamic Server,
- izdelava in pošiljanje elektronskih pošt preko mbox, Maildir, POP3 in IMAP4,
- fleksibilen pomnilniški podsistem, ki podpira različne tipe.

Zend Technologies so za poganjanje aplikacij napisanih v okolju Zend razvili Zend Server, ki naj bi po njihovih besedah bil optimiziran posebej za aplikacije Zend in z izboljšanim izvajanjem kode PHP. Zend Server ponuja tudi hranjenje operacijske kode v pomnilniku, ki pripomore k hitrejšemu izvajanju, orodja za nadzor nad izvajanjem aplikacij in za diagnosticiranje samega okolja. Zend Server je kompatibilen tudi z orodji Xdebug. Za razvoj aplikacij pa so razvili Zend Studio, ki vsebuje dodatke napisane za podporo Zend okolju. Obe tehnologiji sta plačljivi, vendar obstaja tudi brezplačna okrnjena različica Zend Serverja (Community Edition).

Skupaj s partnerji je Zend Technologies leta 2009 začel razvijati okolje za poganjanje aplikacij v oblaku. Trenutna verzija je dovolj kvalitetna, da so jo dali v produkcijo pod imenom Simple Cloud API.

Trenutna verzija Zend okolja je 2.0 in prinaša marsikatere izboljšave, kot so na novo napisani razredi, izboljšani paket funkcij za testiranje, zmanjšanje singleton razredov ter največja prednost, uporaba namespaceov, ki jih v splošno rabo uvaja PHP 5.3. Več o Zend okolju lahko najdete na [9].

3.2.4 ASP.NET MVC, pogon Razor View in SQL Compact Edition 4.0

ASP.NET je spletno okolje za gradnjo aplikacij, ki ga je razvilo podjetje Microsoft. Namenjeno je razvijanju dinamičnih spletnih strani. Razvila sta ga Mark Anders in Scott Guthrie, ki sta leta 1997 po končanem študiju dobila nalogo, da začrtata nov način izvajanja

dinamičnih strani na Microsoftovih strežnikih IIS z ločenim nivojem med prezentacijo in vsebino. Najprej sta poskušala razviti platformo v programskem okolju Java, vendar sta se kasneje odločila, da bosta razvila popolnoma novo platformo zasnovano na programskem jeziku C#. Na konferenci v Phoenixu leta 2000 je bila predstavljena prva javna različica jezika ASP+. Nekaj mesecev kasneje so dodali še podporo za nove jezike, kot so Visual Basic .NET, Python in Perl. Sčasoma so jezik preimenovali iz ASP+ v ASP.NET in tako leta 2002 izdali prvo verzijo pod imenom ASP.NET 1.0. Zadnje čase so se pri razvoju fokusirali predvsem na podporo mobilnim napravam.

Internetne strani ASP.NET so uradno znane pod imenom Web Forms. Web Forms so vsebovane v datotekah s končnico .aspx. Večinoma vsebujejo statično vsebino (HTML koda), vsebujejo pa tudi oznake za kontrole na strani strežnika kakor tudi za uporabniške kontrole. Kako se bo stran izvedla, definiramo v direktivah, ki se običajno nahajajo na začetku datoteke.

Na začetku okolje ASP.NET MVC ni imelo razvitega načina za implementacijo predlog. Programerji so razvili lastne razrede za implementacijo skupnih komponent v spletne strani in se temu izognili vključevanju komponent v vsako stran posebej. Kasneje so razvili master strani, ki so lahko bile vgnezdene in vsaka stran je lahko imela več master strani (več na [10]). Glede na to da je view pogon modularen in razširljiv z vtičniki, ASP.NET MVC omogoča implementacijo različnih sintaktičnih načinov pisanja predlog. Med najbolj znanimi sta Spark in NHaml. Med njimi je tudi prvotno mišljeni način ASP.NET Web Forms. Kljub temu so razvijalci razvili pogon Razor view, npr. slika 6. Pri razvoju novega načina so skušali doseči kompaktnost, izraznost in tekoče pisanje kode, ki se ne vsiljuje v HTML sintakso. Poleg tega so skušali ohraniti enostavnost, omogočiti pisanje sintakse v kateremkoli urejevalniku, pri pisanju kode ponuditi podporo za namige in možnost testiranja kode.

```
<ul id='list'>
      <% foreach(var n in list) { %>
          <li><%=n.Name%> ($<%=n.Price%>)</li>
      <% } %>
</ul>
```

```
<ul id='list'>
      @foreach(var n in list) {
          <li>@n.Name ($@n.Price)</li>
      }
</ul>
```

Slika 6. Razlika med navadnim načinom ASP.NET Web Forms in načinom Razor.

SQL CE je brezplačen in vdelan podatkovni sistem, ki nam omogoča enostavno hrambo podatkov. Za uporabo ni potrebna predhodna inštalacija baze ali kakršnegakoli strežnika. Za delo nad bazami ne potrebujemo računa z administratorskimi pravicami. Aplikacijo lahko prenašamo iz sistema na sistem, ne da bi nam bilo potrebno skrbeti za pravice nad izvajanjem aplikacije. Naloži se skupaj z aplikacijo v pomnilnik ob prvem dostopu do baze. Shranjena je kot datoteka in se ponavadi nahaja v direktoriju \App_Data. Deluje z vsemi obstoječimi .NET podatkovnimi API-ji in podpira tudi SQL Server način pisanja poizvedb. Uporabljamo lahko ADO.NET, kakor tudi višje nivojske tehnologije Entity Framework ali NHibernate. Uporablja se v razvojnih, testnih ali manjših produkcijskih okoljih. Za razliko od starejših različic SQL CE 4.0 podpira tudi večnitno poganjanje aplikacij. Je prenosljiva na druge platforme npr. SQL Server Express, SQL Server ali SQL Azure. Ne podpira stored procedur. Ima tudi fizične omejitve, kot so velikost posamezne baze ter število povezav na bazo.

3.2.5 Django

Django je odprtokodno okolje napisano v programskem jeziku Python. Sprva je bilo razvito za upravljanje večih novičarskih spletnih strani podjetja The World Company in bilo kot tako leta 2005 izdano pod licenco BSD. Ime je dobilo po francoskem kitaristu romskih korenin Djangu Reinhardtu. Leta 2008 je bilo oznanjeno, da bo Django pod okrilje prevzela neodvisna fundacija Django Software Foundation, ki bo skrbela za razvoj okolja.

Poglavitni cilj, ki so si ga zastavili pri razvoju Djanga, je olajšanje razvoja kompleksnih in podatkovnih spletnih aplikacij. Django poudarja predvsem večkratno rabo in način rabe komponent kot vtičnike, pospešen razvoj in princip neponavljanja. Python se uporablja povsod tudi v nastavitvah, datotekah, kakor tudi v model-ih. Django nam opcijsko ponuja tudi administrativni vmesni CRUD, ki ga lahko dinamično ustvarimo.

Pri inštalaciji Django okolja dobimo še strežnik za razvijanje ali testiranje, sistem za serializacijo in potrditev podatkov, ki jih skušamo shraniti v podatkovno bazo, sistem za hranjenje podatkov v pomnilniku, podporo za middleware razrede, notranji sistem za razpošiljanje, ki omogoča komponentam medsebojno komuniciranje, internacionalizacijo komponent, ki omogoča prevajanje v druge jezike, serializacijo, ki omogoča izdelavo in branje modelov strukturiranih v načinu XML ali JSON, sistem za razširitev predlog ter sistem za testiranje zgrajen v okolju Python. Ob inštalaciji lahko osnovni paket razširimo z različnimi dodatki, kot so sistem za potrjevanje, orodja za generiranje novic RSS in Atom, fleksibilen sistem za pisanje komentarjev, orodja za podporo Google Sitemaps, orodja za preprečevanje zahtev preko tujih spletnih strani, okolje na, katerem lahko ob eni sami inštalaciji poganjamo več različnih spletnih aplikacij ter podporo za aplikacije GIS.

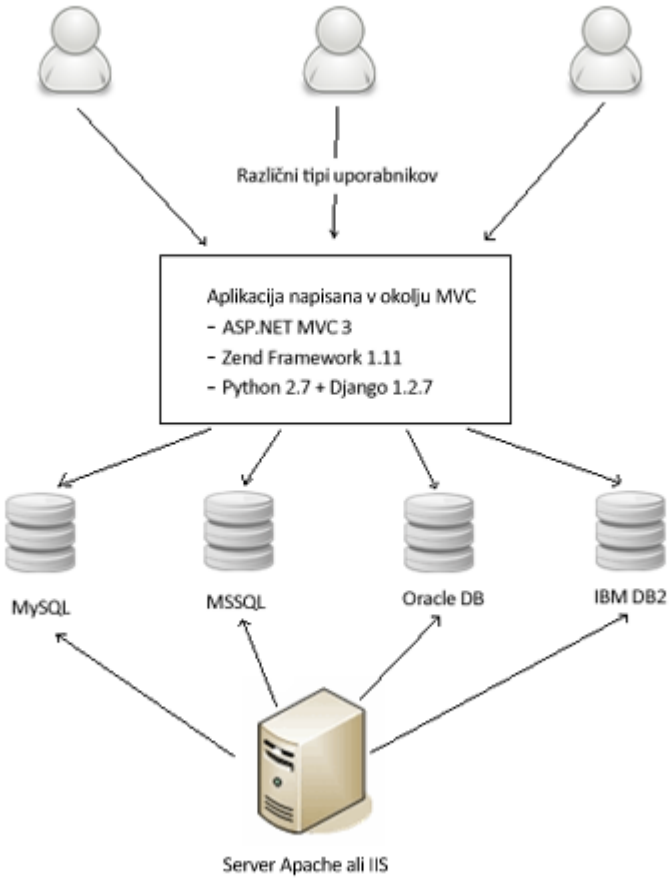
Django lahko poganjamo na strežnikih Apache, če imamo nameščene module mod_wsgi in mod_python, kakor tudi ostalih strežnikih, ki podpirajo WSGI vmesnik. Več o okolju Django si lahko preberete na [11].

4. IZDELAVA PROJEKTA

4.1 Opis naloge

Kot poglaviten del naloge smo si zastavili primerjavo med različnimi okolji (osnovni koncept naše aplikacije, npr. slika 7). V primerjavo smo vključili vse podrobnosti, ki so pomembne pri obravnavi nekega programskega okolja. Ključne so vsekakor prenosljivost, implementacija (v primerih, ko postavljamo na novo ali ko posodabljamo že obstoječo infrastrukturo in v kolikšni meri takšna posodobitev nekoga obremeni ekonomsko in funkcionalno), skalabilnost, modularnost, v kolikšni meri lahko v osnovnih nastavitvah dobimo vse potrebno in koliko tega je treba naknadno nadgraditi bodisi z uradnimi dodatki bodisi z dodatki drugih proizvajalcev.

V nalogi se bomo lotili tudi primerjave samega razvoja aplikacije, koliko je potrebno, da se nek programer vključi v delovni proces razvoja, primerjavo med samo strukturo projekta, določene specifike enega in drugega okolja, koliko so določeni procesi avtomatizirani, kakšna orodja ima razvijalec na voljo itd.



Slika 7. Prikaz sheme, ki nam služi pri razvoju aplikacije v različnih okoljih.

