UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Bogdan Fürst

Razvoj preprostega in cenovno ugodnega sistema za upravljanje spletnih vsebin

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM

RAČUNALNIŠTVA IN INFORMATIKE

Ljubljana, 2016

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Bogdan Fürst

Razvoj preprostega in cenovno ugodnega sistema za upravljanje spletnih vsebin

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM

RAČUNALNIŠTVA IN INFORMATIKE

MENTOR: doc. dr. Luka Šajn

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil *Microsoft Word 2016*.

Št. naloge:

Datum:

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

Kandidat: Bogdan Fürst

Naslov: Razvoj preprostega in cenovno ugodnega sistema za upravljanje spletnih vsebin

Vrste naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Delo naj predstavi celoten razvoj sistema za upravljanje vsebin, vse od sprejemanja zahtev naročnika pa do izvedbe. Sistem za upravljanje vsebin naj zajema spletno aplikacijo ter namizno nadzorno aplikacijo. Sistem naj bo razvit v čim krajšem času. Zahtevana je visoka prožnost rešitve, zanesljivost delovanja ter upravljanje s strani naročnika brez dodatnih stroškov. Prednost naj imajo odprte tehnologije.

Mentor: Dekan:

doc. dr. Luka Šajn prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Bogdan Fürst, z vpisno številko 63990191, sem avtor diplomskega dela z naslovom

Razvoj preprostega in cenovno ugodnega sistema za upravljanje spletnih vsebin

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal smostojno po mentorstvom doc. dr. Luke Šajna
- 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 DRI”

V Ljubljani, dne _____ Podpis avtorja:_____

Zahvaljujem se vsem zaposlenim na Fakulteti za računačništvo in informatiko v Ljubljani in še posebej mentorju doc. dr. Luki Šajnu.

KAZALO

TERMINOLOŠKI SLOVAR

Termini se uporabljajo tudi v praksi in po izkušnjah avtorja praktično brez izjeme.

NAROČNIK

Subjekt, ki krije stroške razvoja. Naročnik zahteva tudi materialne avtorske pravice za izvorno kodo. Izraz v pričujočem delu služi kot krovni termin za tri zagonska podjetja.

RAZVOJ REŠITVE

Termin označuje sklop aktivnosti znotraj časovnega obdobja, v katerem se analizira, načrtuje razvija, testira in namešča programsko opremo. Posamezna faza zajema izdelavo enega ali več sklopov funkcionalnosti (t.j. programskih modulov), katerih pravilno delovanje je pogojeno z že obstoječimi funkcionalnostmi iz prejšnjih faz. Pri izdelavi kompleksnejše programske opreme je razvoj praviloma deljen na več faz, s čimer zmanjšamo nezaupanje med naročnikom in izvajalcem. Finančni načrt se pripravi ločeno za vsako fazo. Za namene tega dela bomo razvoj razdelili zajeli v eno samo razvojno fazo.

SISTEM ZA VERZIONIRANJE

(ang: *revision control software*)

Sistem, ki omogoča upravljanje s spremembami razvojnih vsebin. Primeri nekaterih najbolj popularnih sistemov za verzioniranje: SVN, TFS, Git, TortoiseHg (ne zahteva centralnega strežnika).

CMS

Sistem za upravljanje (spletnih) vsebin [1] (ang. *(web) content management system*) je programska oprema, ki omogoča objavo, urejanje ter spremembo spletnih vsebin uporabnikom, ki niso vešči v uporabi spletnih tehnologij. Nekateri izmed najpopularnejših odprtokodnih CMS (ažuren seznam se nahaja na Wikipediji [1]): Wordpress, Joomla!, Drupal.

NADZORNA APLIKACIJA

Aplikacija, katere primarni namen je podpora upravljanju vsebin in spremljanju stanja sistema.

NADZORNIK

Skupni izraz za uporabnike, ki izvajajo vsaj eno od naslednjih aktivnosti:

- vstopajo v nadzorno aplikacijo,
- upravljajo vsebine z nadzorno aplikacijo.

PODATKOVNI MODEL

Abstrakcija, ki predstavlja relacije med podatki na človeku berljiv (vizualen) način (ang. *human-readable*). Za naše potrebe sta zanimivi predvsem dve orodji, ki omogočata zelo enostavno grafično ustvarjanje tako modela kakor baze na SQL strežniku: MySQL Workbench, Microsoft Visio.

SPLETNA APLIKACIJA

Aplikacija, do katere uporabniki dostopajo preko omrežja [2]. Pojma ne gre povsem enačiti s spletno stranjo [3], saj se spletna aplikacija z uporabniškega vidika obnaša podobno, kot se obnaša npr. Windows Forms aplikacija [4], le da se odjemalec izvaja v brskalniku, medtem ko spletna stran običajno služi predvsem predstavitvi vsebin na omrežju in ne zajema avtentikacije uporabnikov ali naprednejših uporabniških funkcionalnosti ipd. Youtube, na primer, je spletna aplikacija, MSDN pa spletna stran. Standardizirana definicija v času pisanja ne obstaja.

MVC

Model-View-Controller [5] je pristop k načrtovanju uporabniških vmesnikov, pogosto uporabljan za spletne odjemalce, predvsem tam, kjer se ne uporablja strežniške seje (ang. *session*) in kjer je funkcionalnosti uporabniškega vmesnika mogoče razdeliti na manjše in medsebojno neodvisne korake. MVC ogrodja so prisotna v vseh popularnejših jezikih, npr. PHP, C# (Microsoft MVC v okolju .NET). Poenostavi načrtovanje in izdelavo uporabniških vmesnikov glede na klasične pristope (npr. ASP.NET). MVC pristop je postal tudi aktualen pristop za načrtovanje spletnih vmesnikov. Prednost MVC pred ASP.NET je v hitrosti izvajanja in ločenosti predstavitve podatkov od logične komponente (ang. *controller*). Slabost MVC se pokaže pri implementaciji kompleksnih vnosnih mask uporabniškega vmesnika, kjer praviloma zahteva več dela kot ASP.NET, izvorna koda pa postane podobno zapletena.

POVZETEK

Naslov: Razvoj preprostega in cenovno ugodnega sistema za upravljanje spletnih vsebin

Namen naloge je dokumentirati način razmišljanja, tehnologije in odločitve, potrebne za razvoj programske opreme, ki posnema, ampak ne vkjučuje obstoječih rešitev na trgu. Naloga se sklicuje na realen projekt izdelave CMS, katerega razvoj je cenovno sprejemljiv za skupino manjših, vsebinsko sorodnih podjetij, ki želijo svoje izdelke tržiti na spletu, vendar jim obstoječe patforme na zadoščajo. Dokumentiran je celoten razvoj rešitve, od pogajanj z naročnikom do namestitve na produkcijski sistem. Sekundarni namen naloge je, da neizkušenega razvijalca programske opreme opozarja na tveganja, ki se pojavljajo pri izvedbi manjših projektov v zagonskih podjetjih (ang. *startup*), pa tudi večjih podjetjih. Prikazana rešitev je enostavno nadgradljiva, prenosljiva in uporabna kot podlaga za vsebinsko sorodne projekte, kar zagotovimo z uporabo okolij MVC in Forms.NET, trinivojsko arhitekturo rešitve ter sledenjem nekaterim priporočenim razvojnim smernicam podjetja Microsoft.

Ključne besede: CMS, C#, .NET, MVC, Windows Forms, MySQL, HTML, Javascript, Microsoft Visual Studio, Web application development, WebAPI.

ABSTRACT

Title: Development of a simple and inexspensive web content management system

The purpose of this thesis is to document rationalization, technology and descision making required to develop a software solution that mimics, but excludes existing solutions avaliable on market. Thesis refers to a real CMS development project, development of which is inexpensive enough for a group of smaller, content-related businesses that wish to sell their products on the web and cannot make use of existing systems for various reasons. Thesis encompasses the whole development process, ranging from negotiations with clients to final production system setup. Another goal of this thesis is to serve as a point of reference to an inexperienced software developer and warns of certain risks invoved with realization of smaller projects in startup (and general) businesses. Presented solution is easily upgradeable, platform independent and may be of use as a reference for content-related projects. Latter properties are acheived with 3-tier solution architecture, use of MVC and Forms.NET frameworks and following of some of Microsoft's application development guidelines.

Keywords: CMS, C#, .NET, MVC, Windows Forms, MySQL, HTML, Javascript, Microsoft Visual Studio, Web application development, WebAPI.

1 UVOD

Trg spletnih storitev je že pred časom dosegel točko, kjer vprašanje, ali je spletna storitev izvedljiva ali ne, skoraj ni več aktualno, odgovor je namreč praviloma pritrdilen. Na spletu je mogoče najti ogromno orodij in razvojnih knjižnic, ki razvijalcem praktično brez poznavanja delovanja (ang. *out of the box*) ter na visokem nivoju nudijo možnost razvoja in postavitve spletnih storitev, včasih celo samo s prilagoditvijo nastavitev (npr. Wordpress, Joomla, Wiki). Vendar pa naročniki rešitev z obstoječimi možnostmi dostikrat niso seznanjeni ali pa imajo želje, ki bi zahtevale poseg v samo ogrodje izbrane tehnologije (t.i. kršitev ang. *black box* pristopa), kar lahko pomeni nesorazmerno visoke stroške razvoja, lahko pa tudi nestabilnost osnovnih funkcionalnosti izbrane tehnologije. Prav tako trenutno vsaj na slovenskem trgu v obdobju 2012 -2015 zaradi ekonomske krize vladajo razmere, kje imajo naročniki razvojnih storitev nizko zaupanje v izvajalce, in obratno. Posledično sta zaupanje stranke in socialno mreženje, vzpostavljena v fazi pridobivanja posla, pogosto tudi najpomembnejša pogoja za uspešno pridobitev posla. Pomembno je, da se naročniku zagotovi čim boljše pogoje za zaupanje v izvajalca, da razvoj poteka gibko (agilno), t.j. da naročnik lahko spremlja potek razvoja v živo in da končni izdelek ni preobremenjen s stroški licenc, kjer to ni nujno potrebno ter da končni izdelek deluje brez napak. Zelo pomembna dejavnika sta tudi stroška kasnejših nadgradenj ter vzdrževanja. Še pred nekaj leti je bil največji problem tehnične zahteve sploh uresničiti, saj je na trgu vladalo pomanjkanje brezplačnih odprtokodnih rešitev ali pa so bile le-te preveč nestabilne za poslovno rabo. Danes pa se osredotočamo predvsem na sam izbor tehnologij, ki že same po sebi zagotavljajo rešitev za del problema, ki ga rešujemo, in k cilju pripeljejo s čim manj učenja, dela, posegov v jedro rešitve in posledično stroškov. Na trgu zaznavam rast povpraševanja po ožje specializiranih kadrih, razvijalci pa vedno težje stroškovno upravičijo odstopanje od že preverjenih pristopov in tehnologij. V pričujoči nalogi bom prikazal izvedbo projekta, ki skuša slediti zgornjim opažanjem in jih upoštevati glede na moje poznavanje tehnologij.

Primarni namen pričujočega dela je predstaviti razvojno iteracijo (fazo) spletne storitve, izdelane po naročilu. Gre za razvoj preprostega sistema za upravljanje spletnih vsebin, s katerim bomo izdelali tri spletne trgovine za tri podjetja, ki v nadaljevanju nastopajo pod skupnim terminom *naročnik*. Prikazali bomo najbolj osnovne principe pri načrtovanju programske opreme, kot tudi uporabo aktualnih tehnologij, preučili trg, prednosti in slabosti izbranih tehnologij ter izdelali končno rešitev po aktualnih razvojnih standardih. Prikazali bomo tudi visoko stopnjo neodvisnosti izbranih razvojnih pristopov od vsebinskih zahtev naročnika, kar pomeni, da je znanje, predstavljeno v nalogi, mogoče ponovno uporabiti skoraj povsem neodvisno od vsebinskih zahtev in kot podlago za manjše razvojne projekte.

V poglavju Osnove se bomo predstavili termine, potrebne za učinkovito razumevanje besedila. V nadaljevanju sledi predstavitev zahtev naročnika in obrazložitev izbora tehnologij, ki so bile predočene naročniku v okviru razpisa (C#, .NET Framework 4.0/4.5, MySQL). Pregled bo zajel tudi vsebinsko analizo, t.j. oceno obsega dela ter popis predvidenih odstopanj.