4.2 Potek izdelave projekta v vseh treh programskih jezikih

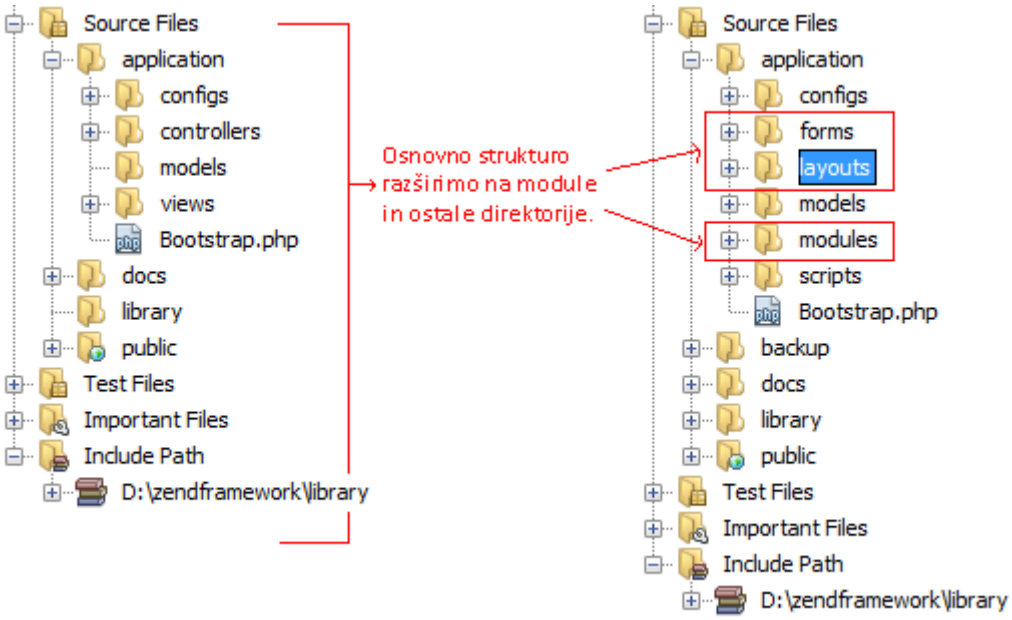
Aplikacija, ki smo jo razvili, naj bi predstavljala osnutek foruma. Na prvi strani vsebuje imena tem, kot podrobnosti pa smo dodali število sporočil, ki jih tema vsebuje, datum nastanka in avtorja teme. Ob kliku na temo skočimo na seznam sporočil, ki jih tema vsebuje. Tu ravno tako ob vsakem sporočilu, dobimo še podatke o času nastanka sporočila, kdo je avtor ter vsebino sporočila. Samega foruma nismo nadgrajevali oz. smo preskočili implementacijo avtorizacije uporabnika kakor tudi registracijo uporabnikov. Ob obeh pogledih smo dodali tudi gumbe, ki nas preusmerijo na obrazce (za vsak obrazec posamezen pogled) za dodajanje tem in sporočil. Med temami in sporočili smo v nekaterih okoljih implementirali relacijo »Many-to-many«, npr. slika 8, kjer je to okolje dopuščalo, v ostalih pa smo se morali zadovoljiti z relacijo »One-to-many« (v nadaljevanju O2M), npr. slika 9. »Many-to-many« (v nadaljevanju M2M) pomeni da npr. imamo dve entiteti, sta med seboj povezani, tako da imamo recimo v podatkovni tabeli A vrstico, ki je povezana z večimi vrsticami v tabeli B in obratno. Pri taki realizaciji nimamo tretje tabele, preko katere shranjujemo relacije oz. relacijske tabele. V rešitvi »One-to-many« pa imamo relacijsko tabelo, ki hrani vrednosti iz tabele A kakor tudi iz tabele B.

Slika 8. Primer relacije »Many-to-many« v tabeli.

Slika 9. Primer relacije »One-to-many« v tabeli.

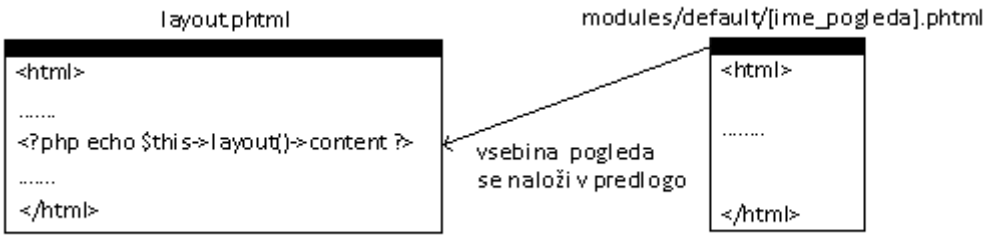
4.2.1 Programsko okolje Zend

Sprva smo se lotili izdelave projekta v Zend Framework-u. V NetBeansu (potem, ko smo v nastavitvah pravilno nastavili pot do datoteke zf.bat, ki poskrbi da se kreira osnovna struktura projekta ter pomaga izvajati ukaze Zend znotraj NetBeansa) ustvarimo osnovni projekt, kjer nastavimo ime projekta, verzijo PHP-ja, v katerem bomo programirali aplikacijo, nastavimo domeno, preko katere bomo testirali projekt ter izberemo Zend kot obliko strukture. Strukturo projekta smo razširili po potrebi. Dodali smo tri nove direktorije in sicer forms, layouts in modules, npr. slika 10.



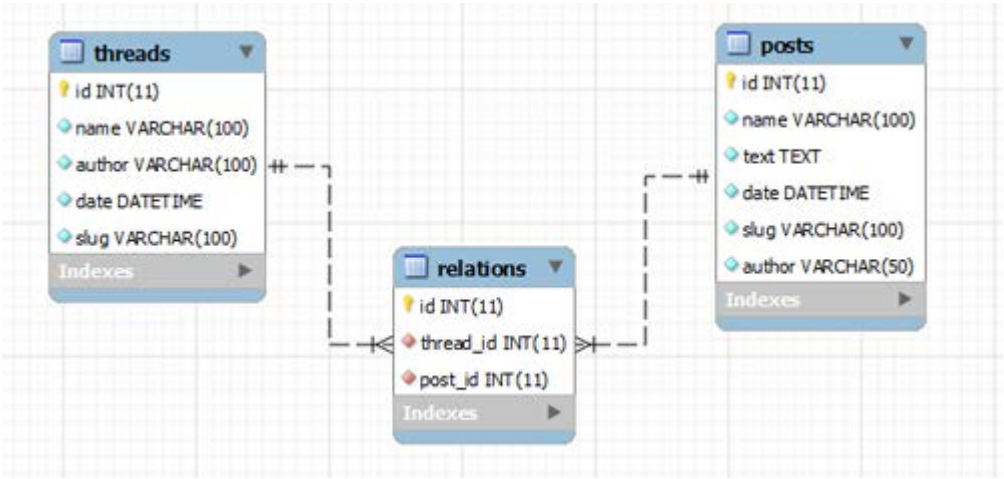
Slika 10. Struktura projekta v Zend Frameworku.

V direktoriju forms smo kreirali razrede, v katerih smo definirali obrazce za vnos tem in sporočil. V razredih za obrazce lahko definiramo različne parametre in atribute, kot so na primer preverjanje pravilnosti podatkov, omejitve vnosnih polj, preoblikovanje izgleda polj s t.i. decoratorji. V direktoriju layouts smo postavili pogled za predlogo, ki je skupna vsem pogledom znotraj tega modula. Če na primer definiramo modul default, bodo vsi pogledi v direktoriju modules > default podedovali nastavitve iz pogleda layouts > layout.phtml, npr. slika 11. S predlogami smo si olajšali nastavljanje skupnih atributov znotraj pogledov npr. doctype, povezave do datotek JavaScript, povezave do stilov itd.



Slika 11. Prikaz predloge in pogleda v Zend Fremworku.

Po tem, ko smo določili našo strukturo znotraj projekta, smo se osredotočili na izgradnjo podatkovne baze v okolju MySQL. Tu smo naleteli na prvo slabost Zend Frameworka. Podatkovno bazo je treba kreirati ročno. V našem primeru smo jo kreirali s pomočjo vmesnika phpMyAdmin. Lahko pa si pomagamo tudi z brezplačnim orodjem MySQL Workbench, v katerem kreiramo diagram in preko katerega kasneje kreiramo podatkovno bazo, npr. slika 12.



Slika 12. Primer načrtovanja podatkovnih baz v orodju MySQL Workbench.

Najpomembnejši korak pri kreiranju projekta v Zend Frameworku so nastavitve. Ob pravilni nastavitvi nam ob uporabi določenih razredov znotraj projekta ni potrebno na začetku razreda vključevati uporabljenih razredov, ker so vidni znotraj celotnega projetka. Nastavitve se nahajajo v direktoriju application > configs v datoteki application.ini. V application.ini nastavljamo takorekoč vse od sistemskih nastavitev (ločene nastavitve za različna okolja, prikazovanje napak, osnovno domeno, časovni pas, dostopni podatki do podatkovnih baz) do strukturnih nastavitev (pot do knjižnic, pot do modulov, pot do predlog, predpona za lastne razrede).

V datoteki index.php, ki se nahaja v direktoriju public, smo nastavili pot do direktorija, kjer se nahaja celotna aplikacija (v našem primeru direktorij application). Okolje smo nastavili na development, kar pomeni razvojno okolje. Poleg slednjih dveh smo nastavili še pot do knjižnic.

Predno aplikacijo dokončno poženemo, naložimo še nastavitve iz datoteke application.ini. Nekatere atribute v application.ini, ob predpostavki, da uporabimo rezervirana imena, aplikacija samodejno naloži. Katera so ta imena, lahko preverimo v knjižnici Zend v razredu Application.php. Vse ostale nastavitve naložimo v datoteki Bootstrap.php. Pred imena funkcij, ki se avtomatično izvedejo znotraj datoteke Bootstrap.php, moramo dodati predpono `_init[ime_funkcije]`, saj tako povemo aplikaciji, da inicializira vsebino funkcije. Za imena funkcij in razredov se uporablja oblika CamelCase (prva črka je velika ali mala začetnica, besede se ločujejo z velikimi začetnicami in brez presledkov ali vezajev).

Na naslednji strani imamo primer iz vseh treh datotek (application.ini, index.php in Bootstrap.php). V .htaccess nastavitvah moramo preveriti, če je `SetEnv APPLICATION_ENV` nastavljen na development.

Primer nastavitev, ki smo jih uporabili:

application.ini

```
[production]
phpSettings.display_startup_errors = 0
phpSettings.display_errors = 0
includePaths.library = APPLICATION_PATH "/../library"
bootstrap.path = APPLICATION_PATH "/Bootstrap.php"
bootstrap.class = "Bootstrap"
appnamespace = "Application"
resources.frontController.moduleDirectory = APPLICATION_PATH "/modules"
resources.frontController.params.displayExceptions = 0
autoloaderNamespaces.my = "My_"
resources.layout.layout = "layout"
resources.layout.layoutPath = APPLICATION_PATH "/layouts"
resources.view[] =
resources.db.adapter                = "PDO_MYSQL"
resources.db.params.host            = "localhost"
resources.db.params.dbname          = "dipl_forum"
resources.db.params.username        = "root"
resources.db.params.password        = ""
resources.db.isDefaultTableAdapter  = true

[staging : production]

[testing : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1

[development : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1
resources.frontController.params.displayExceptions = 1
resources.db.adapter                = "PDO_MYSQL"
resources.db.params.host            = "localhost"
resources.db.params.dbname          = "dipl_forum"
resources.db.params.username        = "root"
resources.db.params.password        = ""
resources.db.isDefaultTableAdapter  = true
```

index.php

```
<?php

// Define path to application directory
defined('APPLICATION_PATH')
    || define('APPLICATION_PATH', realpath(dirname(__FILE__) .
'/../application'));

// Define application environment
defined('APPLICATION_ENV')
    || define('APPLICATION_ENV', (getenv('APPLICATION_ENV') ?
getenv('APPLICATION_ENV') : 'production'));

// Ensure library/ is on include_path
set_include_path(implode(PATH_SEPARATOR, array(
    realpath(APPLICATION_PATH . '/../library'),
    get_include_path(),
)));

/** Zend_Application */
require_once 'Zend/Application.php';
// Create application, bootstrap, and run
$application = new Zend_Application(
    APPLICATION_ENV,
    APPLICATION_PATH . '/configs/application.ini'
);
$application->bootstrap()
            ->run();
```

Bootstrap.php

```
<?php

class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
    protected function _initAppAutoload() {
        $moduleLoader = new Zend_Application_Module_Autoloader(array(
           'namespace' => '',
           'basePath'   => APPLICATION_PATH
        ));
        return $moduleLoader;
    }

    protected function _initDb() {
        $config = new Zend_Config_Ini(APPLICATION_PATH.
                '/configs/application.ini', 'production');
        $db = Zend_Db::factory('PDO_MYSQL', $config->resources->db->params);
        Zend_Db_Table_Abstract::setDefaultAdapter($db);
        Zend_Registry::set('db' ,$db);
    }

    protected function _initDoctype() {
        $this->bootstrap('view');
        $view = $this->getResource('view');
        $view->doctype('XHTML1_STRICT');
        return $view;
    }
}
```

Na vrsti je kreiranje razredov, ki predstavljajo tabele v podatkovni bazi. Razrede smo kreirali v direktoriju models in jim dodelili imena enaka tabelam za lažjo prepoznavnost (imena razredov niso striktno povezana z imeni tabel in so lahko poljubna). Paziti pa moramo na celotno ime znotraj razreda. Sestavljeno mora biti iz direktorijev, v katerem se nahaja končni razred npr. `Application_Model_[ime_razreda]`, kar pomeni, da se razred nahaja v direktoriju application > models. Ime tabele, ki jo predstavlja razred, definiramo preko atributa `$_name`. V ostalih parametrih definiramo še ključ primary z atributom `$_primary`, ter tabele, ki so odvisne od trenutne – `$_dependentTables` (če imamo v podatkovni bazi pri relacijah uporabljen način CASCADE nam ni potrebno definirati atributa `$_dependentTables`). Poleg teh parametrov imamo na voljo še parameter `$_referenceMap`, s katerim določimo povezave do drugih tabel t.i. foreign key. `$_referenceMap` smo definirali, da smo lahko uporabili princip M2M. Zend Framework nima pravega načina M2M in ga simulira preko navadne relacije O2M. V razredih, kjer so definirani modeli, imamo na voljo že nekaj osnovnih funkcij CRUD. Za naše potrebe smo dodali še nekaj prirejenih funkcij za točno določeno iskanje in shranjevanje. Za relacije M2M se uporablja vgrajena funkcija `findManyToManyRowset` na naslednji način:

```
$select = $this->find($select[0]['id']);

$threadSlug = $select->current();

$result = $threadSlug->findManyToManyRowset('Application_Model_Posts',
             'Application_Model_Relations');
$result = $result->toArray();
```

Primeri naših model razredov:

Threads.php

```
class Application_Model_Threads extends Zend_Db_Table_Abstract
{
    protected $_name = 'threads';
    protected $_primary = 'id';

    protected $_dependentTables = array('Application_Model_Relations');

    function __construct() {
        parent::__construct();
    }

    //seznam nekaterih prirejenih funkcij
    public function getOneBySlug($slug) {
        $select = $this->select()->where('slug=?', $slug);
        return $this->_fetch($select);
    }

    …

}
```

Posts.php

```
class Application_Model_Posts extends Zend_Db_Table_Abstract
{
    protected $_name = 'posts';

    protected $_dependentTables = array('Application_Model_Relations');


    //seznam nekaterih prirejenih funkcij
    public function getOneByName($name) {
        $select = $this->select()->where('name=?', $name);
        return $this->_fetch($select);
    }
    …

}
```

Relations.php

```
class Application_Model_Relations extends Zend_Db_Table_Abstract
{
    protected $_name = 'relations';
    protected $_primary = 'id';

    protected $_referenceMap = array(
        'Thread' => array(
            'columns' => array('thread_id'),
            'refTableClass' => 'Application_Model_Threads',
            'refColumns' => array('id')
        ),
        'Post' => array(
            'columns' => array('post_id'),
            'refTableClass' => 'Application_Model_Posts',
            'refColumns' => array('id')
        )
    );

    function __construct() {
        parent::__construct();
    }

    public function insertRelation($data) {
        $this->insert($data);
    }
}
```

Za vnos podatkov v podatkovno bazo smo uporabili obrazce. Obrazce definiramo v direktoriju forms. Ker moramo vsako polje oz. vneseno vrednost preveriti, da bi ugotovili pravilno vrednost podatkov in s tem preprečili vnos škodljive kode v podatkovno bazo, katera lahko povzroči, da se celotna baza zbriše, spremenijo podatki ali celo pridobi pravice za dostope do nepooblaščenih sekcij. Ker smo projekt uporabljali zgolj v testne namene, smo preverjanje verodostojnosti podatkov preskočili. Za primer smo kreirali prirejene validatorje za preverbo unikatnega imena tem in sporočil. Vnešeno vrednost validator preveri s trenutnimi vrednostmi v podatkovni bazi. Če vnešena vrednost že obstaja, opozorimo uporabnika, drugače uporabniku dovolimo vnos. Validator smo kreirali v direktoriju library, kjer se nahajajo vse knjižnice. V imenu razreda uporabimo predpono My_, ki smo jo predhodno definirali v datoteki application.ini, kjer bomo hranili lastne razrede (v našem primeru `My_Validator_UniqueThread`, kar pomeni da se nahaja znotraj library > My > Validator).

Primer forme in prirejenega validatorja:

Thread.php

```
class Application_Form_Thread extends Zend_Form
{
    public function init()
    {
        $name = new Zend_Form_Element_Text('name');
        $name->setLabel('Ime teme')
                ->addValidator(new My_Validator_UniqueThread());

        $author = new Zend_Form_Element_Text('author');
        $author->setLabel('Avtor');

        $submit = new Zend_Form_Element_Submit('submit');
        $submit->setLabel('Potrdi');

        $this->addElements(array(
            $name,
            $author,
            $submit
        ));

        $this->setAttrib('class', 'thread');
    }
}
```

UniqueThread.php

```
class My_Validator_UniqueThread extends Zend_Validate_Abstract {

    const UNIQUE = 'unique';
    protected $_messageTemplates = array(
        self::UNIQUE => "'%value%' ni unikatno ime teme"
    );

    public function isValid($value) {
        $this->_setValue($value);
        $threadModel = new Application_Model_Threads();
        $thread = $threadModel->getOneByName($value);

        if(!empty($thread)) {
            $this->_error(self::UNIQUE);
            return false;
        }
        return true;
    }
}
```

Zadnji del projekta, ki smo ga morali implementirati, je bila logika projekta (pogledi in kontrolerji). Poglede smo kreirali v direktoriju modules > default > views > scripts, kontrolerje pa v direktoriju modules > default > controllers. Vsak kreirani kontroler mora imeti v poddirektoriju scripts svoj direktorij z enakim imenom (npr. IndexController mora imeti direktorij index) in poglede, ki predstavljajo akcije znotraj kontrolerja (npr. indexAction znotraj Index kontrolerja ima pogled index.phtml v direktoriju index). Imena akcij znotraj kontrolerja so lahko poljubna, razen v IndexController.php, kjer moramo imeti akcijo indexAction, ki je privzeta akcija, ko odpremo stran v brskalniku. Poljubnemu imenu moramo na koncu dodati pripono `[ime_akcije]Action` (začne se z malo začetnico). Pri projektu smo kreirali dva kontrolerja in znotraj vsakega kontrolerja, določeno število akcij oz. funkcij. ErrorController.php je namenjen napakam, ki se ob izvajanju sprožijo. IndexController.php je namenjen izvajanju naše aplikacije. Ker ima naša aplikacije štiri poglede smo kreirali štiri akcije. En pogled nam prikaže teme, en sporočila in ostala dva za vnos.

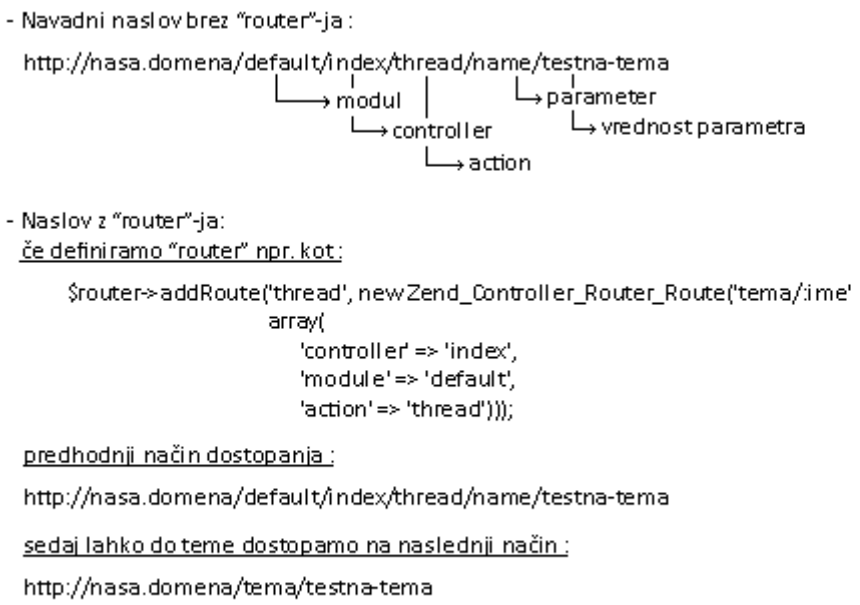
Primer IndexController.php:

```
class IndexController extends Zend_Controller_Action
{

    public function init()
    {
        /* Initialize action controller here */
    }

    public function indexAction()
    {
        $threadModel = new Application_Model_Threads();
        $this->view->threads = $threadModel->getAllThreads();
    }

    public function newthreadAction() {
        $threadForm = new Application_Form_Thread();
        $this->view->form = $threadForm;

        if($this->getRequest()->isPost()) {

            $form = $this->view->form;

            if($form->isValid($this->_getAllParams())) {

                $data = array(
                    'name' => $this->_getParam('name'),
                    'author' => $this->_getParam('author'),
                    'slug' => $this->_slugify($this->_getParam('name')),
                    'date' => date( 'Y-m-d H:i:s', time() )
                );

                $threadModel = new Application_Model_Threads();
                $threadModel->insertThread($data);

                $this->_helper->redirector('');
            }
        }
    }

  //nadaljevanje na drugi strani
```

```
public function newpostAction() {

        $s = $this->_request->s;

        $postForm = new Application_Form_Post();
        $this->view->form = $postForm;

        if($this->getRequest()->isPost()) {

            $form = $this->view->form;

            if($form->isValid($this->_getAllParams())) {

                $data = array(
                    'name' => $this->_getParam('name'),
                    'author' => $this->_getParam('author'),
                    'text' => $this->_getParam('text'),
                    'slug' => $this->_slugify($this->_getParam('name')),
                    'date' => date( 'Y-m-d H:i:s', time() )
                );

                $postModel = new Application_Model_Posts();
                $postModel->insertPost($data, $s);


                $threadModel = new Application_Model_Threads();
                $thread = $threadModel->getOneById($s);

                $this->_helper->redirector('thread', 'index', null, array('s' =>
                    $thread[0]['slug']));

            }
        }
    }

    public function threadAction() {
        $s = $this->_request->s;

        //var_dump($s);

        if($s != '') {
            $threadModel = new Application_Model_Threads();
            $posts = $threadModel->getPostsByThreadSlug($s);

            $this->view->posts = $posts;
            $this->view->threadName = $posts[0]['t_name'];
            $this->view->threadId = $posts[0]['t_id'];
        }
    }

    private function _slugify($name, $length = 50) {

        $string = strtolower($name);
        $string = preg_replace('/[^a-z0-9\s-]/', '', $string);
        $string = preg_replace('/\s[\s]+/', ' ', $string);
        $string = trim($string);
        //var_dump(strlen($string) <= $length ? strlen($string) : $length);
        substr(0, strlen($string) <= $length ? strlen($string) : $length);
        $string = trim($string);
        $string = preg_replace('/\s/', '-', $string);

        return $string;
    }
}
```

Za prikaz imena tem v spletnih naslovih smo uporabili funkcijo `_slugify`, ki nam iz imena izlušči vse posebne znake, ki niso podprti v naslovih s strani brskalnika npr. šumnike, presledke pa nam nadomesti z znakom minus, ker se tako izognemo nadležnemu zapisu presledkov s sklopom %20. Tak naslov je bolj prijazen spletnim iskalnikom tudi z vidika optimizacije SEO.