V poglavju Izvedba se bomo posvetili izvedbenemu delu razvojne iteracije, katere razvoj je trajal približno 30 delovnih dni. Pričeli bomo s postavitvijo razvojnega okolja, izbrali sistem za verzioniranje kode in dokumentov, definirali datotečno strukturo projekta in način za arhiviranje že opravljenega dela ter pričeli z razvojem v Visual Studiu. Nadaljevali bomo z analizo ponovne uporabe kode (ang. *code reusability*) in podatkovnih modelov iz drugih projektov. Prikazana bo visoka vrednost ponovne uporabe kode, saj bomo imeli vnaprej pripravljen del podatkovnega modela za prijavo uporabnikov ter uporabniški vmesnik skrbniške aplikacije. Nato pride na vrsto postavitev spletne aplikacije in preizkus koncepta. Ko je preizkus koncepta uspešno zaključen, nadaljujemo z razvojem posameznih naročnikovih zahtev in omogočimo spremljanje ter testiranje storitve kar v živo in med samim razvojem. Postavimo grob osnutek izgleda spletne strani in pripravimo skrbniško aplikacijo, spletno stran pa uporabljamo za testiranje delovanja. Ko zaključimo z razvojem skrbniške aplikacije, strani vizualno oblikujemo ter prilagodimo naročnikovim zahtevam. S tem zaključimo izvedbeni del razvojne iteracije.

V poglavju Sklepne ugotovitve bomo ocenili uspešnost izvedbe ter ugotavili, kje je prostor za izboljšave ter izpostavili glavne pomanjkljivosti načrtovanja in izvedbe.

2 PRIPRAVE

2.1 ZAHTEVE NAROČNIKA

Naročnik je startup podjetje z omejeno odgovornostjo (d.o.o.). Na sestanku predstavi srednjeročni cilj - razviti informacijsko podporo za trženje lastnih izdelkov, ki jih končni uporabnik uporabi in po uporabi vrne naročniku v analizo. Za namene trženja želi naročnik vzpostaviti CMS, s katerim lahko upravlja oseba, ki ni strokovnjak za vzdrževanje spletnih strani. Vsaj v okviru prve razvojne faze naj gre za preprosto spletno aplikacijo. Končni rezultat bo naročniku predstavljen kot izsledek analize izdelka. V naslednjih fazah želi naročnik taisti informacijski sistem razširiti v spletno skupnost, posvečeno analizi vsebinsko sorodnih izdelkov. Ta skupnost bi s časom postala tudi glavni vir prihodkov za naročnika. Cene naročnikovih izdelkov se gibljejo od 20 do 200, lahko pa tudi več evrov. Naročnik je prehodno postavil sistem za podporo izvajanju laboratorijskih analiz. Informacijski razvoj želi nadaljevati s CMS in postaviti več spletnih trgovin, po eno za vsako vpleteno pravno osebo oz. blagovno znamko. Naročnik se strinja s predlogom, da se v okviru razvoja CMS naprej izdela eno samo predlogo za spletno trgovino, nato pa se rešitev prilagodi še posebnostim, ki jih zahtevajo preostale pravneosebe. Rok za dostavo testne verzije naše rešitve je predvidoma 1 mesec.

2.2 PRIPRAVA PONUDBE

Analiziramo zahteve naročnika ter informacije, ki so nam na voljo. Takšna analiza nikoli ni dokončna in jo je vedno mogoče opraviti še bolj natančno. Prav tako jo je potrebno včasih dopolniti kar med sestankom ali telefonskim pogovorom z naročnikom, zato je priporočljivo pripraviti sistem vrednotenja lastnega dela.

2.2.1 ODLOČNOST NAROČNIKA, DA IZVEDE PROJEKT

Zanesljivost naročila je ključnega pomena, saj se veliko časa zapravi za razglabljanje s potencialnimi strankami, ki se pravzaprav samo informirajo in nimajo namena naročati izdelave programske opreme, saj so naročnika že izbrali in samo preverjajo, ali je cena, ki jo je zastavil izbrani izvajalec, previsoka. V našem primeru ima naročnik jasno vizijo, sistem za spremljanje analiz je že razvit in v fazi testiranja, kar za nas pomeni veliko verjetnost, da bo prišlo do naročila.

2.2.2 PLAČILNA SPOSOBNOST NAROČNIKA

Priporočljivo je preveriti, ali je naročnik sposoben finančnega bremena, ki ga prinaša izvedba (npr. AJPES, SISBON).

2.2.3 VSEBINA PONUDBE

Kaj naročnik pravzaprav želi in kaj bo želel v nadaljnjih fazah? Vse to je potrebno vsaj okvirno prepoznati iz zahtev naročnika, saj napačna zasnova rešitve lahko za nas pomeni visoke stroške ali celo izgubo naročil za potencialne nadaljnje faze razvoja:

- izdelava CMS,
- nadgradljivost CMS v spletno skupnost,
- občasno nadgrajevanje po zaključenem razvoju,
- povezljivost z informacijskim sistemom za izvajanje laboratorijskih analiz.

2.2.4 KONKURENČNE PONUDBE

Zelo hiter pregled platform, ki so na voljo, jasno pokaže, da je konkurenca tako na področju CMS kot tudi razvoja programske opreme po naročilu izjemno močna. Naša glavna prednost je, da naročnik ne zmore sam razviti zahtevane rešitve ter primerna cena, katero lahko izračunamo na podlagi cene ure dela v podjetjih, ki razvijajo rešitve s primerljivo kvaliteto. Tovrstne informacije je mogoče izluščiti iz cenikov, objavljenih na spletnih straneh teh podjetij ter podatkov, ki jih o teh podjetjih hrani Agencija Republike Slovenija za javnopravne evidence in storitve (AJPES). Za pravilen izračun je potrebnih nekaj let izkušenj s področja razvoja programske opreme.

2.3 IZBIRA PROGRAMSKEGA JEZIKA IN STREŽNIKA

Implicitne predpostavke, ki izhajajo iz avtorjevih izkušenj:

- naročniku je v interesu, da se za razvoj uporabi enake ali sorodne tehnologije, kot so bile uporabljene za razvoj sistema v podporo izvajanju analiz,
- v današnjem času specializirano poznavanje neke tehnologije s podporo spletne dokumentacije (Stack Overflow, MSDN,...) v večini primerov pretehta uporabnost primernejše, a nepoznane tehnologije.

Na ta način se poenostavi vzdrževanje v primeru, da se sodelovanje z naročnikom po zaključku faze prekine.

Prednost naj imajo torej:

- Izvajalno okolje Microsoft .NET, jezik C#,
- Sistem za upravljanje s podatkovnimi bazami MySQL.

2.3.1 C#/MS .NET FRAMEWORK 4+/IIS

C# je namenjen hitremu razvoju poslovnih aplikacij. Poznavanje jezika C# zahteva kodiranje po navodilih, ki jih je izdal Microsoft in katerih upoštevanje v okviru funkcionalnosti Code Analysis zagotavlja Visual Studio 2008 ali novejši. Koda bo posledično berljiva in enostavna za vzdrževanje. Okolje .NET nam skupaj z jezikom C# in Visual Studiem omogoča zelo hiter in udoben razvoj (RAD - Rapid application development). Okolje .NET zahteva licenco za operacijski sistem Windows (XP ali kasnejši).

Namestitev aplikacije na strežnik IIS je trivialna in ne zahteva veliko znanja in izkušenj.

Glavna slabost je strojna zahtevnost.

2.3.2 JAVA/ECLIPSE/JSP

Javansko okolje v navezi z Eclipse-om ponuja približno enako enostavnost razvoja kot okolje .NET. Glavna slabost je, da ne obstajajo standardi za kodiranje, jezik C# pa odpravlja nekatere največje pomanjkljivosti Jave [6]. Na voljo je veliko razvojnih knjižnic, vendar ne večinoma ne sledijo istim standardom kodiranja in poimenovanja, kar poveča porabljen čas za spoznavanje tehnologije. Glavni prednosti Jave sta razširjenost jezika in velika izbira brezplačnih knjižnic, vendar je v zadnjem času veliko teh knjižnic prirejenih tudi za okolje .NET.

2.3.3 C#/MONO /APACHE

Tehnologija združuje prednosti jezika C# in okolja MS .NET in Jave ter za povrh teče tako v okolju Windows, kot tudi Linux/Unix. Glavna slabost je, da v času nastanka razvoja Mono še ne premore implementacije specifikacije NET 4.5. Velika slabost je tudi nepreverjenost tehnologije, kar pomeni potencialne zaplete pri razvoju in lahko znatno upočasni razvoj. Okolje je sicer podprto tudi v Visual Studiu 2010 in novejših.

2.3.4 PHP/APACHE

Gre za popularno tehnologijo, ki na področju spletnih strani/aplikacij zlahka konkurira okolju C#/MS .NET. Podobno kot pri Javi tudi pri jeziku PHP obstaja preveč nestandardiziranih načinov za dosego istega cilja [7], jasnih razvojnih standardov ni zaslediti, nekatera popularnejša ogrodja so plačljiva. Posledično je tudi vzdrževanje zahtevnejše in dražje kot v okolju .NET, saj je verjetnost, da bi naročnik za namene vzdrževanja pridobil razvijalca s primernim znanjem, manjša.

2.3.5 RUBY

Mlada in popularna eksperimentalna tehnologija. Znotraj spletne skupnosti je v letu 2016 mogoče zaslediti namige, da se razvoj jezika zaustavi ali vsaj upočasni. Avtor nima izkušenj z uporabo okolja Ruby, kar pomeni, da je izbira precej tvegana.

2.3.6 IZBOR

Avtor se odloči za jezik C# in okolje Microsoft .NET 4, predvsem zato, ker ima s tega področja največ izkušenj. V prihodnosti bo razvoj mogoče prenesti na okolje Mono, s katerim lahko našo rešitev selimo na široko paleto Linux in Unix platform. Izvajalno okolje (ang. *language runtime*) Mono je brezplačno in odprtokodno.

Ker aktualna različica .NET Framework-a, 4.6, ni podprta v okolju Windows XP (ki je še vedno močno prisotno v poslovnem svetu), izberemo .NET Framework 4.0, za katerega je Microstoft jamčil delovanje v okolju Windows XP in s katerim ima avtor izkušnje na vseh novejših Windows platformah. Za razvoj spletne aplikacije bi lahko uporabili tudi .NET Framework 4.6, vendar bi s tem upočasnili morebitni prehod na Mono, saj razvijalci okolja Mono za implementacijo zadnje različice .NET specifikacije potrebujejo leto ali več od izida zadnje različice. Prehod iz okolja .NET na okolje Mono je po izkušnjah avtorja relativno enostaven in ne zahteva večjih prilagoditev kode v kolikor se za razvoj uporablja izključno .NET knjižnice, ki so implementirane v okolju Mono.

2.4 IZBIRA PODATKOVNE BAZE

Potrebovali bomo relacijsko bazo, ki nam omogoča osnovne SQL ukaze (SELECT, INSERT, UPDATE, DELETE) po standardu ACID [8], primarne ključe (ang. *primary key*), tuje ključe (ang. *foreign key*), transakcije (ang. *transaction*) in poglede (ang. *view*).

2.4.1 MS SQL SERVER/MS SQL SERVER EXPRESS

Strežnik je namenjen poslovni rabi in je precej robusten. Glavni slabosti sta visoka cena in zaprtost kode. Brezplačna verzija (Express) pa omejuje količino podatkov na 10Gb na instanco strežnika, kar bi za nas pomenilo kasnejšo zahtevo po prehodu na plačljivo verzijo ali pa prehod na katero drugo tehnologijo.

2.4.2 ORACLEDB/DB2

Široka in robustna sistema pogosto srečamo v poslovnih rešitvah. Tehnologiji sta zelo obsežni in presegata naše potrebe, cene licenc so previsoke.

2.4.3 MYSQL COMMUNITY SERVER

Strežnik MySQL je zelo razširjen, brezplačen, pogon InnoDB pa podpira transakcije, tuje ključe, indeksiranje (ang. *indexing*) in poglede, kar za naše potrebe povsem zadošča. MySQL je enostavno povezljiv z vsemi v tem delu naštetimi programskimi jeziki. Za povezavo z jeziki se uporablja namenske komponente za povezovanje (npr. MySQL .NET Connector). Po potrebi je MySQL povezljiv v gručo (ang. *cluster*). MySQL je licenciran z licenco GPL2 in nam dovoljuje uporabo v komercialne namene [9].

2.4.4 POSTGRESQL

Obsega in presega vse funkcionalnosti MySQL, vendar je sistem obsežnejši in je šele pred nekaj leti po uporabnosti začel na določenih področjih prehitevati MySQL. Na spletu so na voljo številne primerjave obeh okolij [10]. Licenca je PostgreSQL in nam, podobno kot MIT, omogoča skoraj neomejeno uporabo izvorne kode.

2.4.5 NOSQL

Okolja NoSQL ne podpirajo tujih ključev. Ker želimo, da bo naš podatkovni model preverjal vsebinsko smiselnost vnešenih podatkov na nivoju podatkovne baze, je uporaba tujih ključev obvezna, posledično NoSQL okolja niso zanimiva za namene te naloge.

2.4.6 IZBOR

Ker se bo pri delu predvidoma uporabljalo samo najosnovnejše funkcionalnosti relacijskih baz in ker imamo na voljo grafično orodje za načrtovanje relacijskih baz istega proizvajalca (MySQL Workbench), se avtor odloči za MySQL. Gre za preverjen odprtokodni izdelek s široko bazo uporabnikov in podporo povezljivosti z .NET okoljem. V kolikor bi pri razvoju naleteli na težave, bomo poskusili težave odpraviti s PostgreSQL.

2.5 IZBIRA ORODJA ZA NAČRTOVANJE PODATKOVNEGA MODELA

2.5.1 MYSQL WORKBENCH 6.X

Brezplačno odprtokodno orodje za načrtovanje relacijskih baz, ki kljub morju takšnih in drugačnih napak služi svojemu namenu. Omogoča tudi upravljanje z nastavitvami strežnika MySQL ter poganjanje SQL skript. Primerno je za izkušenejše uporabnike, ki jih občasno nedelovanje in izguba podatkov ne zmedeta preveč. Razvijalci orodja precej dosledno odpravljajo prijavljene napake. Izvorna koda orodja je v letu 2015 bila v fazi prestrukturiranja (ang. *refactoring*).

2.5.2 MICROSOFT VISIO

Razširjeno in preverjeno orodje za načrtovanje relacijskih baz za MS SQL Server. Glavna slabost glede na naše potrebe je visoka cena. Povezljivost z bazama MySQL in PostgreSQL je zagotovljena s podporo vmesniku ODBC [11]. Vmesnik ODBC sicer ne podpira nekaterih naprednih funkcij, kar bi nam lahko povzročilo težave pri uporabi.

2.5.3 IZBOR

Avtor se odloči za MySQL Workbench. Glavni razlogi za izbiro so:

- predhodna izbira sistema za upravljanje z relacijskimi bazami MySQL,
- MySQL Workbench je brezplačno in redno vzdrževano orodje.

2.6 IZBIRA SISTEMA ZA VERZIONIRANJE

2.6.1 VISUAL SVN SERVER/TORTOISE SVN CLIENT/WINMERGE

Na Subversionu (SVN) temelječa, učinkovita in preverjena kombinacija robustnih in brezplačnih orodij.

Strežnik Visual SVN Server je v osnovni različici na voljo brezplačno, dovoljena je komercialno raba [12]. Zajema preprost grafični vmesnik za urejanje nastavitev in upravljanje z odlagališči (ang. *repository*). Implementira osnovne funkcionalnosti odlagališč, kot so izločitve posameznih datotek, samodejno in rekurzivno zaznavanje sprememb na datotečnem sistemu, vtičniki (ang. *plugins*), ...

Orodje WinMerge je namenjeno poenotenju dveh različic iste datoteke (ang. *merge*). Podobnih brezplačnih orodij je na voljo še precej in boj ali manj odpravijo različne sklope težav, ki se pojavljajo pri verzioniranju.

2.6.2 TFS

Napredno in obsežno okolje za podporo timskemu delu. Omogoča npr. določanje stilskih ter vsebinskih pravil za pisanje programske kode (ang. *coding policy*) in upravljanje z razvojnimi opravili. Uporaba je smiselna v podjetjih, kjer mora pisanje programske kode (t.j. *kodiranje*) slediti natančno določenim pravilom ali standardom. TFS ponuja veliko več kot potrebujemo. Predvideni obseg našega dela je namreč precej majhen, do 20.000 vrstic programske kode, kar je, ob primerni porazdelitvi kode po datotekah, enostavno obvladljiva količina. Uvajanje pravil kodiranja je sicer smiselno, kadar obstaja zahteva naročnika, da se zagotovi čitljivost izvorne kode za poljubnega razvijalca, t.j. če bi z izvorno kodo rešitve kasneje uporabljale še tretje osebe. TFS predvsem zaradi plačljive licence postane učinkovita izbira šele, ko razvojna ekipa sestoji iz večih manjših skupin razvijalcev ali pa je fluktuacija sodelavcev na razvojnih projektih relativno visoka in je potrebno določiti jasna pravila kodiranja zaradi lažjega vzdrževanja in uvajanja novih sodelavcev.

2.6.3 TORTOISEHG/WINMERGE

Odprtokodna tehnologija, ki se postopoma uveljavlja. Glavna prednost je, da za delo ni potreben centralni strežnik, ampak je mogoče programsko kodo upravljati z združevanjem stanj dveh lokalnih odložišč (ang. *merge*). Uporabnika se nato dogovorita, katere spremembe imajo prednost. TortoiseHg je zelo primerna rešitev za razvoj programske opreme, katere koda ne sme uiti na splet, za naše potrebe pa je preokorna, saj je postopek združitve dveh vzporedno razvitih odložišč običajno precej zamuden. Združevanja doložišč pri našem delu predvidoma ne bo, saj bomo rešitev predvidoma v celoti izdelali sami.

2.6.4 IZBOR

Potrebujemo preprost, poceni in časovno učinkovit sistem za verzioniranje, namenjen eni osebi ali manjši ekipi ter dostop do programske kode preko spleta. Vse naštete tehnologije izpolnjujejo vse naše potrebe. Avtor izbere nabor orodij Visual SVN Server, Tortoise SVN client in WinMerge. Razlogi za izbiro so sledeči:

- avtor ima največ izkušenj s tem naborom orodij,
- SVN omogoča varen dostop (HTTPS) do programske kode preko spleta,
- preostali nabori tehnologij ne izboljšajo bistveno nobene od funkcionalnosti, ki jih bomo potrebovali.

2.7 IZBIRA INTEGRIRANEGA RAZVOJNEGA OKOLJA - IDE

2.7.1 ECLIPSE

Razširjeno, razširljivo, odprto in brezplačno okolje. Vtičniki omogočajo integracijo sistemov za verzioniranje. Priučevanje je za začetnika zaradi ogromne količine vtičnikov in javanskih knjižnic precej zahtevno. Ne podpira jezika C#, zato za naše razvojne potrebe ni primerno.

2.7.2 MONODEVELOP (XAMARIN)

Razširljivo, odprto in brezplačno orodje, ki omogoča platformno neodvisnot rešitev. Je primarno namenjeno razvoju v okolju Mono in jeziku C#. Orodje je postalo aktualno, ko so avtorji ogrodja Mono le-to prenesli tudi na razširjene mobilne platforme, npr. iOS in Android (Xamarin). Vtičniki omogočajo integracijo sistemov za verzioniranje. Okolje ponuja zadosten nabor orodij, ki jih programer v okolju .NET uporablja in potrebuje.

2.7.3 MICROSOFT VISUAL STUDIO COMMUNITY 2013

Gre za eno najnaprednejših in najpopularnejših orodij za razvoj poslovne programske probleme. Omogoča vse ali vsaj večino funkcionalnosti MonoDevelop-a, tudi razvoj z ogrodjem Mono. Orodje je standardna izbira pri razvoju na Microsoftovih tehnologijah.

2.7.4 IZBOR

Avtor izbere Visual Studio Community, ker ima z okoljem Visual Studio večletne izkušnje in ker je naprednejše od okolja MonoDevelop.

2.8 IZBIRA STROJNE OPREME

Podjetja prepogosto zanemarijo čas, ki se v razvoju izgubi zaradi neprimerne strojne opreme. Hitra trdi disk in CPE omogočata hitro delo v Visual Studiu, t.j. prevajanje kode v MSIL (ang. *Microsoft Intermediate Language*) ter odzivnost pomožnih procesov, ki tečejo v ozadju in programerju lajšajo pisanje kode.

Spodnjo mejo uporabnosti na podlagi avtorjevih izkušenj definiramo na sledeč način:

- poženemo ciljni sistem in
- v IDE odpremo obsežnejšo spletno stran (> 1000 vrstic interpretirane kode), spisano v kombinaciji jezikov HTML, Javascript, lahko tudi C#.

Če v IDE procesi, ki v ozadju validirajo kodo, izpišejo rezultate še preden taiste napake/neustreznosti odkrije programer sam, je sistem dovolj zmogljiv za razvijalca. Če je razvijalec hitrejši od teh procesov, ga bo sistem oviral pri delu.

2.9 PRIPRAVA OCENE DELA

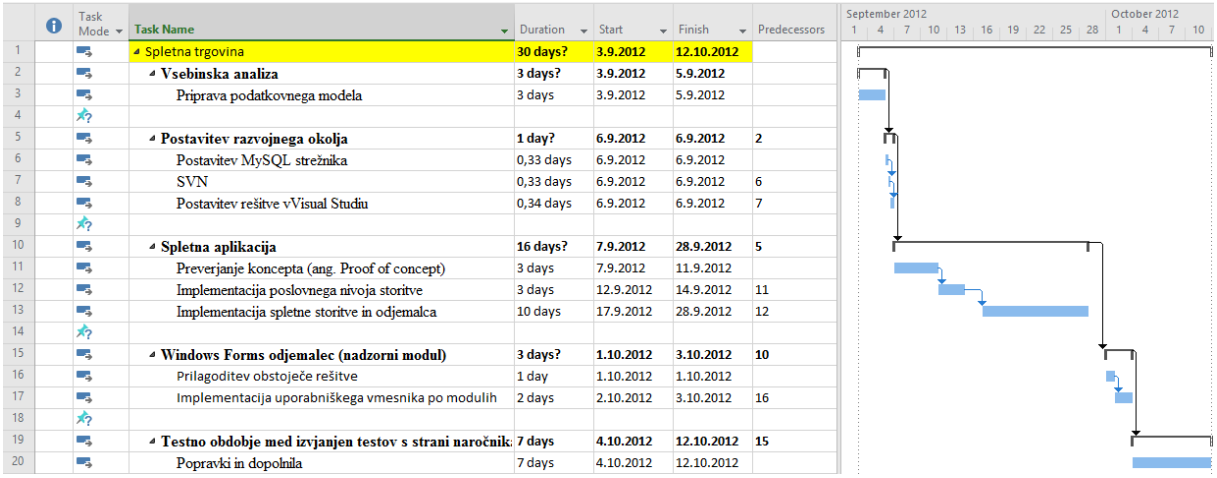
Izbrali smo jezik C# in orodje MySQL Workbench ter strežnik za relacijske baze MySQL Community Server. Kombinacija tehnologij je netipična. PHP se običajno uporablja v navezi z MySQL, C#/.NET pa v navezi s strežnikom MS SQL. Posledično moramo pričakovati vsaj kakšen zaplet pri uporabi. Izkaže se, da ponudniki spletnega gostovanja praviloma ne podpirajo povezave med .NET spletno aplikacijo ter instancami MySQL strežnika, kar pomeni, da bo imel naročnik dodaten strošek, ker bomo v okviru namestitve zahtevali bodisi namenski strežnik ali pa najem virtualnega okolja pri ponudnikih spletnega gostovanja.

Glede na poznavanje izbranih tehnologij imamo zdaj dovolj informacij, da sestavimo grobo oceno časovne zahtevnosti opravil, ki so pred nami. Seznam opravil variira glede na izbiro tehnologij, v našem primeru predvsem z izbiro jezika, saj bomo pri načrtovanju podatkovnega modela in postavitvi relacijske baze uporabili samo osnovne funkcionalnosti jezika SQL. Uporaba osnovnih SQL ukazov za razvijalca predstavlja približno enako časovno zahtevnost ne glede na implementacijo [13].

Razvoj bo zajemal naslednje sklope opravil:

- sestavljanje ponudbe,
- vsebinska analiza in načrtovanje,
- programiranje in testiranje,
- namestitev v produkcijsko okolje.