Zend Framework ima pri izbiri modula, kontrolerja in pogleda sledečo logiko: ko v brskalnik vtipkamo npr. http://nasa.domena/default/index/index, to pomeni, da se bo sprožila akcija indexAction znotraj kontrolerja IndexController.php v modulu default, npr. slika 13. Vse, kar je naknadno dodano, je v akcijo posredovano kot parameter, ki ga preberemo preko spremenljivke `$_REQUEST` (v Zend Frameworku uporabimo `$this->_request->[ime_parametra]`). Če želimo spletni naslov prirediti za lastne potrebe oz., če želimo izboljšati SEO aplikacije, lahko s pomočjo routerjev definiramo poljuben naslov. Routerji nam prav pridejo tudi, ko izdelujemo večjezične aplikacije in hočemo za vsak jezik kar najbolje optimizirati spletne povezave.



Slika 13. Načini, kako lahko definiramo spletne naslove po lastni potrebi.

V našem projektu smo vključili štiri poglede. Pogledi vsebujejo kodo HTML in PHP (od tod tudi ime za pripono .phtml). V pogledih prikažemo vrednosti, ki smo jih predhodno shranili v pripadajočem kontrolerju. V kontrolerju dodelimo vrednost z `$this->view->[ime_variable]`, v pogledu pa jo prikličemo z `$this->[ime_variable]`. V pogledih lahko uporabimo navadno PHP kodo ali nekoliko prirejeno za potrebe pogledov, s katero ohranimo sintakso HTML skoraj neokrnjeno.

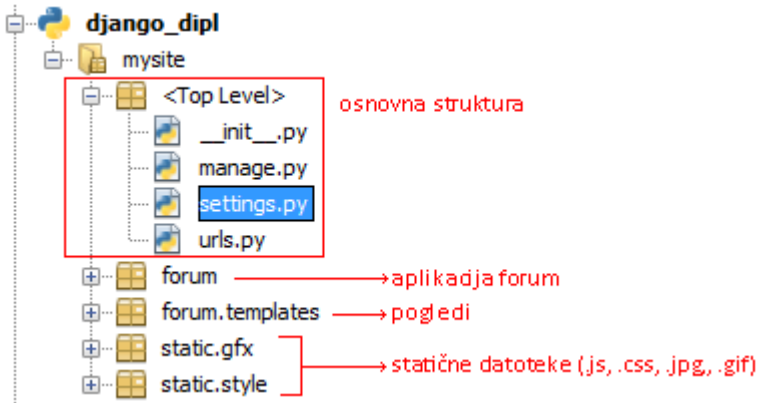
Primer našega »index.phtml« view-a:

```
<div id="content">
    <table cellspacing="0" width="100%" border="0">
        <tr>
            <th align="left" class="border-left border-top">Ime teme</th>
            <th align="center" class="border-top">Število sporočil</th>
            <th align="center" class="border-top">Avtor</th>
            <th align="center" class="border-top border-right">Datum
nastanka</th>
        </tr>
        <?php if(!empty($this->threads)): ?>
            <?php //var_dump($this->threads); ?>
            <?php $idx = count($this->threads)-1; ?>
            <?php //var_dump($idx); ?>
            <?php foreach($this->threads as $key => $value): ?>
                <?php //var_dump($key); ?>
                <tr class="thread">
                     <td class="name no-border<?= ($idx == $key) ? ' last' : '';
                         ?>" align="left" width="70%"><a href="<?php echo
                         '/index/thread?s=' . $value['slug'] ?>"><?php echo
                         $value['name'] ?></a></td>
                     <td class="posts<?= ($idx == $key) ? ' last' : ''; ?>"
                         align="center" width="10%"><?php echo $value['count']
                         ?></td>
                     <td class="author<?= ($idx == $key) ? ' last' : ''; ?>"
                         align="center" width="10%"><?php echo $value['author']
                         ?></td>
                     <td class="date<?= ($idx == $key) ? ' last' : ''; ?>"
                         align="center" width="10%"><?php echo $value['date']
                         ?></td>
                </tr>
            <?php endforeach; ?>
        <?php endif; ?>
    </table>
    <div id="actions"><button
onclick="location.href='/index/newthread'">Nova</button></div>
</div>
```

Za lepši prikaz celotnega foruma in form smo dodali še nekaj stilov v datoteko styles.css, ki se nahaja v direktoriju public > styles.

4.2.2 Programsko okolje Django

Pri izdelavi projekta v okolju Django, smo morali najprej dodati knjižnice (v direktorij Lib/site-packages) v direktorij Python, nato smo morali namestiti še modul za poganjanje podatkovnih baz MySQL. Za kreiranje projekta v okolju Django smo nastavili še v Environment Variables pot do direktorija django/bin. Pri združevanju okolja Django in programskega jezika Python moramo paziti na kompatibilnost obeh verzij. Ko smo imeli funkcionalno okolje, smo kreirali naš projekt v konzoli z ukazom django-admin.py startproject mysite. S slednjim ukazom smo dobili osnovno strukturo projekta, npr. slika 14, ki vsebuje __init__.py (prazna datoteka, ki pove da gre za Pythonov paket), `manage.py` (pripomoček za izvajanje ukazov v konzoli), `settings.py` (vsebuje nastavitve) ter `urls.py` (kjer definiramo veljavne spletne povezave za naš projekt). Potem, ko smo ustvarili projekt smo pognali ukaz python manage.py startapp forum s katerim smo ustvarili direktorij forum znotraj našega projekta z naslednjimi datotekami __init__.py, models.py, tests.py in views.py. Za naše potrebe smo ustvarili še direktorij `templates`, kjer smo hranili poglede ter direktorij `static`, kjer smo hranili statične datoteke kot npr. predloge CSS ter slike.



Slika 14. Struktura direktorijev v projektu Django.

Sedaj, ko smo imeli strukturo projekta, smo nastavili nekaj parametrov v `settings.py` datoteki. V datoteki je veliko nastavitev. Izpostavili bomo samo za nas relevantne nastavitve (nastavitev podatkov za bazo, poti do direktorijev itd.).

Primer naše `settings.py` datoteke:

```
DATABASES = {
          'default': {
            'ENGINE': 'django.db.backends.mysql', # Add
                  'postgresql_psycopg2', 'postgresql', 'mysql', 'sqlite3'
                  or 'oracle'.
            'NAME': 'dipl_forum_django',                      # Or path
                  to database file if using sqlite3.
            'USER': 'root',                      # Not used with sqlite3.
            'PASSWORD': '',                  # Not used with sqlite3.
            'HOST': '',                      # Set to empty string for
                        localhost. Not used with sqlite3.
            'PORT': '',                      # Set to empty string for
                  default. Not used with sqlite3.
    }
}

MEDIA_ROOT = '/static/'
MEDIA_URL = '/static/'
ROOT_URLCONF = 'mysite.urls'

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    # dodamo našo aplikacijo
    'forum'
)
```

Da bi Django samodejno ustvaril bazo in tabele v njej, smo morali definirati modele v datoteki `models.py`. Vse modele definiramo znotraj ene datoteke za razliko od Zend Frameworka, kjer je vsak model imel svoj razred v svoji datoteki.

Primer našega `models.py`:

```
from django.db import models

# Create your models here.
class Posts(models.Model):
    name = models.CharField(max_length=100)
    text = models.TextField()
    author = models.CharField(max_length=100)
    date = models.DateTimeField("date published")
    slug = models.SlugField()

    def __unicode__(self):
        return self.name

class Threads(models.Model):
    name = models.CharField(max_length=100)
    author = models.CharField(max_length=100)
    date = models.DateTimeField("date published")
    slug = models.SlugField()
    posts = models.ManyToManyField(Posts, through='Relations')

    def __unicode__(self):
        return self.name

class Relations(models.Model):
    thread = models.ForeignKey(Threads, related_name =
      "%(app_label)s_%(class)s_related")
    post = models.ForeignKey(Posts, related_name =
      "%(app_label)s_%(class)s_related")
```

Tukaj smo imeli nekaj težav z implementacijo M2M, saj smo se zadeve lotili na podoben način kot pri Zend Frameworku. Za relacijo med temami (Threads) in sporočili (Posts) smo poskrbeli z definicijo `models.ManyToManyField(Posts, through='Relations')`. V tabeli, ki hrani relacije, pa definiramo foreign key za vsako od relevantnih tabel. Ko smo definirali vse potrebne razrede, smo v konzoli pognali ukaz python manage.py syncdb, ki nam je ustvaril vse tabele v podatkovni bazi, ki smo jo predhodno nastavili v datoteki `settings.py`.

Naslednje na vrsti so sledili obrazci, ki smo jih definirali v `forms.py`. Tako kot pri modelih tudi tu definiramo vse znotraj ene datoteke. Vsakemu vnosnemu polju znotraj posameznega obrazca lahko po potrebi dodamo parametre. Obrazcem so sledili kontrolerji in pogledi. Ime views.py je lahko zavajajoče, saj tu definiramo kontrolerje. Enako kot pri Zend Frameworku imamo tudi tu za vsak pogled posamezno akcijo. Za rabo modelov in obrazcev moramo na začetku vključiti razrede, kjer smo jih definirali.