Predpostavimo, da bo naročnik imel vpogled v razvojno verzijo že pred začetkom testnega obdobja. Naročniku naj bo omogočeno tako testiranje kot podajanje pripomb; uporabili bomo t.i. agilno (ang. *agile*) razvojno metodologijo [14]. Ocena naj vsebuje seznam sklopov ali večjih podsklopov opravil in predviden obseg ur dela za vsako opravil. Ocene temeljijo na avtorjevih izkušnjah. V kolikor je bralec v določenem opravilu bolj ali manj izkušen od avtorja, naj oceno temu primerno relativizira glede. Za izračun relativizacijskega faktorja naj bralec primerja izmerjeno porabo časa z enim od preprostejših opravil (npr. postavitvi razvojnega okolja). Za pripravo ocene dela, iskanje povezav med delovnimi nalogami ter izdelavo časovnice avtor priporoča orodje Microsoft Project. Seznam opravil, vključenih v osnutek ocene, in časovnica sta prikazani na sliki *2.9-1*. V okviru preverjanja koncepta (ang. *proof of concept*) za nadzorno aplikacijo bomo izhajali iz izvorne kode že izdelane aplikacije za istega naročnika, zato je ocena znižana iz 10 *dni* na 3 *dni*.



	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		**Spletna trgovina**	**30 days?**	**3.9.2012**	**12.10.2012**	
2		**Vsebinska analiza**	**3 days?**	**3.9.2012**	**5.9.2012**	
3		Priprava podatkovnega modela	3 days	3.9.2012	5.9.2012	
4						
5		**Postavitev razvojnega okolja**	**1 day?**	**6.9.2012**	**6.9.2012**	**2**
6		Postavitev MySQL strežnika	0,33 days	6.9.2012	6.9.2012	
7		SVN	0,33 days	6.9.2012	6.9.2012	6
8		Postavitev rešitve vVisual Studiu	0,34 days	6.9.2012	6.9.2012	7
9						
10		**Spletna aplikacija**	**16 days?**	**7.9.2012**	**28.9.2012**	**5**
11		Preverjanje koncepta (ang. Proof of concept)	3 days	7.9.2012	11.9.2012	
12		Implementacija poslovnega nivoja storitve	3 days	12.9.2012	14.9.2012	11
13		Implementacija spletne storitve in odjemalca	10 days	17.9.2012	28.9.2012	12
14						
15		**Windows Forms odjemalec (nadzorni modul)**	**3 days?**	**1.10.2012**	**3.10.2012**	**10**
16		Prilagoditev obstoječe rešitve	1 day	1.10.2012	1.10.2012	
17		Implementacija uporabniškega vmesnika po modulih	2 days	2.10.2012	3.10.2012	16
18						
19		**Testno obdobje med izvjanjen testov s strani naročnik**	**7 days**	**4.10.2012**	**12.10.2012**	**15**
20		Popravki in dopolnila	7 days	4.10.2012	12.10.2012	

Slika 2.9-1: Seznam opravil in časovna ocena - Microsoft Project

Celoten obseg del je ocenjen na $T_r = 30$ delovnih dni. Naročnik oceno potrdi, prav tako potrdi okvirno ceno razvoja, ki izhaja iz ocene. Z naročnikom sklenemo dogovor, da v kolikor je naša ocena prenizka, dopolnimo manjkajoče dni na lastne stroške, naročnik pa dodatne zahteve doplača po ceni za uro dela, ki je razvidna iz ocene. S tem naročniku zagotovimo, da smo pravilno ocenili obseg dela. Strošek projekta C_r, ki sledi iz ocene pri urni postavki $P_h =$ 15 € za nas znaša:

$$C_r = T_r \times P_h = (30 \times 8h) \times 15\ €/h = 3600\ € \quad (1)$$

kar ustreza 1,5-kratniku povprečne mesečne bruto plače programerja. Ker gre za projekt po naročilu, katerega trženje je po končanem naročilu vprašljivo, je potrebno v ceno všteti še čas za pridobivanje novega naročila T_n podobne velikosti, kar v praksi pomeni minimalno en mesec aktivnosti. Prišteti je potrebno še potne in druge stroške C_t, ki bodo nastali pride delu in iskanju novih naročil. Stroški C_t naj zajemajo tudi predvidljivo izkušnjo, da naročniki dostikrat zahtevajo manjše popravke ali dopolnila, ker delajo napake pri definiranju zahtev.

Končna cena v ponudbi C_p naj torej znaša:

$$C_p = C_r + T_n \times P_h + C_t = (T_r + T_n) \times P_h + C_t \quad (2)$$

$$C_p = (T_r + T_n) \times P_h + C_t \quad (3)$$

$$C_p = (30 \times 8h + 30 \times 8h) \times 15\ €/h + C_t = 7200\ € \quad (4)$$

$$C_p = 7200\ € + C_t \quad (5)$$

3 RAZVOJ KONČNE REŠITVE

V nadaljevanju bomo prikazali, kako poteka ravojni proces programske opreme po naročilu in opozorili na dogodke in dileme, ki so v praksi praviloma prisotni pri projektno orientiranem razvoju.

3.1 POSVETOVANJE Z NAROČNIKOM

Da ugotovimo, kaj natančno naročnik želi od nas, je potrebno od naročnika zahtevati čim bolj natančen seznam funkcionalnosti (t.j. *funkcionalno specifikacijo*). Predvidljivost razvojne faze je v veliki meri odvisna od natančnosti popisa zahtev. Naročnik v našem primeru ni tehnično izobražena oseba in v kolikor spregledamo prikrite zahteve, bomo narobe ocenili zahtevnost razvojne faze, s tem pa pridelali odvečne stroške ali zapadli v nova pogajanja za povišanje cene celotne rešitve. Oboje je za nas zelo slabo, saj poveča možnost plačilne nedisciplne naročnika, morda celo odstop naročnika od pogodbe.

3.1.1 VSEBINSKE ZAHTEVE

Vsebinske zahteve za spletno stran, ki se izvaja na CMS, so naslednje:

- vstopna stran,
- novice,
- predstavitev izdelkov in storitev po kategorijah,
- spletno naročanje izdelkov, za nakup izbranega izdelka izdelka uporabnik ne sme porabiti več kot dveh klikov,
- predstavitev podjetja,
- kontakt,
- partnerji,
- pogosta vprašanja,
- kariera,
- pravna obvestila.

Naročnik na tej točki postavi zahtevo, da se v okviru CMS postavi tri spletne trgovine. Naročnik ugodi naši pripombi, da naj imajo strani identične funkcionalnosti in se razlikujejo le po barvni shemi in nekaterih fiksnih vsebinah (logotip, imena bližnjic), saj je sicer obseg dela močno povečan. Za nas to pomeni dodatna dva dni dela. Nova ocena torej znaša:

$$T'_r = T_r + 2 = 32\ dni \tag{6}$$

3.1.2 TEHNIČNE ZAHTEVE

Naročnik zahteva naslednje funkcionalnosti na sistemski ravni:

- vsebino sistema naj bo mogoče upravljati na daljavo,
- aplikacija za upravljanje naj bo Windows Forms aplikacija,
- z izdelano predlogo naj bo mogoče predstaviti tri po dejavnostih in izdelkih sorodna podjetja.

Naročnik se obveže, da bo pravočasno zagotovil primerno strojno opremo, ki obsega:

- nabavo srežnika (PC),
- nabavo licenčne programske opreme Microsoft Windows Server 2008R2,
- nabavo tipkovnice,
- nabavo računalniške miške,
- nabavo zaslona,
- najem močne povezave v internet

ali

- najem zasebnega virtualnega strežnika (ang. *VPS*).

Potencialna odstopanja pri izvedbi nabave strojne opreme na naše načrtovanje načeloma nimajo velikega vpliva. Pri načrtovanju je smiselno upoštevati vsaj:

- usposobljenost naročnika, da dobavi strojno opremo,
- možnost, da naročnik zamudi pri dobavi strojne opreme in to uporabi kot izgovor za neplačilo ali odlog plačila pogobenih obveznosti do izvajalca (nas); tovrstni primeri slabih poslovnih praks se dogajajo...

Pristop k reševanju scenarija 1:

- strojno opremo v imenu naročnika zagotovimo sami.

Pristop k reševanju scenarija 2:

- pogodbeno določilo, ki od naročnika terja plačilo na vnaprej določen datum,
- pogodbeno določilo, ki naročniku naloži dolžnost, da pozorno spremlja razvoj projekta in pravočasno opozori na vse morebitne probleme

3.1.3 OBLIKOVNE ZAHTEVE

Glede na naročnikov predlog, da grafično podobo izdela zunanji oblikovalec, upoštevamo možnost zamude pri dobavi:

- CMS predloga naj bo oblikovana minimalistično, po zgledu iStore,
- osnutek grafične podobe izdela zunanji sodelavec.

3.1.4 FINANČNI PARAMETRI

Naročnik posluje kot mlada (ang. *startup*) družba z omejeno odgovornostjo, t.j. d.o.o. Želijo cenovno ugodno informacijsko rešitev, na kateri bi gradili, v kolikor se primarna dejavnost primerno razširi.

Konkurenčnost izboljšujejo:

- izkušenje iz razvoja poslovnih informacijskih sistemov,
- nizka cena prve faze razvoja,
- pripravljenost na možnost nadaljnjega sodelovanja in sposobnost sprejemanja večje količine podobnih naročil.

Konkurenčnost nižajo:

- cena zahtevane strojne opreme (v kolikor je potrebna nabava nove),
- cena sistemske programske opreme,
- cena nekaterih plačljivih orodij, ki jih bomo uporabili za razvoj.

Pomembno je omeniti, da je struktura naročnikove družbe takšna, da upravljalec družbe nima neposrednega dostopa do finančnih sredstev. Gre za obliko družbe z omejeno odgovornostjo, d.o.o. Podjetje je šele v zagonu in sloni na nizko razpršenem produktnem portfelju. Obstaja realna možnost, da ne bodo zmožni pokriti pogodbenih obveznosti. Če podjetje preneha z delovanjem, bi bilo potrebno vložiti tožbo in črpati sredstva iz stečajne mase, ki pa je najverjetneje ne bi bilo, saj podjetje pridobiva izključno namenska sredstva, in sicer jih črpa iz krovne organizacije.

Naročnik nam zaupa, da konkurečnih ponudb še ni iskal, poiskal pa je referenčno, od katere smo po prvih ocenah precej ugodnejši. Kot referenco navaja že izdelane CMS-je, ki po mnenju naročnika:

- niso dovolj razširljivi (navadna spletna stran),

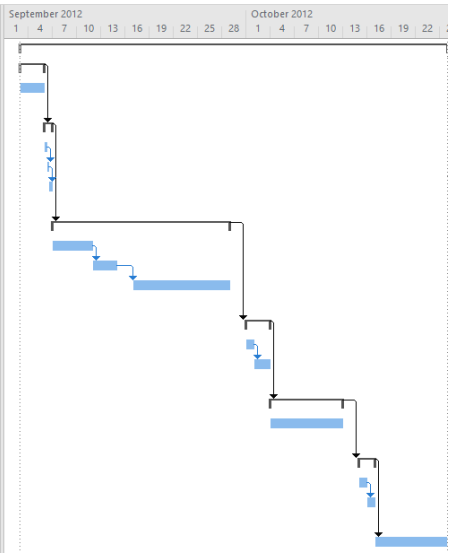
- so razširljivi, vendar nimajo dovolj nizkega zagonskega stroška (> 10.000 eur), npr. Microsoft Sharepoint,
- so razširljivi, vendar zahtevajo najem zunanjih sodelavcev, katerih skupna strokovnost je vprašljiva (PHP, WordPress) ali pa težko dosegljiva (Linux).

3.1.5 ČASOVNICA

Glede na ocenjen obseg dela in plan iz poglavja 2.9 določimo časovni okvir izvajanja projekta. Vhodni parametri za določanje nove časovnice so prikazani na sliki *2.9-2*:

- Datum pričetka izvajanja projekta $D_z = 1.9.2012$ (7)
- Ocenjen obseg dela v številu delovnih dni $T'_r = 32\ dni$ (8)
- Obdobje, namenjeno testiranju in manjšim prilagoditvam $T_t = 7\ dni$ (9)
- Razni nepredvideni dogodki $T_n = 20\% \approx 7\ dni$ (10)
- Vzdrževalna dela T_v (neznano) (11)

	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		Spletna trgovina	39 days?	3.9.2012	25.10.2012	
2		Vsebinska analiza	3 days?	3.9.2012	5.9.2012	
3		Priprava podatkovnega modela	3 days	3.9.2012	5.9.2012	
4						
5		Postavitev razvojnega okolja	1 day?	6.9.2012	6.9.2012	2
6		Postavitev MySQL strežnika	0,33 days	6.9.2012	6.9.2012	
7		SVN	0,33 days	6.9.2012	6.9.2012	6
8		Postavitev rešitve vVisual Studiu	0,34 days	6.9.2012	6.9.2012	7
9						
10		Spletna aplikacija	16 days?	7.9.2012	28.9.2012	5
11		Preverjanje koncepta (ang. Proof of concept)	3 days	7.9.2012	11.9.2012	
12		Implementacija poslovnega nivoja storitve	3 days	12.9.2012	14.9.2012	11
13		Implementacija spletne storitve in odjemalca	10 days	17.9.2012	28.9.2012	12
14						
15		Windows Forms odjemalec (nadzorni modul)	3 days?	1.10.2012	3.10.2012	10
16		Prilagoditev obstoječe rešitve	1 day	1.10.2012	1.10.2012	
17		Implementacija uporabniškega vmesnika po modulih	2 days	2.10.2012	3.10.2012	16
18						
19		Testno obdobje med izvjanjen testov s strani naročnik	7 days	4.10.2012	12.10.2012	15
20		Popravki in dopolnila	7 days	4.10.2012	12.10.2012	
21						
22		Spremembe glede na začetno oceno	2 days	15.10.2012	16.10.2012	19
23		Prilagoditve spletne trgovine za naročnika 2	1 day	15.10.2012	15.10.2012	
24		Prilagoditve spletne trgovine za naročnika 3	1 day	16.10.2012	16.10.2012	23
25						
26		Nepredvideni dogodki	7 days	17.10.2012	25.10.2012	22



Slika 3.1-1: Končna časovnica - Microsoft Project

Za obdobje testiranja T_t predvidimo, da se novih funkcionalnosti ne dodaja, ampak se izvaja manjše in časovno nezahtevne prilagoditve ter odpravlja napake, ki jih prijavi naročnik.

V praksi si naročniki programskih rešitev pogosto zadnji hip premislijo glede večjih funkcionalnosti in s tem povzročijo dodaten strošek sebi in/ali razvijalcu, zato naročnika opomnimo, da je potrebno čim bolj natančno definirati zahteve ter spremljati razvoj. Naročnik nato določi osebo, ki bo spremljala razvoj in testirala preizkusne različice rešitve (ang. *test version*).

Glede na vhodne parametre lahko zdaj izračunamo datum prevzema končnega izdelka, t.j. datum zaključka razvoja. D_p naj bo torej večji ali enak:

$$D_{pmin} \leq D_z + T'_r + T_t = 16.10.2012 \quad (12)$$

Ker iz prakse vemo, da se praviloma vedno pojavijo tudi nepredvideni dogodki, naj bo

$$D_p = D_{pmin} + T_n = 25.10.2012 \quad (13)$$

Realno torej ne bo mogoče zaključiti projekta pred 25.10.2012.

Naročnik časovnico potrdi in poda pripombo, da je pripravljen poravnati celoten znesek pogodbe tudi v kolikor je projekt zaključen predčasno. Obseg morebitnih vzdrževalnih del in dodatnih naročil je trenutno neznan, vendar lahko iz dejstva, da ima naročnik jasno razvojno vizijo, sklepamo, da bo prej ali slej potreboval razne podporne storitve.

3.1.6 IZVAJALČEVE PRIPOMBE K ZAHTEVAM

Ker je spletna stran precej preprosta in ker velja zakonitost, da nepreglednost spletne strani odvrača uporabnike od nakupa, naročniku predlagamo, da naj kot vstopna stran s predstavitvijo produktov služi kar domača stran trgovine, ostale strani pa naj bodo dosegljive preko povezav v glavi (ang. *header*) ali nogi (ang. *footer*) trgovine, s čimer uporabniku pri nakupu prihranimo dodaten klik. Naročnik privoli v poenostavitev. Glede na naročnikove zahteve bomo namesto splošnonamenskega CMS pripravljali sistem za upravljanje generičnih spletnih trgovin.

Glede na naročnikovo zahtevo po upravljanju sistema na daljavo priporočamo delo preko spletnega odjemalca, vendar se naročnik predloga ne sprejme, saj naj bi tak način upravljanja po naročnikovi oceni bil prezahteven za osebo, ki bo opravljala vlogo upravljalca z vsebinami (ang. *content manager*). Obenem pa je cena izdelave Windows Forms odjemalca po naši oceni nižja. Nadzorna aplikacija, s katero se bo upravljalo vsebine, bo nameščena na končnem številu prenosnih delovnih postaj. To pomeni majhno in končno število namestitev nadzorne aplikacije ter manj tehničnih težav. Predvidoma bo nameščanje nadzorne aplikacije temeljilo na pripravi in nameščanju namestitvenega paketa (ang. *setup* ali *installer*).

3.1.7 PODPIS POGODBE

Ker ocena dela iz faze posvetovanja z naročnikom ni bistveno drugačna kot ocena dela iz faze priprav, lahko z naročnikom sklenemo pogodbeno razmerje.

Ravnali bi pravilno, če bi po pogajanjih z naročnikom pripravili novo oceno, osnovano na podrobnejši vsebinski analizi in posodobili seznam opravil. Tudi vsebinsko analizo bi morali

naročniku zaračunati ločeno, vendar je obseg dela tako majhen, da odstopanja v količini dela ne bodo bistveno vplivala na ceno rešitve, zato pripravo podrobnejše ocene za zdaj izpustimo.

3.2 ARHITEKTURA REŠITVE

V osnovi bomo izhajali iz t.i. klasične arhitekture spletne aplikacije [15]. Rešitev naj sestoji iz:

- podatkovnega nivoja (ang. *data access layer* oz. *DAL*),
- poslovnega nivoja (ang. *business rules layer* oz. *BL*),
- spletne storitve (ang. *web service*), gostujoče na strežniku IIS, različici 7.5 ali novejši.
- predstavitvenega nivoja (ang. *presentation layer* oz. *user interface layer* oz. *UI*),

Predstavitveni naj nivo sestoji iz:

- ozkega (ang. *thin cient*) [16] spletnega odjemalca (ang. *web client*) [17], namenjenega v uporabo obiskovalcem strani,
- ozkega Windows Forms odjemalca (ang. *Windows desktop client*) [18], namenjenega skrbniku.

Rešitev torej sestoji iz Windows Forms namizne aplikacije in spletne aplikacije. Obe aplikaciji naj služita kot ločena predstavitvena nivoja za spletno storitev. Spletna storitev naj sloni na logiki, zajeti v poslovni nivo, le-ta pa za dostop do podatkov uporablja podatkovni nivo. Gre za uveljavljen pristop odjemalec-strežnik [19], kjer imamo na strežniški strani spletno storitev, ki sloni na protokolu HTTP, odjemalca pa v celoti slonita na programskem vmesniku (ang. *interface*), ki ga ponuja spletna storitev. Podrobnosti bomo pojasnili v poglavjih 3.5 - Spletna aplikacija ter 3.6 - Windows Forms odjemalec.

V skladu z ustaljeno prakso predpostavimo da se bo spletna aplikacija izvajala na namenskem strežniku. Odjemalca spletne aplikacije sta brskalnik, nameščenem na računalniku obiskovalca strani, in nadzorna aplikacija. Nadzorna aplikacija se bo izvajala v okolju Microsoft Windows, na delovni postaji nadzornika sistema.

3.3 VSEBINSKA ANALIZA IN PRIPRAVA PODATKOVNEGA MODELA

V okviru priprave podatkovnega modela opravimo osnovne analize podatkovnih struktur in namestimo orodja za obdelavo podatkov in načrtovanje relacijskih baz.

Namestimo orodji MySQL Workbench 6.2 in MySQL Server 5.6.22 Community Edition. Konfiguracijsko datoteko strežnika MySQL *C:\ProgramData\MySQL\MySQL Server 5.6\my.ini* dopolnimo za naslednjimi nastavitvami:

```
[mysqld]
# dovolimo poimenovanje tabel z veliko začetnico
lower_case_table_names=2
# nastavimo pot, kjer naj se hranijo podatkovne baze
datadir=D:/data_mysql
```

Poženemo orodje MySQL Workbench, odpremo predpripravljeno povezavo na lokalni MySQL strežnik ter preverimo, če strežnik teče. Če je strežnik aktiven, v razdelku Schemas odstranimo testno bazo in osvežimo pogled. Vrnemo se na začetno stran in ustvarimo nov podatkovni model s klikom na gumb <+>. Nato v prvem razdelku modela kliknemo ikono <+ *Add Diagram*>. Odpre se diagram podatkovnega modela. V prikazu iz orodne vrstice na levi dodamo tabelo, tabeli pa dodamo stolpec. Shranimo spremembe s <*Ctrl+S*>. V meniju *Database* izberemo možnost *Forward Engineer*, prikazane vnosne maske potrjujemo z gumbom <*Next*>, brez sprememb, vse do konca postopka. Ko je postopek zaključen, se vrnemo na zavihek s povezavo na strežnik in v levem oknu v razdelku *Schemas* osvežimo pogled. V pogledu bi se moral prikazati naš model s privzetim imenom *mydb*.

Izberemo razdelek *Users* and *Privileges* ter dodamo uporabnika z uporabniškim imenom *portal* (izpolnimo polji *Login Name* in *Password*). Uporabnika shranimo in v zavihku *Schema* privileges dodamo nov privilegij s klikom na <*Add Entry...*>. Obkljukamo *SELECT*, *INSERT*, *UPDATE*, *DELETE*, *LOCK TABLES*.

Ogrodje našega podatkovnega modela je s tem pripravljeno, nadaljujemo z analizo in implementacijo naročnikovih zahtev.

3.3.1 PRAVILA ZA POIMENOVANJE PODATKOVNIH STRUKTUR

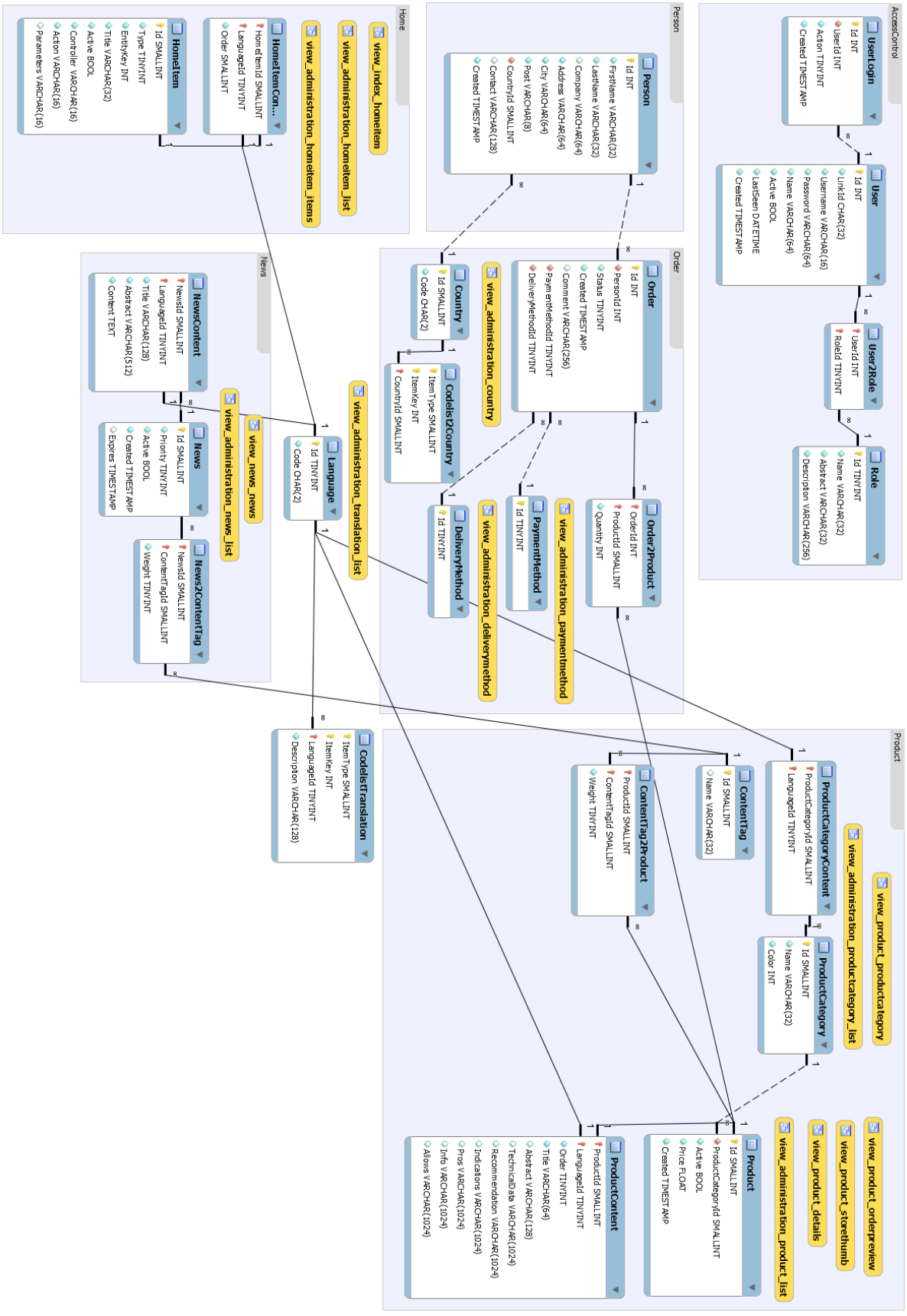
Doseči želimo čim večjo preglednost podatkovnih struktur in s tem lažjo morebitno nadgradnjo naše rešitve. Podatkovne strukture naj bodo poimenovane z nedvoumnimi pravili poimenovanja. Preglednost podatkovnih struktur zagotovimo s poimenovanji MySQL gradnikov (tabel, polj, relacij) po sledečih pravilih:

1. vsi primarni ključi tabel naj bodo

   1.1. poimenovani *Id*,
   1.2. celoštevilskega podatkovnega tipa števila (ang. *integer*): int, mediumint, smallint ali tinyint,
   1.3. samopovečevalni (ang. *auto-incremented*, *AI*) - vsakemu novemu zapisu v tabeli bo samodejno dodeljena zaporedna številka,
2. vsa poimenovanja so v angleškem jeziku,
3. vsa poimenovanja so opisna, v ednini, in skušajo s čim manj besedami nakazati namenskost uporabe (npr.: tabela uporabnikov naj se imenuje *User*, šifrant uporabniških vlog naj se imenuje *Role*),
4. povezovalne tabele naj imajo imena oblike {0}2{1} (npr. *User2Role*), kjer je
   4.1. {0} - ime podrejene (ang. *child*) tabele,
   4.2. 2 - znak, ki se prebere "to" (sl. *od*),
   4.3. {1} - ime nadrejene (ang. *parent*) tabele.
5. Vsaka tabela, ki vsebuje uporabniške vsebine, naj ima polje, poimenovano *Created* (sl. *ustvarjeno*), ki ima privzeto vrednost (ang. *default value*) nastavljeno na *CURRENT_TIMESTAMP*, kar pomeni da se polje samodejno izpolni s trenutnim datumom in časom vsakič, ko se v tabeli ustvari nova vrstica.

Zakaj črka '2' v poimenovanju povezovalnih tabel? Če je ni, postane poimenovanje dvostranskih relacij dvoumno. Primer: definiramo tabele *News* (sl. *novice*), *NewsContent* (sl. *vsebina novice*) in *ContentTag* (sl. *vsebinski zaznamek*). Vsaka novica ima vsebino, vsebina pa vsebinske zaznamke. Obstajajo pa tudi vsebine, ki niso del novice, ampak neke druge podatkovne strukture. Če ustvarimo tabelo z imenom *NewsContentTag*, bi bralec verjetno sklepal, da gre za tabelo vsebinskih zaznamkov za novice. Na vsebinske zaznamke se bodo sklicevale tudi druge spletne vsebine, ne samo novice. Upoštevajoč naša pravila poimenovanja bi bralec smel sklepati, da gre, vsebinsko, za tabelo *New2ContentTag* ali *NewContent2Tag*. Če pa tabelo poimenujemo *News2ContentTag*, je jasno razvidno, da gre za povezovalno tabelo med novicami in vsebinskimi zaznamki.

Končna oblika podatkovnega modela je prikazana na sliki *2.3-1*.



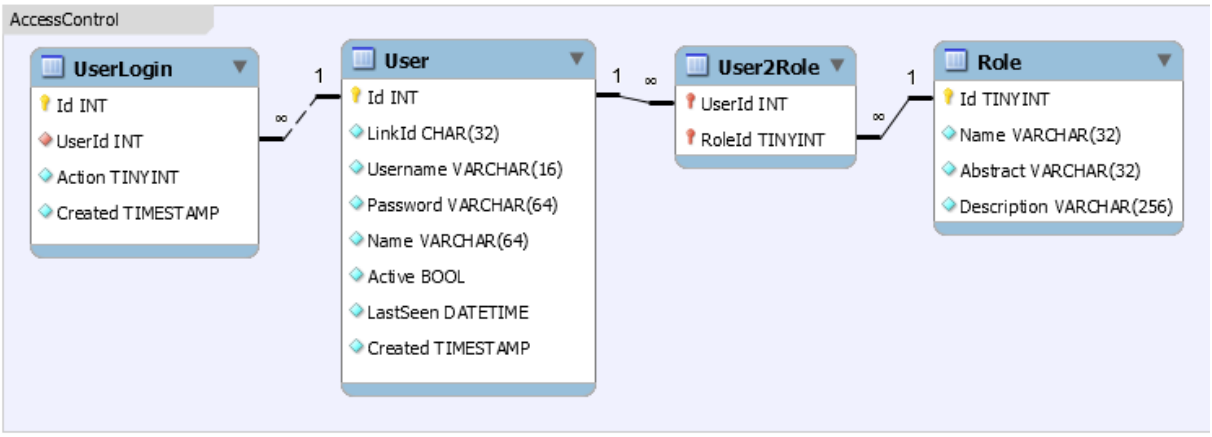
Slika 3.3-1: Podatkovni model

3.3.2 PODATKOVNE STRUKTURE NADZORNE APLIKACIJE

Nadzorna aplikacija v osnovi zajema upravljanje z uporabniki (ang. *user management*), v naši rešitvi pa bo nadzornik upravljal tudi s spletnimi vsebinami (ang. *content management*).

Potrebujemo torej vsaj sledeče tabele (glej sliko *2.3-2*):

- tabela uporabnikov, ki vstopajo v sistem s prijavo, *User*,
- tabela vlog uporabnikov *Role*,
- povezovalna tabela vlog in uporabnikov *User2Role*.



Slika 3.3-2: Podatkovne strukture nadornega modula

3.3.2.1 VLOGE UPORABNIKOV

Vsaka zapis v tabeli vlog naj vsebuje unikaten (ang. *unique key*) identifikator vloge *Id*, naziv vloge, kratek povzetek, namenjen končnim uporabnikom za lažje razumevanje pomena vloge ter daljši opis, ki vsebuje celoten popis funkcionalnosti, ki pritičejo vlogi.

Od naročnika pridobimo informacijo o tem, kdo vse se bo prijavljal v sistem in kakšna bo vloga teh oseb v sistemu. Naročnik odgovori naslednje:

- za tehnično vzdrževanje sistema bo zadolžena ena sama oseba (skrbnik),
- druga oseba (upravljalec vsebin) bo zadolžena za upravljanje z vsebinami in za izdelavo statistik, vendar ta oseba ni tehnično izobražena, zato naj bo nadzorna aplikacija prilagojena tehnično neveščemu uporabniku,
- tretja oseba (nadzornik) bo preverjala delo skrbnika in upravljalca vsebin, vendar v sam sistem naj ne bi aktivno posegala, delo bodo ocenjevali na podlagi javno dostopnega dela sistema,
- vse zgoraj naštete osebe so zaposlene v istem podjetju.

Avtor na podlagi izkušenj sklepa, da bodo nekoč tudi nadzorniki želeli vstopati v sistem in aktivno spremljati delovanje sistema s pomočjo poročil o delovanju ter upravljati z vsebinami.

Iz zgoraj naštetega je razvidno, da potrebujemo naslednje uporabniške vloge:

- skrbnik sistema (ang. *administrator*): tehnično vzdrževanje, ne upravlja vsebin,
- skrbnik vsebin (ang. *content manager*) - upravljanje vsebin,
- nadzornik (ang. *supervizor*) - upravljanje vsebin, upravljanje s poročili.

Imamo dovolj informacij, da začrtamo vloge uporabnikov. Ker v naročilu poročila niso bila predvidena, potrebujemo eno samo vlogo, ki jo poimenujemo

- skrbnik sistema, *Administrator* (v nadaljevanju *nadzornik*).

3.3.2.2 UPORABNIKI

Vsak zapis v tabeli *User* naj ima uporabniško ime *Username*, z algoritmom SHA 256 kriptirano geslo *Password*, ime *Name*, indikator aktivnosti *Active*, datum zadnjega vstopa v sistem *LastSeen*, datum registracija uporabnika *Created* ter unikatno ime spletne dostopne točke *LinkId*. *LinkId* je unikatni ključ, ki ga bomo pripeli spletnemu naslovu, preko katerega bo uporabnik vstopal v sistem. Več o *LinkId* bomo povedali v okviru priprave windows Forms odjemalca v nadaljevanju.

Tabela *User2Role* hrani povezave med uporabniki in vlogami, t.j. katere vloge so dodeljene posameznemu uporabniku.

Tabela *UserLogin* hrani podatke o aktivnosti uporabnika v sistemu.

3.3.2.3 ŠIFRANT VSEBINSKIH ZAZNAMKOV

Šifrant *ContentTag* vsebuje vsebinske zaznamke, ki bodo služili avtomatizirani klasifikaciji vsebin. Primer takšne klasifikacije je iskanje po kategoriji izdelka, kjer uporabnik spletne strani želi najti vse izdelke, ki so uvrščeni v kategoriji *Welness* in *Sport*.

Skrbnik naj ima možnost urejanja šifranta.

3.3.2.4 ŠIFRANT JEZIKOV

Šifrant *Language* vsebuje identifikator *Id* in šifro jezika *Code* (ang. c*ode*).

Šifrant jezikov definira naročnik.

Sistem naj podpira poljubno število jezikov, z vidika naročnika pa naj bo število jezikov za zdaj fiksno, saj podatkovnega modela še ne poznamo dokončno in je povsem mogoče, da bo v

okviru podpore novemu jeziku potrebno na naše stroške prevajati vsebine (npr. šifrant držav) oz. zagotavljati druge kulturno pogojene zahteve končnih uporabnikov.

Skrbnik nima možnost urejanja šifranta.

3.3.2.5 ŠIFRANT PREVODOV ŠIFER

Šifrant prevodov šifer *CodelistTranslation* vsebuje prevode šifer drugih šifrantov.

Vsak prevod sestoji iz vrste šifre *ItemType*, identifikatorja šifre *ItemKey*, identifikator jezika *LanguageId* ter prevoda *Description*.

Skrbnik naj ima možnost urejanja šifranta, vendar zgolj na mestu, kjer se ureja šifra, ki ji prevod pripada. Ob vnosu ali urejanju posamezne šifre naj se od skrbnika zahteva prevode za vse jezike iz šifranta jezikov.

3.3.2.6 ŠIFRANT DRŽAV

Šifrant držav vsebuje identifikator *Id* in šifro države *Code*.

Ker gre za šifrant, ki bo prikazan končnim uporabnikom, ga je potrebno prevesti v vse jezike iz šifranta jezikov.

Skrbnik naj ima možnost urejanja šifranta.

Pogledi:

- *view_administration_country*: prikaz seznama držav v Windows Forms odjemalcu.

3.3.3 PODATKOVNE STRUKTURE NOVICE

Naročnik želi prikazati novico v večih jezikih, prav tako želi možnost oblikovanja vsebine posamezne novice. Z novicami bo upravljal skrbnik sistema.