Primer `forms.py`:

```
#! /usr/bin/python
from django import forms

class ThreadForm(forms.Form):
    name = forms.CharField()
    author = forms.CharField()

class PostForm(forms.Form):
    name = forms.CharField()
    author = forms.CharField()
    text = forms.CharField( widget=forms.Textarea(attrs={'rows': 24,
'cols': 80}) )
```

Primer `views.py`:

```
# Create your views here.
from django.template import Context, loader
from django.http import HttpResponse, HttpResponseRedirect, Http404
from django.template.loader import get_template
from forum.forms import ThreadForm, PostForm
from django.shortcuts import render_to_response
from forum.models import Threads, Posts, Relations
from datetime import datetime
from django.template.defaultfilters import slugify
from django.db.models import Count
# from django.db import connection


def index(request):
    threads = Threads.objects.all().annotate(post_count=Count('posts'))

    return render_to_response('index.html', {
            'threads': threads,
    })

#nadaljevanje na drugi strani…
```

```
def newthread(request):
    if request.method == 'POST': # If the form has been submitted...
        form = ThreadForm(request.POST) # A form bound to the POST data
        if form.is_valid(): # All validation rules pass
            # Process the data in form.cleaned_data

            name = form.cleaned_data['name']
            author = form.cleaned_data['author']
            slug = slugify(name)

            thread = Threads(name = name, author = author,
date=datetime.now(), slug=slug)
            thread.save()

            return HttpResponseRedirect('/forum/') # Redirect after POST
    else:
        form = ThreadForm() # An unbound form
        return render_to_response('newthread.html', {
            'thread_form': form,
        })

def thread(request, slug):
    try:
        thread = Threads.objects.get(slug=slug)
        posts = thread.posts.all()

    except Threads.DoesNotExist:
        return HttpResponseRedirect('/forum/')

    return render_to_response('thread.html', {
            'thread_name' : thread.name,
            'thread_id' : thread.id,
            'posts' : posts
    })

def newpost(request, id):
    try:
        thread = Threads.objects.get(pk = id)
    except Threads.DoesNotExist:
        return HttpResponseRedirect('/forum/')

    if request.method == 'POST': # If the form has been submitted...
        form = PostForm(request.POST) # A form bound to the POST data
        if form.is_valid(): # All validation rules pass
            # Process the data in form.cleaned_data

            name = form.cleaned_data['name']
            author = form.cleaned_data['author']
            slug = slugify(name)
            text = form.cleaned_data['text']
            post = Posts(name = name, author = author,
                  date=datetime.now(),text=text, slug=slug)
            post.save()

            relation = Relations(thread_id = id, post_id = post.id)
            relation.save()

#nadaljevanje na drugi strani…
```

```
          # Redirect after POST
          return HttpResponseRedirect('/forum/thread/' + thread.slug +
                '/')
      else:
        form = PostForm() # An unbound form

        return render_to_response('newpost.html', {
            'post_form': form,
            'thread_id': id
        })

# custom function for debugging
def __print_obj(obj):
    print ''
    print '-----------------------------------'
    for value in obj.values():
        print value
    print '-----------------------------------'
    print ''
```

Predno lahko aplikacijo uporabljamo moramo definirati še vse veljavne spletne povezave v datoteki `urls.py`. Za nadzor nad pravilnostjo parametrov lahko uporabimo tudi regularne izraze. Tu se srečamo tudi s prvo striktnostjo okolja Django. Za razliko od Zend Frameworka moramo tu definirati vse parametre, ki jih bomo posredovali akcijam; če slučajno nismo definirali parametra in ga naknadno posredujemo v naslovu, se sproži napaka. Pravila za definiranje spletnih naslovov se definirajo v paru, najprej seveda struktura in nato še pripadajoča akcija. Pri pisanju pogledov smo kodo prekopirali iz Zend Frameworka in jo prilagodili okolju Django (sintakso <? smo zamenjali z {%, za izpis vrednosti spremenljivk smo uporabili dvojne zavite oklepaje in zaklepaje). Vrednosti se prenašajo na pogled preko `return render_to_response`. Posebnost so tudi posebni znaki, v našem primeru šumniki, ki smo jih morali predstaviti v številskem zapisu, npr. »Š« kot »Š«, ker je drugače javljalo napako.

Primer `urls.py`:

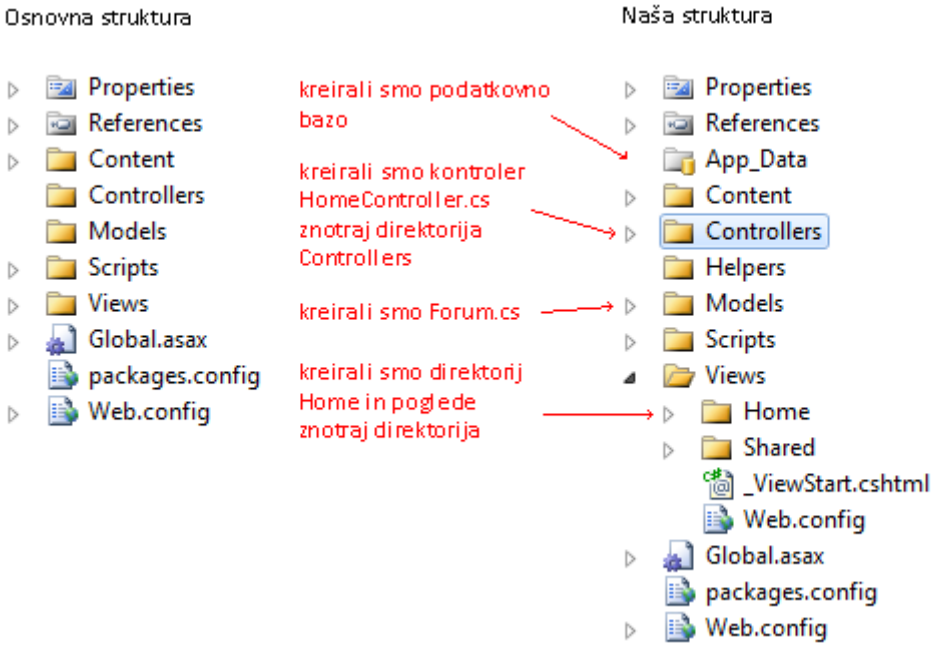
```
from django.conf.urls.defaults import *

# Uncomment the next two lines to enable the admin:
# from django.contrib import admin
# admin.autodiscover()

urlpatterns = patterns('',
    url(r'^forum/$', 'forum.views.index'),
    url(r'^forum/newthread/$', 'forum.views.newthread'),
    url(r'^forum/thread/(?P<slug>[a-z-]+)/$', 'forum.views.thread'),
    url(r'^forum/newpost/(?P<id>[0-9]+)/$', 'forum.views.newpost'),
    (r'^static/(?P<path>.*)$', 'django.views.static.serve',
        {'document_root': 'static'})
)
```

Primer `index.html`:

```
{% extends "base.html" %}

{% block content %}
<table cellspacing="0" width="100%" border="0">
    <tr>
        <th align="left" class="border-left border-top">Ime teme</th>
        <th align="center" class="border-top">&#352;tevilo
sporo&#269;il</th>
        <th align="center" class="border-top">Avtor</th>
        <th align="center" class="border-right border-top">Datum
nastanka</th>
    </tr>
    {% if threads|length > 0 %}
        {% for thread in threads %}
            <tr class="thread">
                <td class="name no-border{% if forloop.last %} last {%
                      endif %}" align="left" width="70%"><a
                      href="/forum/thread/{{ thread.slug }}/">{{
                      thread.name }}</a></td>
                <td class="posts{% if forloop.last %} last {% endif %}"
                      align="center" width="10%">{{ thread.post_count
                      }}</td>
                <td class="author{% if forloop.last %} last {% endif %}"
                      align="center" width="10%">{{ thread.author }}</td>
                <td class="date{% if forloop.last %} last {% endif %}"
                      align="center" width="10%">{{ thread.date }}</td>
            </tr>
        {% endfor %}
    {% endif %}
</table>
<div id="actions">
   <button onclick="location.href='/forum/newthread'">Nova</button>
</div>
{% endblock %}
```

4.2.3 Programsko okolje ASP.NET MVC

Namestitev okolja ASP.NET MVC je bila vseh treh primerih še najbolj enostavna. Potem, ko smo z interneta povlekli namestitveno datoteko, smo imeli v nekaj klikih postavljeno celotno okolje. Namestitveni paket vsebuje vse od programa za razvijanje, strežnika za poganjanje aplikacij do strežnika za podatkovne baze. Ko se je namestitev končala, smo pognali Microsoft Visual Web Developer. Med ponujenimi opcijami smo izbrali New Project. Za tip projekta smo izbrali ASP.NET MVC 3 Web Application v programskem jeziku C# ter ga poimenovali aspnet_forum, npr. slika 15. Sistem, v katerem bomo pisali predloge, smo pustili na privzeto (Razor) ter izbrali Empty, kot prazen projekt z osnovno strukturo.



Slika 15. Struktura projekta ASP.NET MVC.

Z desnim klikom na projekt smo prvo dodali direktorij App_Data, kjer so shranjeni podatki v obliki SQL Compact Edition 4.0, na voljo pa imamo tudi SQL Express Edition, ki smo jo kasneje tudi pretestirali. Za uporabo ene ali druge podatkovne baze je treba definirati atribut `connectionString` znotraj datoteke WebConfig. Za začetek smo definirali modele v direktoriju Models znotraj datoteke Forum.cs. Znotraj datoteke smo definirali vse potrebne razrede `Post` in `Thread` ter razred `ForumDBContext`, ki predstavlja sejo s podatkovno bazo. Naknadno smo dodali še razred `ForumInitializer`, preko katerega smo ob vsakem ponovnem zagonu aplikacije prenesli spremembe narejene na modelih v podatkovno bazo. Da bi razred `ForumInitializer` dejansko imel funkcijo, smo morali dodati še vrstico kode v datoteko Global.asax in sicer `System.Data.Entity.Database.DbDatabase.SetInitializer(new ForumInitializer());` pod `Application_Start`.

Primer obeh povezav:

```
<connectionStrings>
    <!--način za SQL Compact Edition -->
    <add name="ForumDBContext"
         connectionString="Data Source=|DataDirectory|Forum.sdf"
         providerName="System.Data.SqlServerCe.4.0"/>

    <!--način za SQL Express Edition -->
    <add name="ForumDBContext"
          connectionString="Data
          Source=.\SQLEXPRESS;database=Database2;AttachDbFilename=|DataDirectory|Databa
          se2.mdf;Integrated Security=True;User Instance=True"
         providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Primer »Forum.cs«:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;
using System.Data.Entity.Database;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity.ModelConfiguration;

namespace aspnet_forum.Models
{
    public class Forum
    {
    }

    public class Thread
    {
        public int ID { get; set; }

        [Required]
        public string Name { get; set; }

        [Required]
        public string Author { get; set; }

        public DateTime Date { get; set; }
        public string Slug { get; set; }

        public virtual ICollection<Post> Posts { get; set; }
    }

    public class Post
    {
        public int ID { get; set; }

        [Required]
        public string Name { get; set; }

        [Required]
        public string Author { get; set; }

        public DateTime Date { get; set; }

        [Required]
        public string Text { get; set; }
        public string Slug { get; set; }

        [ForeignKey("Thread")]
        public int ThreadId { get; set; }
        public virtual Thread Thread { get; set; }
    }

// nadaljevanje na drugi strani
```

```
    public class ForumDBContext : DbContext
    {
        public DbSet<Thread> Threads { get; set; }
        public DbSet<Post> Posts { get; set; }
    }

    public class ForumInitializer : DropCreateDatabaseAlways<ForumDBContext>
    {

    }
}
```

V razred `ForumInitializer` lahko dodamo tudi testne podatke, ki se ob zagonu aplikacije naložijo v bazo. Ob vsakem modelu lahko pri lastnostih razreda dodamo tudi atribute, kot so `Required`, `MaxLength`, `DisplayFormat`, `DisplayName` itd. Preko tek atributov določimo, katere vnose bomo dovolili in katere napake bomo uporabnikom sporočili ob morebitnem napačnem vnosu. Z atributi, kot je `ForeignKey`, definiramo relacijo z drugim razredom (fizično je to tabela).

Zdaj, ko smo imeli definirane modele in s tem tudi strukturo baze, smo se lotili programiranja kontrolerja HomeController.cs. Akcije v ASP.NET MVC-u so še nekoliko bolj striktne kot pri okolju Django. Najbolj smo to opazili pri pogledu NewPost.cshtml, kjer smo morali v parametrih podati ID teme, ko smo skušali dodati sporočilo. Pri potrditvi podatkov pa smo morali posredovati ID teme ter ostale podatke. Za uporabo parametrov v naslovih je treba pred akcijami predhodno definirati atributa `HttpPost` ali `HttpGet`. Imena parametrov, ki jih posredujemo morajo biti enaka tudi v definicijah akcij, npr. če posredujemo `threadId=2`, mora biti v akciji definirano `int threadId`. Število parametrov, ki jih akcija lahko sprejme, je enako številu vhodnih parametrov, ki smo jih definirali (v primeru, ko smo definirali razred `Post post` lahko sprejema le atribute razreda).

Primer HomeController.cs:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using aspnet_forum.Models;
using System.Text.RegularExpressions;

namespace aspnet_forum.Controllers
{
    public class HomeController : Controller
    {

        private ForumDBContext dbForum = new ForumDBContext();

        public ActionResult Index()
        {

            ViewBag.Threads = dbForum.Threads.ToList();
            return View();
        }

        public ActionResult NewThread()
        {
            return View();
        }

        [HttpPost]
        public ActionResult NewThread(Thread thread)
        {
            if (ModelState.IsValid)
            {
                thread.Slug = Slugify(thread.Name);
                thread.Date = DateTime.Now;
                dbForum.Entry(thread).State = EntityState.Added;
                dbForum.SaveChanges();
                return RedirectToAction("Index");
            }

    //nadaljevanje na drugi strani
```

```
            else
            {
                return View();
            }
        }

        public ActionResult Thread(string threadSlug)
        {
            Thread thread = dbForum.Threads.First(model => model.Slug ==
                        threadSlug);
            var Thread = dbForum.Threads.Include("Posts").SingleOrDefault(t =>
                        t.Slug == threadSlug);
            ViewBag.Posts = Thread.Posts;
            ViewBag.ThreadName = Thread.Name;
            ViewBag.ThreadId = Thread.ID;
            return View();
        }

        [HttpGet]
        public ActionResult NewPost(int threadId)
        {
            ViewBag.ThreadId = threadId;
            return View();
        }

        [HttpPost]
        public ActionResult NewPost(Post post, int threadId)
        {
            if (ModelState.IsValid)
            {
                Thread thread = dbForum.Threads.First(model => model.ID ==
                        threadId);
                post.Slug = Slugify(post.Name);
                post.Date = DateTime.Now;
                dbForum.Entry(post).State = EntityState.Added;
                dbForum.SaveChanges();
                return RedirectToAction("Thread", "Home", new { threadSlug =
                        thread.Slug });
            }
            else
            {
                return View();
            }
        }

        private static string Slugify(string phrase, int maxLength = 50)
        {
            string str = phrase.ToLower();
            str = Regex.Replace(str, @"[^a-z0-9\s-]", "");
            str = Regex.Replace(str, @"[\s-]+", " ").Trim();
            str = str.Substring(0, str.Length <= maxLength ? str.Length :
                  maxLength).Trim();
            str = Regex.Replace(str, @"\s", "-");
            return str;
        }
    }
}
```

Pogledi so praktično v vseh okoljih enaki. Koda se lahko prenese iz enega okolja v drugo. Paziti je treba le na sintaktične pravilnosti pri izpisovanju. Če uporabimo način Razor, ki smo ga izbrali tudi mi, potem pred imenom spremenljivke uporabimo znak »@«. V prejšnjih primerih smo izpostavili pogled index, v tem primeru bomo izpostavili pogled NewPost.cshtml, ker se nazorno prikaže uporaba definicije model razreda in njegovih atributov, ki smo jih definirali v datoteki Forum.cs. Na začetku kode je pomembno, da v pogled vključimo model, v našem primeru je bil to razred `Post` (`@model aspnet_forum.Models.Post`). Pod vnosna polja, ki jih mora vnesti uporabnik, definiramo lastnosti razreda, ki so relevantne za določeno vnosno polje. Vrednosti iz kontrolerja v pogled prenašamo preko objekta ViewBag ali objekta ViewData (do vrednosti dostopamo preko ključa in ne preko lastnosti kot pri ViewBagu). Pri našem projektu smo se odločili za način ViewBag, ker podpira dinamično dodajanje objektov. Pri ViewData pa je slabost tudi ta, da je treba v pogledu objekte ponovno prirediti prvotnemu tipu.

Razlika med načinom ViewData in ViewBag:

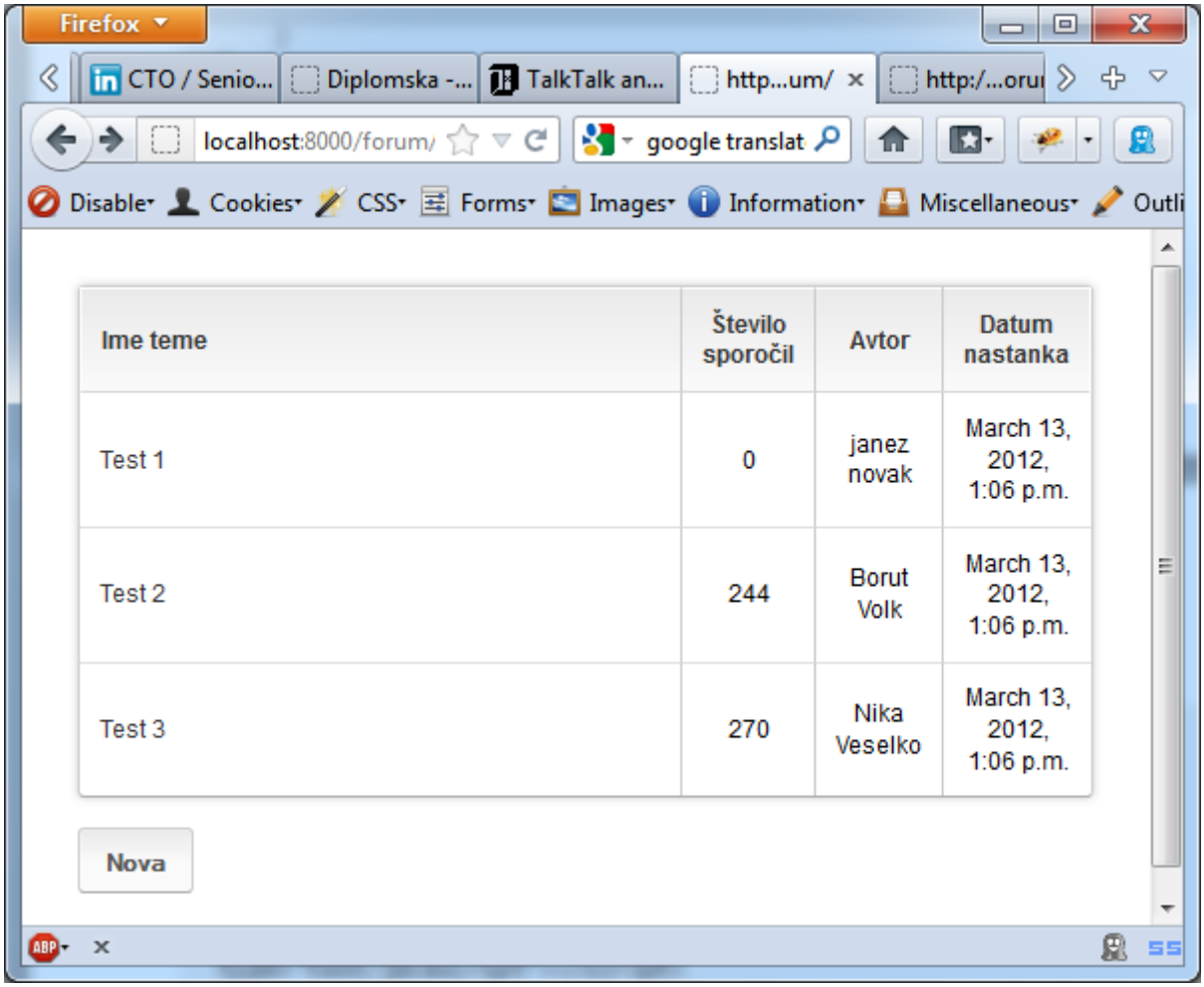
```
// *** raba v controller-ju ***

var thread = new Thread
{
    Name = »Test«,
    Author = »Janez Novak«,
    Date = DateTime.Now,
    Slug = »test«
}

ViewData[»Thread«] = thread;
ViewBag.Thread = thread;

// *** raba v view-u ***

// v ViewBag lahko dodajamo nove objekte za razliko od ViewData

ViewBag.FooBar = »FooBar«;
var thread = ViewData[»Thread«] as Thread;

// izpis vrednosti

@ViewBag.Thread.Author
@thread.Author
```

Primer pogleda NewPost.cshtml:

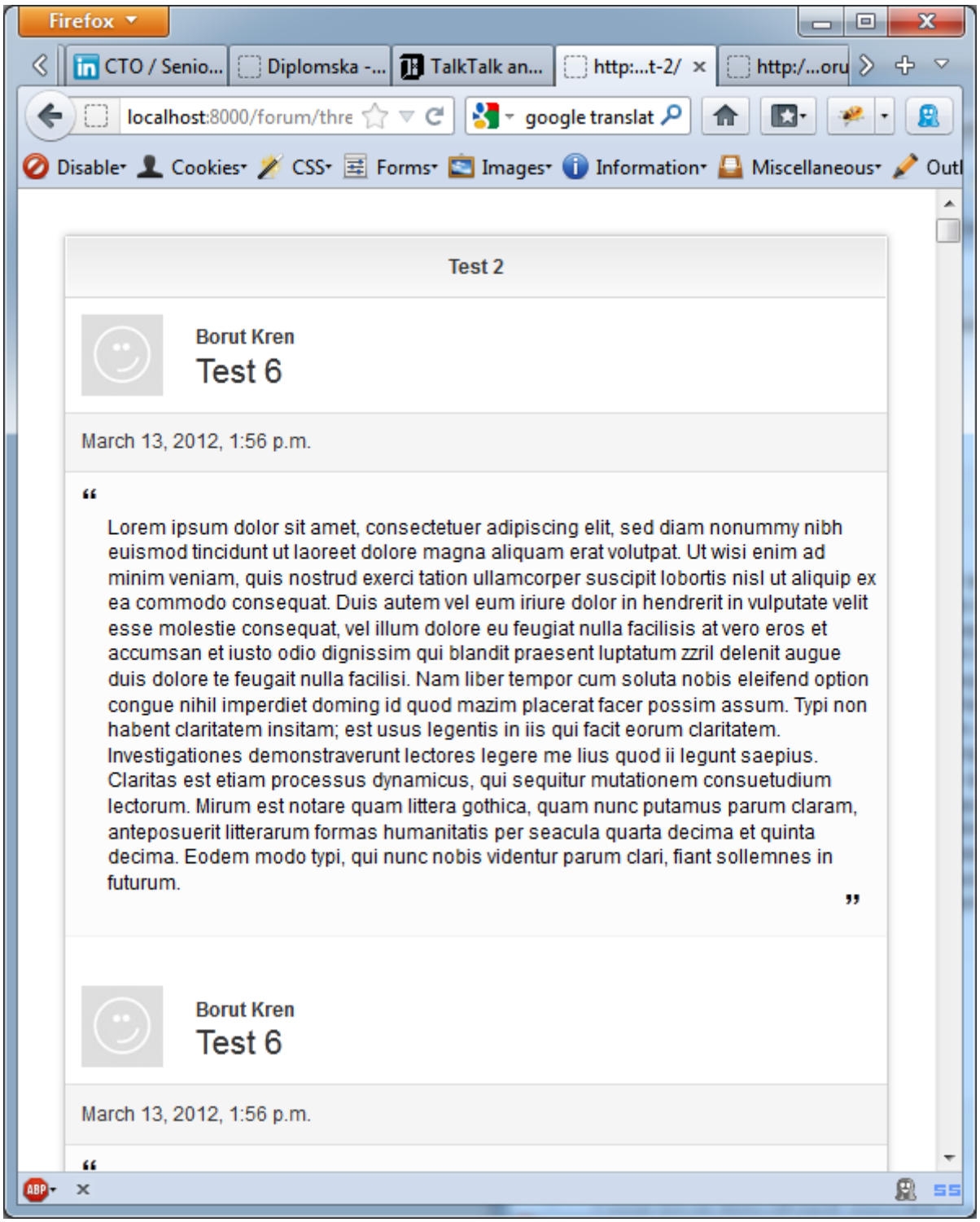
```
@model aspnet_forum.Models.Post