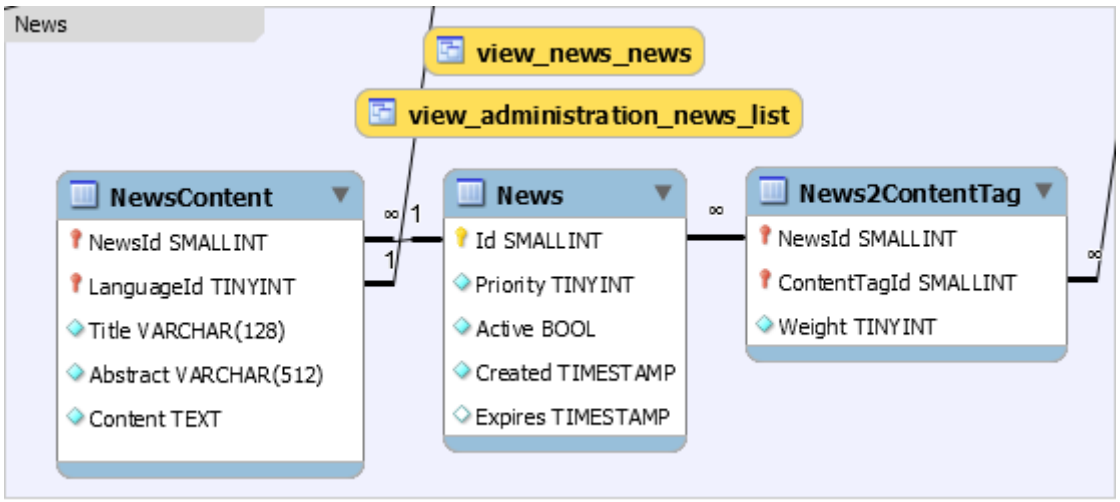
Posamezna novica sestoji iz zapisa v tabeli *News*, enega ali več zapisov v tabeli *NewsContent* ter enega ali več zapisov v tabeli *News2ContentTag*.

Zapis v tabeli *News* vsebuje identifikator novice *Id*, prikazno prioriteto *Priority*, indikator aktivnosti *Active*, datum stvaritve *Created* in datum poteka *Expires*. Novici mora skrbnik definirati še naslov *Title*, povzetek *Abstract* in vsebino *Content* za enega ali več jezikov iz šifranta jezikov.

Vsaki novici naj bo mogoče dodeliti enega ali več zapisov iz šifranta vsebinskih zaznamkov, povezave se hranijo v tabeli *News2ContentTag*. Predvidena je tudi mehka pripadnost novice posameznemu vsebinskemu zaznamku *Weight*.

Pogledi:

- *view_news_news*: prikaz novic na spletni strani,
- *view_administration_news_list*: prikaz seznama novic v Windows Forms odjemalcu.



Slika 3.3-3: Podatkovne strukture novice

3.3.4 PODATKOVNE STRUKTURE IZDELKA

Naročnik želi prikazati izdelek v večih jezikih, prav tako želi možnost oblikovanja vsebine posameznega izdelka. Z izdelki bo upravljal skrbnik sistema. Naročnik v tej fazi še ni zmožen natančno določiti, kateri podatki so potrebni za predstavitev izdelka ali kategroije izdelkov, zato v podatkovni model zajamemo vse na sestanku omenjene podatke in predpostavimo, da bo kasneje prišlo do sprememb.

Posamezen izdelek naj sestoji iz zapisa v tabeli *Product*, enega ali več zapisov v tabeli *ProductContent* ter enega ali več zapisov v tabeli *ContentTag2Product*.

Zapis v tabeli Product vsebuje identifikator izdelka *Id*, identifikator kategorije *ProductCategoryId*, indikator aktivnosti *Active*, ceno *Price* in datum stvaritve *Created*. Izdelku mora skrbnik definirati še:

- vrstni red za prikaz *Order*,
- naslov *Title*,
- povzetek *Abstract*,
- tehnične podatke *TechnicalData*,
- priporočila *Recommendation*,
- indikacije, kdaj je izdelek za uporabnika zanimiv (*Indications*) ,
- vsebino *Content* za enega ali več jezikov iz šifranta jezikov,
- *slika izdelka,
- *slika za predogled,
- *priponke (ang. a*ttachment*) .

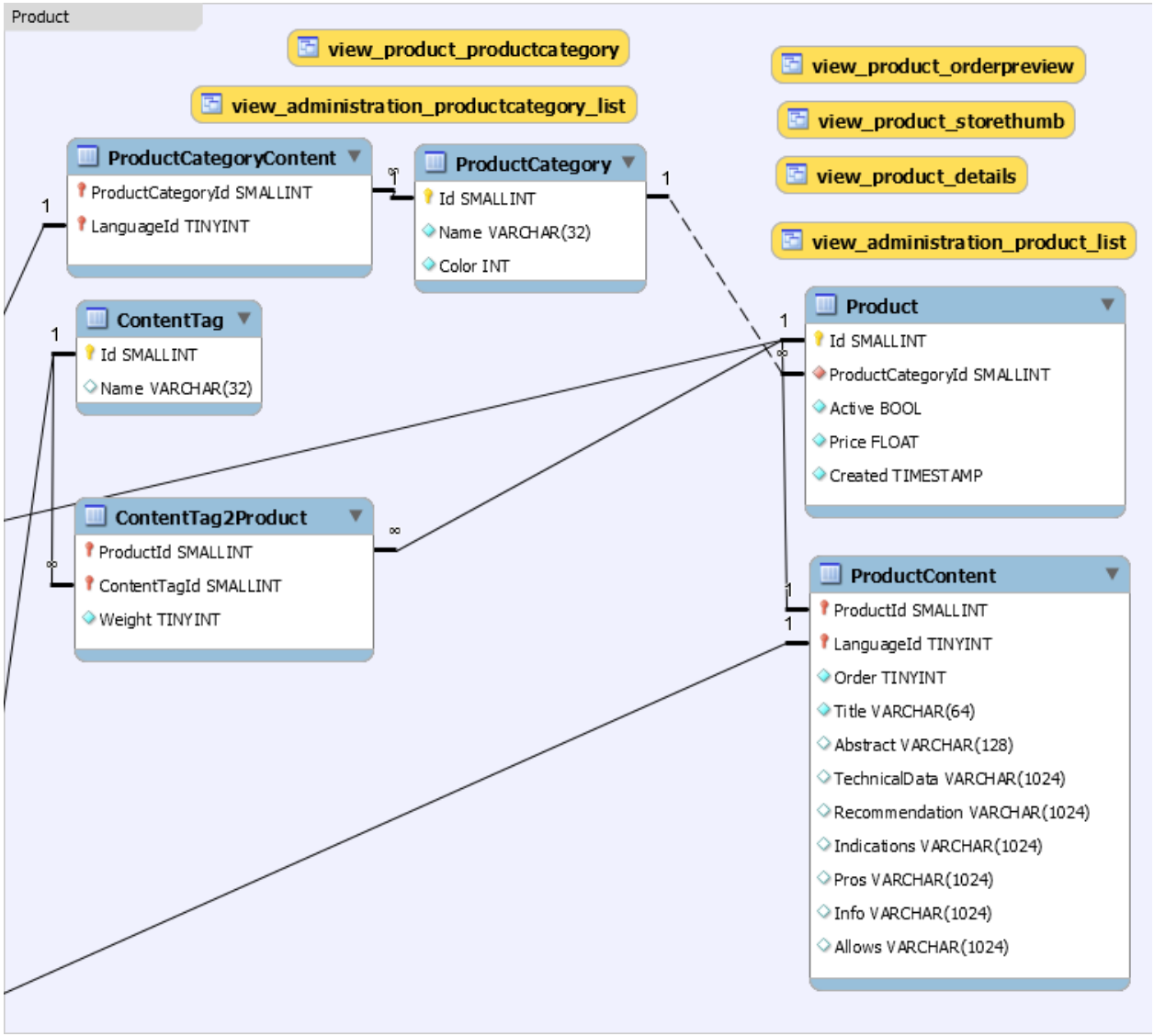
* Te podatke bi sicer lahko hranili v podatkovni bazi, vendar je zas to problematično, saj se ne moremo zanesti, da skrbnik ne bo vnašal ogromnih datotek, ki lahko v hipu upočasnijo delovanje strežnika in močno povečajo porabo pomnilnika ali trdega diska s strani podatkovne baze. Zato bomo slike in priponke hranili neposredno na datotečnemu sistemu kot enolično poimenovane datoteke.

Vsakemu izdelku naj bo mogoče dodeliti enega ali več zapisov iz šifranta vsebinskih zaznamkov, povezave se hranijo v tabeli *ContentTag2Product*. Predvidena je tudi mehka pripadnost izdelka posameznemu vsebinskemu zaznamku Weight.

Pogledi:

- *view_product_productcategory*: vse kategorije v vseh jezikih,
- *view_product_storethumb*: prikaz seznama izdelkov izbrane kategorije na spletni strani,

- *view_product_details*: prikaz podrobnosti izdelka na spletni strani,
- *view_administration_productcategory_list*: prikaz seznama kategorij izdelkov v Windows Forms odjemalcu,
- *view_administration_product_list*: prikaz seznama izdelkov v Windows Forms odjemalcu.



Slika 3.3-4: Podatkovne strukture izdelka

3.3.5 PODATKOVNE STRUKTURE NAROČILA

Naročnik želi sprejemati spletna naročila s prijavo kupca v sistem, s plačilom po povzetju in dostavo po pošti v države EU. V eni od naslednjih faz želi naročnik omogočiti tudi plačevanje preko spleta in uveljavljanje ugodnosti, kot so akcije in popusti ter dovoliti prodajo v države izven EU. Naročniku pojasnimo, da je spletno naročanje povsem izvedljivo tudi brez prijave uporabnika v sistem, saj kupec sporoči osebne podatke in elektronski naslov, na katerega se lahko pošlje potrditveno sporočilo. V najslabšem primeru se kupec odpove pošiljki in naročnik nosi stroške poštne pošiljke. Naročnik se s poenostavitvijo strinja in oceni, da poenostavljeno naročanje zanj pomeni večjo ali enako prodajo.

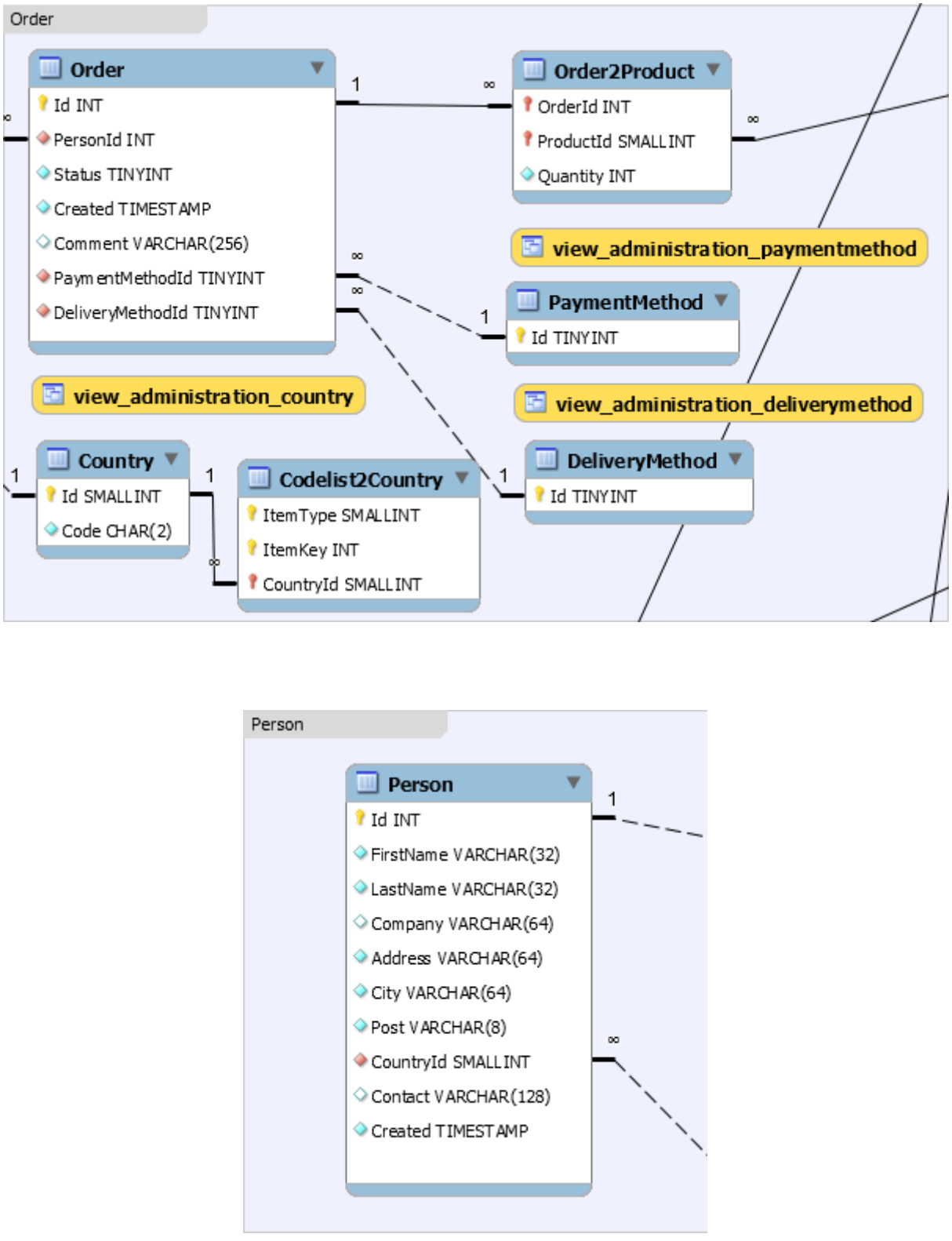
Posamezno naročilo naj sestoji iz zapisa v tabeli *Order*, zapisa v tabeli *Person* ter enega ali več zapisov v tabeli *Order2Product*.