@{
    ViewBag.Title = "NewPost";
}

<script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")"
type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.min.js")"
type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")"
type="text/javascript"></script>

@using (Html.BeginForm("newpost", "home", new { threadId = ViewBag.threadId },
FormMethod.Post))
{
    <dl class="thread_form">
        <dt id="name-label">
            @Html.LabelFor(model => model.Name)
        </dt>
        <dd id="name-element">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </dd>
        <dt id="author-label">
            @Html.LabelFor(model => model.Author)
        </dt>
        <dd id="author-element">
            @Html.EditorFor(model => model.Author)
            @Html.ValidationMessageFor(model => model.Author)
        </dd>
        <dt id="text-label">
            @Html.LabelFor(model => model.Text)
        </dt>
        <dd id="text-element">
            @Html.TextAreaFor(model => model.Text, new { cols = "80", rows = "24"
})
            @Html.ValidationMessageFor(model => model.Text)
        </dd>
        <dt id="submit-label"> </dt>
        <dd id="submit-element">
            <input type="submit" value="Potrdi" />
        </dd>
    </dl>

}
```

4.2.4 Končni izdelek

Končnemu forumu (slika 16) in seznamu sporočil (slika 17) smo na koncu dodali še stile in jih za boljšo preglednost in kompatibilnost v vseh brskalnikih prikazali v tabelah.



Slika 16. Končni izgled foruma.



Slika 17. Končni izgled sporočil v temi.

4.3 Primerjava

Primerjanje programskih okolij vključuje veliko različnih dejavnikov. Ločimo jih na različne vidike. Prvi vidik je programerski. S tega vidika je zelo pomembno, kakšna je krivulja učenja in privajanja na programsko okolje. Kasneje je ključnega pomena tudi vmesnik, v katerem programiramo, kakšno pomoč nam ponuja in koliko so določeni trivialni procesi avtomatizirani, kompatibilnost okolja za nazaj, možnost vključevanja različnih implementacij podatkovnih struktur ter sama abstraktnost okolja. Drugi vidik je ekonomski. Pri tem vidiku je pomembno, kakšni izdatki se bodo pojavili pri nakupu programske opreme in licenc, kakšno podporo bomo imeli ob morebitnih izpadih oz. varnostnih luknjah, koliko dodatnih stroškov bomo imeli ob morebitni nadgradnji aplikacij in najpomembnejše ali nam bo uspelo projekt oz. aplikacijo dokončati v zastavljenem roku.

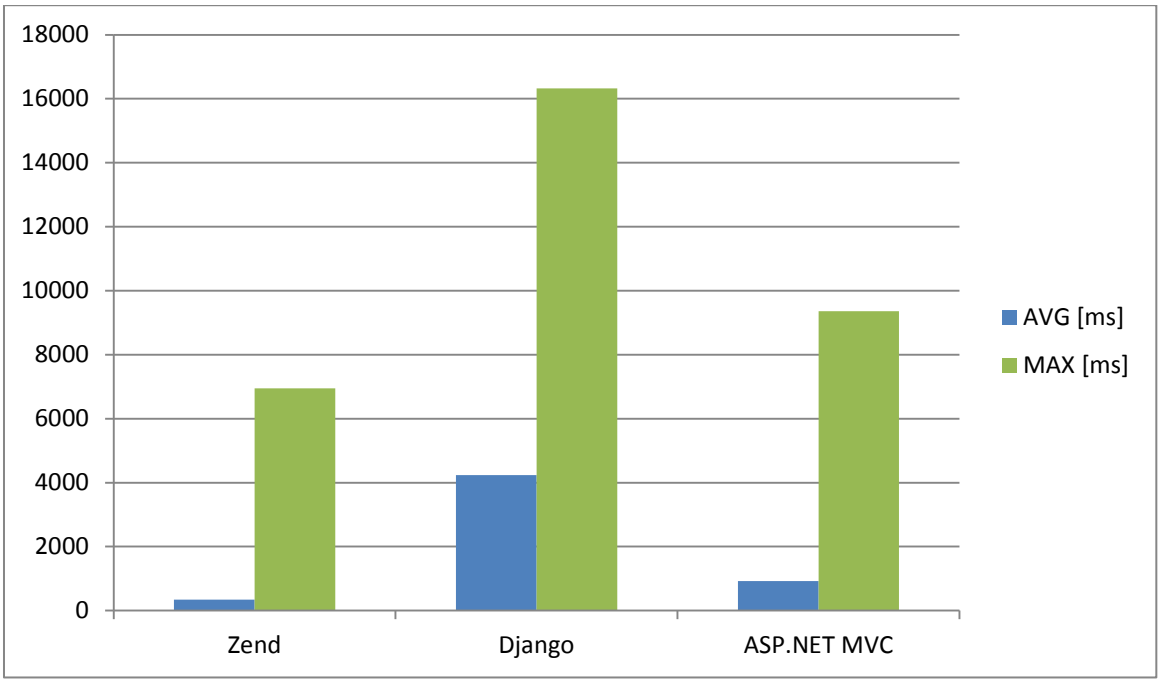
Spoznavanje samega koncepta MVC je še največja ovira pri programiranju z okolji. Pri izdelavi projektov smo največ časa porabili, da smo odkrili kako MVC sploh deluje, katere komponente vsebuje in kaj je namen vsake od teh komponent. Prehod iz enega okolja v drugo niti ni tako zahtevno, previden je treba biti le pri določenih specifikah enega in drugega programskega jezika, kot so npr. sintaktična pravila.

Predvsem Zend Framework in ASP.NET MVC imata dobro podprta vmesnika za razvijanje aplikacij, Django pa jima zvesto sledi na tem področju. Pri prvih dveh se orodja razvijajo v sklopu programskega okolja, pri slednjem pa se orodja razvijajo s strani drugih ponudnikov. Pri vseh je skupno, da moramo za dobro orodje plačati. Zend Studio temelji na platformi Eclipse, ASP.NET MVC pa je podprt v Visual Studiu. Za razvoj aplikacij v Djangu obstajajo različna orodja, ki svoje delo opravljajo več ali manj zadovoljivo. Pri našem razvoju smo pri razvoju aplikacije v Zend Frameworku uporabili NetBeans, ki ima izvrstno podporo in je brezplačen odprtokoden sistem. Pomoč nudi pri poganjanju skriptnih ukazov Zend ter namigov pri pisanju kode, kot pri odpravljanju napak in razhroščevanju aplikacije. Za ASP.NET MVC smo na Microsoftovi strani dobili t.i. celotni paket za razvoj, med katerim je tudi Visual Web Developer, brezplačni program za razvoj aplikacij v ASP.NET MVC-u in še nekaterih drugih jezikih. Pri Djangu smo se sprva odločili za PyScripter, ki je ravno tako brezplačen, vendar zna včasih javiti kako napako. Za razvijanje v okolju Django mnogi prisegajo na IDE PyCharm, vendar je brezplačna verzija na voljo le odprtokodnim projektom in izobraževalnim ustanovam. Za osebno ali komercialno rabo pa je treba kupiti licenco, saj nam je kot poizkusna verzija na voljo le 30 dni. Za naše potrebe smo sčasoma Django projekt nadaljevali v NetBeansu. Za razvoj Django projekta v NetBeansu obstaja vtičnik, ki ga moramo naknadno namestiti. Pomanjkljivost pri vtičniku je, da ni več podprt s strani razvijalcev NetBeansa, s tem pa je razvijanje prepuščeno zunanjim razvijalcem. Vtičnik nam pri pisanju kode ne nudi namigov.

Vsa tri okolja podpirajo tako podatkovne strukture SQL kot NoSQL. Pri strukturah SQL so najbolj uporabljene MySQL, MSSQL, DB2, Oracle DB, PostgreSQL itd., pri NoSQL pa so podpore za CouchDB, MongoDB (vsi trije), ASP.NET pa ima podporo tudi za Redis, RavenDB ter MemcacheDB. Za vse rešitve NoSQL je treba dobro poznati programsko okolje. Vse rešitve, ki se ponujajo na spletu, se v dobršnji meri še razvijajo in dopolnjujejo. Za razvoj aplikacij je pomembno tudi vzdrževanje različic podatkovnih struktur in objektov, ki predstavljajo podatkovno bazo. Migracijska orodja nam omogočajo vsakršno spremembo sheme podatkovnih baz od spremembe podatkovnega tipa kolone v bazi do razširitve tabel z izbiro določene verzije, ki smo jo predhodno shranili. Na voljo imamo opcijo za samodejno shranjevanje različic ob vsakršni spremembi modelov. Za okolje ASP.NET obstaja veliko orodij za ohranjanje različic modelov in podatkovnih baz (med najbolj uporabljenimi so mogoče Migrator.NET, Tarantino, RoundhousE, Fluent Migrations itd.). Vsa so nekako še v zgodnji fazi razvoja, čeprav jih veliko razvijalcev že kar s pridom uporablja. Za okolje Django obstajata predvsem dva projekta, in sicer South ter Django Reversion. Pri ogrodju Zend pa hranjenje in posodabljanje sheme podatkovnih baz, lahko dosežemo z implementacijo odprtokodnega orodja Doctrine ORM. Shemo podatkovne baze in vnosne podatke definiramo v datotekah YAML. Te se ob uspešno izvedenih ukazih v ukaznih vrsticah bodisi naložijo v podatkovno bazo bodisi jo spremenijo.

Pri izbiri ogrodja je dobro biti pozoren tudi na sekundarna orodja ali pa sisteme, ki so že v uporabi in bodo soodvisna z aplikacijo. Microsoft je s svojimi izdelki na trgu več ali manj prva opcija med večjimi podjetji, ko se gre za implementacijo informacijske tehnologije v delovni proces podjetja. Če podjetje ali izobraževalna ustanova uporablja npr. servise Active Directory za hrambo podatkov o zaposlenih ali študentih, je verjetno najboljša izbira ogrodje ASP.NET MVC že zaradi same kompatibilnosti tehnologij. V e-commerce panogi se vse pogosteje uporablja ogrodje Zend, verjetno tudi zato, ker so prvotne odprtokodne rešitve bile razvite v programskem jeziku PHP. Veliko dediščine in vtičnikov je že razvitih v kodi PHP, tako da je prenos v ogrodje Zend praktično enostavno. Za razliko od slednjih dveh pa je ogrodje Django bolj priljubljeno v namenskih aplikacijah (Spotify, Disqus, Instagram, več na [12]) in prva izbira med startup podjetji (tudi med slovenskimi). Priljubljen je tudi zato, ker temelji na programskem jeziku Python in kot tako ponuja nov in kompaktnejši način pisanja kode. Ob enem pa na spletu najdemo vrsto zanimivih pripomočkov, s katerimi lahko izboljšamo aplikacijo in popestrimo uporabniško izkušnjo.

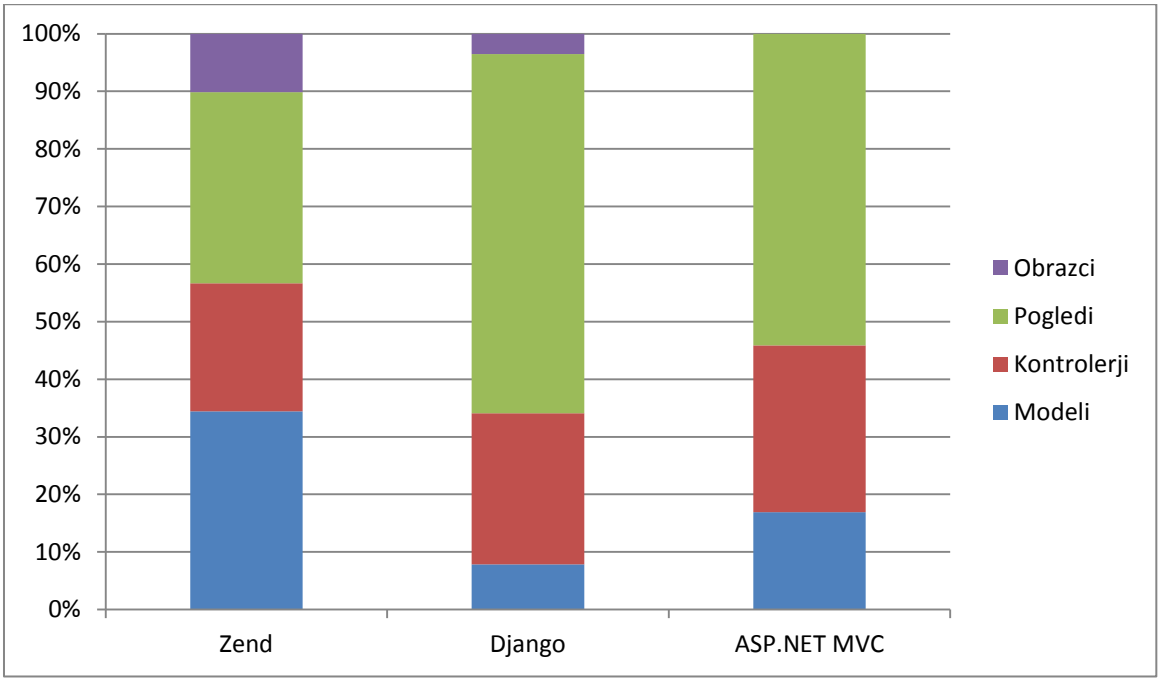
Z vsemi tremi programskimi okolji smo naredili majhen stress in load test. Okolja nismo prirejali in smo jih pustili v čisto osnovni verziji. Nekatera okolja npr. Django smo testirali na razvojnem strežniku, ki seveda ni priporočljiv za produkcijo. S tem nismo želeli prikazati, katero okolje je najboljše oz. dajati prednosti enemu ali drugem, hoteli smo zgolj prikazati kako se okolja odrežejo že v osnovi z navadnimi nastavitvami. Za optimizirano delovanje aplikacij je treba najprej prirediti strežniško strukturo npr. kot cluster rešitev in dodati load balancerje. Test je bil sledeč in sicer 100 vzporednih niti (100 uporabnikov). Začeli smo z desetimi in vsakih 30 sekund dodali 10 novih. Nato smo s končnimi stotimi, nadaljevali še 300 sekund. Pri ASP.NET MVC-u je bilo uspešnih 2906 od 2906 poslanih zahtev (100% uspešnost), pri Zend Frameworku je bilo uspešnih 3003 od 3011 poslanih (99,73% uspešnost), pri Djangu pa le 904 od 2404 poslanih (37,60% uspešnost). Django smo kasneje namestili še na lokalni strežnik WAMP in dosegli 100% uspešnost zahtev, vendar sta AVG in MAX časa bila v enakem rangu kot poprej. Rezultati prikazani na sliki 18, še ne pomenijo, da je v tem segmentu ogrodje Zend v prednosti pred drugima dvema. Pri testiranju nismo upoštevali nobenih specifičnih nastavitev ogrodij, kar se ob nameščanju aplikacij na produkcijski strežnik redko dogaja in bi se po vsej verjetnosti grafi zelo razlikovali.



Slika 18. Prikaz povprečnih in maksimalnih odzivnih časov na zahteve.

Ena od prednosti koncepta MVC je tudi hitrejši razvoj aplikacij, zato smo tudi na to temo naredili krajši test. Vzeli smo vse vrstice kode, ki se nahajajo v pogledih, modelih, kontrolerjih ter obrazcih, in jih združili v eno datoteko ter prešteli število vrstic, ki smo jih morali natipkati. Pri seštevanju smo poskušali čim bolj optimizirati kodo in izločiti nepotrebne vrstice. Naš projekt je bil relativno majhen, da bi lahko z zagotovostjo trdili, da lahko v enem okolju z manj kode, dosežemo enak učinek v nekem drugem okolju.

Ker je seštevek vrstic v vseh treh okoljih bil približno enak (Zend Framework – 247, Django – 255 in ASP.NET MVC – 231), smo se raje odločili, da v grafu predstavimo kolikšen del kode zavzema posamezen sklop v projektu (slika 19). Tu smo prišli do ugotovitve, da so se vsa tri okolja še najbolj ujemala v količini kode napisane v controller-ju, v vseh ostalih pa so bila precejšnja odstopanja.



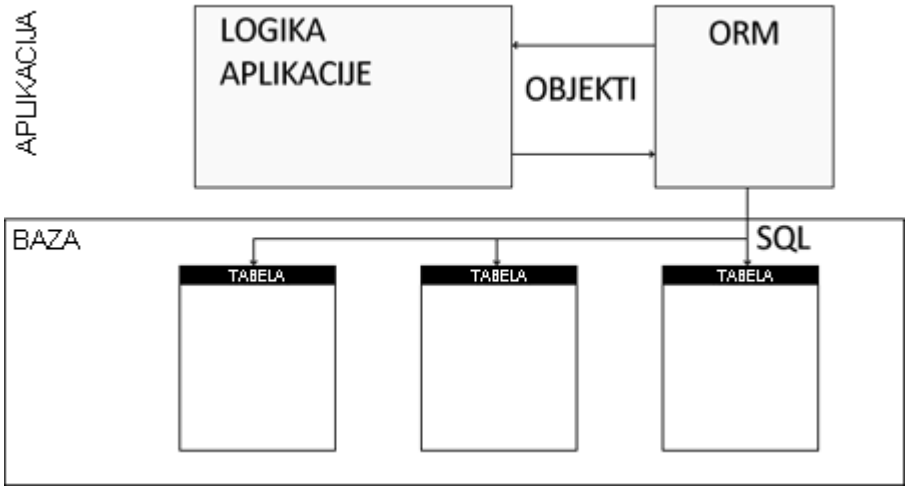
Slika 19. Deleži posameznih komponent v projektu.

4.4 Dopolnitev projekta z ORM

Pri abstrakciji podatkovnih baz (slika 20) v razrede se pri ASP.NET MVC-u, poleg že vključene tehnologije ADO.NET Entity Framework, ponuja rešitev NHibernate (vzorec Domain Model). Za Zend Framework pa je kar nekaj rešitev, poleg Zend_Db_Table, ki ga dobimo že zraven. Med njimi sta najbolj znana Propel in Doctrine. Prvi deluje po načinu Active Record slednji pa na način Table Data Gateway v navezi z vzorcem Table Module. Več o arhitekturnih vzorcih aplikacij [13]. V Django okolju je ORM že v osnovi implementiran in nimamo nikakršne možnosti posegati v samo strukturo ORM-ja.

Implementiranje kakršnekoli tehnologije v projekt povleče za seboj marsikaj dobrega, pa tudi slabega. Za vsako odločitev moramo pretehtati oboje, slabo in dobro, in se odločiti ali nam tehnologija bodisi prinese več dobrega bodisi več slabega.

ORM nam polega abstrakcije in neodvisnosti od sistema na katerem teče aplikacija, ponuja še migracijska orodja, s katerimi lahko vzdržujemo različice podatkovnih baz in se po potrebi vrnemo na katero od prejšnjih struktur pred trenutno strukturo baze, ponuja nam samodejno generiranje podatkovnih baz in relacij med njimi ne glede na različne specifike baz in kreiranje modelov na podlagi že zgrajenih baz.



Slika 20. Prikaz strukture ORM projekta.

5. SKLEP

S projektom in primerjavo okolij nismo skušali favorizirati katerega od uporabljenih orodij, naš cilj je bil predvsem raziskovalne narave. Cilj je bil ugotoviti katere možnosti nam katero okolje ponuja, doslednost pri implementaciji koncepta MVC, koliko je sama struktura projekta toga in ali se je da prilagoditi po lastnih željah. Izbrali smo tri okolja za katera smo menili, da v velikem obsegu predstavljajo tako platformo Microsoft, dediščino PHP ter okolje Django, kot predstavnika vse bolj priljubljenega jezika Python. S tem smo objeli tudi licenčna okolja in odprtokodne rešitve.

V primerjavah smo izpostavili teme, zaradi katerih se arhitekti morebiti odločajo za uvedbo koncepta MVC kot strukturo aplikacije. Poudarili smo sklope, kjer je mogoče pričakovati probleme npr. v primeru, ko je treba aplikacijo ponovno preanalizirati in ugotoviti ali je mogoče aplikacijo le deloma prirediti ali se je treba odločiti za ponovni razvoj ali ko se že odločimo za določeno tehnologijo, kakšne rešitve so že skladne z določenim razvojnim okoljem in ob morebitnih novo nastalih tehnologijah, v kolikšni meri bomo morali posegati v našo aplikacijo.

Pri reševanju programerskih težav se na spletu najde veliko literature v zvezi s samimi okolji. Za naše potrebe so bile več ali manj zadovoljive dokumentacije na uradnih straneh samih orodij. Nekatera specifična vprašanja pa smo reševali s pomočjo drugih preko spletnih portalov, ki so namenjeni diskusijam računalničarjev.

6. LITERATURA

[1] (2012) Model-View-Controller (MVC) Architecture. Dostopno na: http://www.jdl.co.uk/briefings/mvc.html

[2] (2012) The MVC pattern in theory and practice. Dostopno na: http://warp.povusers.org/programming/mvc.html

[3] (2012) Observer design pattern. Dostopno na: http://en.wikipedia.org/wiki/Observer_pattern

[4] (2012) Composite design pattern. Dostopno na: http://en.wikipedia.org/wiki/Composite_pattern

[5] (2012) Strategy design pattern. Dostopno na: http://en.wikipedia.org/wiki/Strategy_pattern

[6] (2012) Factory method design pattern. Dostopno na: http://en.wikipedia.org/wiki/Factory_method_pattern

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, »Design Patterns: Elements of Reusable Object-Oriented Software«, 1994

[8] (2012) Internet Information Services. Dostopno na: http://en.wikipedia.org/wiki/Internet_Information_Services

[9] (2012) Zend Technologies Ltd. Dostopno na: http://www.zend.com/en/

[10] (2012) ASP.NET. Dostopno na: http://en.wikipedia.org/wiki/Asp.net

[11] (2012) Django Software Foundation. Dostopno na: https://www.djangoproject.com/

[12] (2012) Django Sites. Dostopno na: http://www.djangosites.org

[13] M. Fowler, »Patterns of Enterprise Application Architecture«, 2002