Zapis v tabeli *Order* vsebuje identifikator naročila *Id*, identifikator osebe *PersonId*, identifikator stanja naročila *Status*, identifikator *ProductCategoryId*, datum stvaritve zapisa *Created*, neobveznega komentarja kupca, metode plačila (vedno po povzetju) ter metode dostave (vedno po pošti).

Tabela *Person* zajema podatke o kupcih, ki so lahko pravne ali fizične osebe. Ob naročilu naj se zahteva ime, priimek in naslov, kontaktne podatke ter naziv podjetja.

Pogledi:

- *view_administration_order_list*: prikaz seznama naročil v Windows Forms odjemalcu,
- *view_administration_order_details*: prikaz podrobnosti naročila v Windows Forms odjemalcu.



Slika 3.3-5: Podatkovne strukture naročila

3.3.6 PODATKOVNE STRUKTURE DOMAČE STRANI

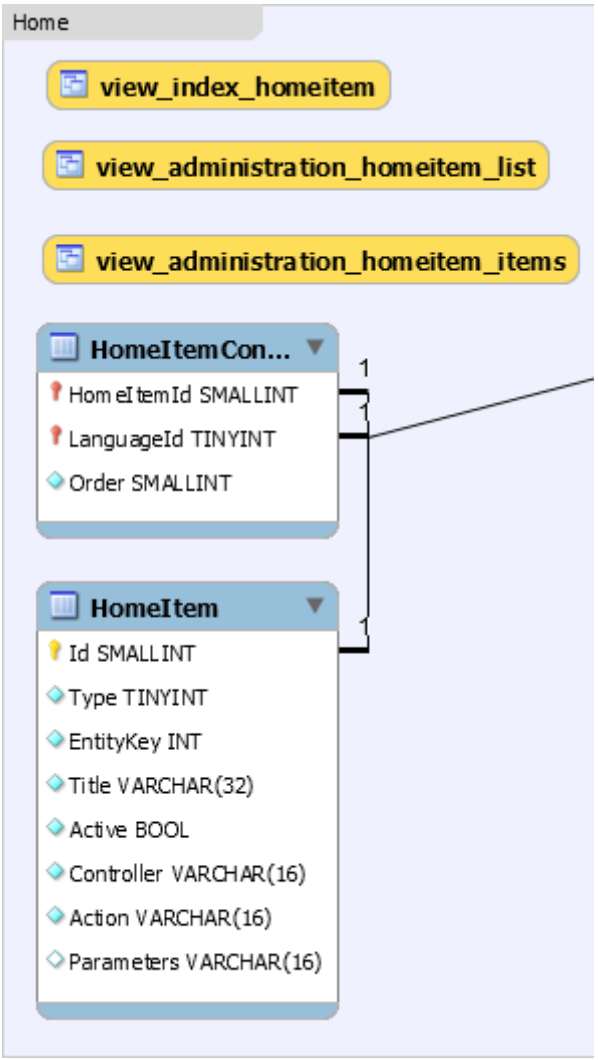
Naročnik želi na domači strani portala prikazati povezave na vsebine, ki so sicer prikazane drugje na portalu. Namen teh vsebin je vzbuditi pozornost in opozoriti na nove izdelke in pomembne informacije, zato naročnik zahteva, da se v prvi razvojni fazi na domači strani omogoči prikazovanje novic, kategorij in izdelkov.

Posamezna povezava sestoji iz zapisa v tabeli *HomeItem*, enega ali več zapisov v tabeli *HomeItemContent*.

Zapis v tabeli *HomeItem* vsebuje identifikator povezave *Id*, tip povezave *Type*, ključ entitete, na katero kaže povezava *EntityKey* (novica, izdelek, ...), naslov *Title*. Polja *Controller*, *Action* in *Parameters* v začetku niso bila predvidena in so zamenjala polje naslov (ang. *universal resource identifier*, *URI*). Določajo obliko zahteve, s katero spletni odjemalec pridobi entiteto s strežnika. Da preprečimo skrbniku dodajanje poljubnih povezav in s tem odpremo možnost napada na obiskovalce spletne strani, spletni naslov vedno tvorimo dinamično iz korena, krmilnika, akcije in parametrov.

Pogledi:

- *view_index_homeitem*: prikaz povezav na spletni strani,
- *view_administration_homeitem_list*: prikaz seznama povezav v Windows Forms odjemalcu,
- *view_administration_homeitem_items*: vse možne povezave na vsebine v vseh jezikih – uporabimo na uporabniščem vmesniku, kjer izbiramo, na kaj naj kaže povezava.



Slika 3.3-6: Podatkovne strukture domače strani

3.3.6.1 PRIMER POVEZAVE NA IZDELEK

Primer vrednosti stolpcev zapisa v tabeli *HomeItem*:

- Koren: *http://localhost/Portal.Web/*
- Krmilnik: *Product*
- Akcija: *Store*
- Parameter (kategorija in identifikator izdelka): *17?ProductId=2*

Sestavljena povezava izgleda tako:
http://localhost/Portal.Web/Product/Store/17?productId=2

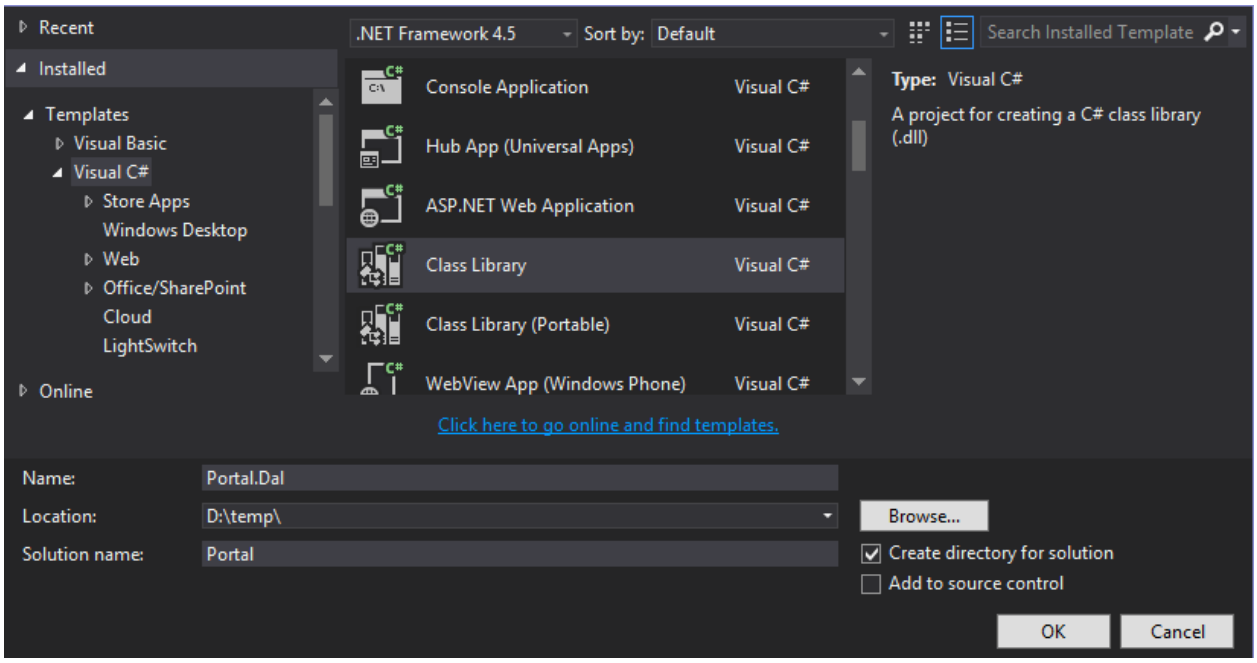
3.3.6.2 PRIMER POGLEDA

Spodnja skripta MySQL ustvari v bazi pogled (ang. *view*), ki poišče vse vsebine, na katere lahko ustvarimo povezavo na domači strani, in jih pretvori v zapise *HomeItem*:

```
CREATE VIEW `portal`.`view_administration_homeitem_items` AS
SELECT
   pc.`Id` AS `EntityKey`
        , 'Product' AS `Controller`
        , 'Store' AS `Action`
        , pc.`Id` AS `Parameters`
        ,l.`Id` AS `LanguageId`
        ,1 AS `Active`
        ,CONCAT('Kategorija izdelkov: ', pc.`Name`) AS `ItemName`
FROM `portal`.`ProductCategory` AS pc
INNER JOIN `portal`.`Language` AS l ON pc.`Id`
UNION ALL
SELECT
   p.`Id` AS `EntityKey`
        ,'Product' AS `Controller`
        ,'Store' AS `Action`
        ,CONCAT(p.`ProductCategoryId`, '?productId=', p.`Id`) AS `Parameters`
        ,l.`Id` AS `LanguageId`
        ,p.`Active`
        ,CONCAT('Izdelek: ', pc.`Title`) AS `ItemName`
FROM `portal`.`Product` AS p
INNER JOIN `portal`.`ProductContent` AS pc ON p.`Id` = pc.`ProductId`
INNER JOIN `portal`.`Language` AS l ON pc.`LanguageId` = l.`Id`
UNION ALL
SELECT
   n.`Id` AS `EntityKey`
        ,'News' AS `Controller`
        ,'News' AS `Action`
        ,n.`Id` AS `Parameters`
        ,l.`Id` AS `LanguageId`
        ,n.`Active`
        ,CONCAT('Novica: ', nc.`Title`) AS `ItemName`
FROM `portal`.`News` AS n
INNER JOIN `portal`.`NewsContent` AS nc ON n.`Id` = nc.`NewsId`
INNER JOIN `portal`.`Language` AS l ON nc.`LanguageId` = l.`Id`
ORDER BY `ItemName`
```

3.4 PRIPRAVA OGRODJA PROGRAMSKE REŠITVE

Namestimo Microsoft Visual Studio Community 2013 (v nadaljevanju VS) in na začetni strani izberemo *New Project -> Visual C# -> Class Library*. Projekt poimenujemo *Portal.Dal*, rešitev pa *Portal*. Podrobnosti so na sliki:

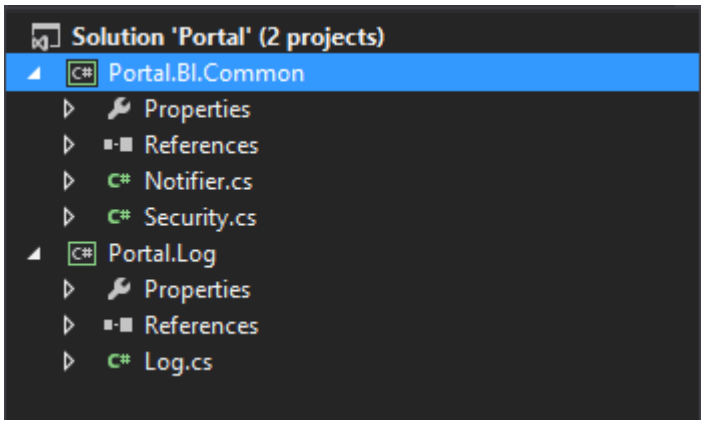


Slika 3.4-1: Priprava rešitve v orodju Microsoft Visual Studio 2013

Ker bomo pri razvoju potrebovali orodja za pisanje dnevnika, kriptirne metode, metode za pošiljanje elektronskih sporočil, ... vključimo v osnutek rešitve nekaj že izdelanih knjižnic iz preteklih projektov:

- namespace *Portal.Bl.CommonBl*:
  - metoda za kodiranje gesel z algoritmom SHA 256
  - metoda za impersonacijo nizkoprivilegiranega Windows uporabnika, ki jo bomo potrebovali za pisanje na disk v okviru strežniškega procesa
  - metoda za pošiljanje sporočil SMTP (email)
- namespace *Portal.Log*
  - Komponento za pisanje in samodejno arhiviranje dnevnika

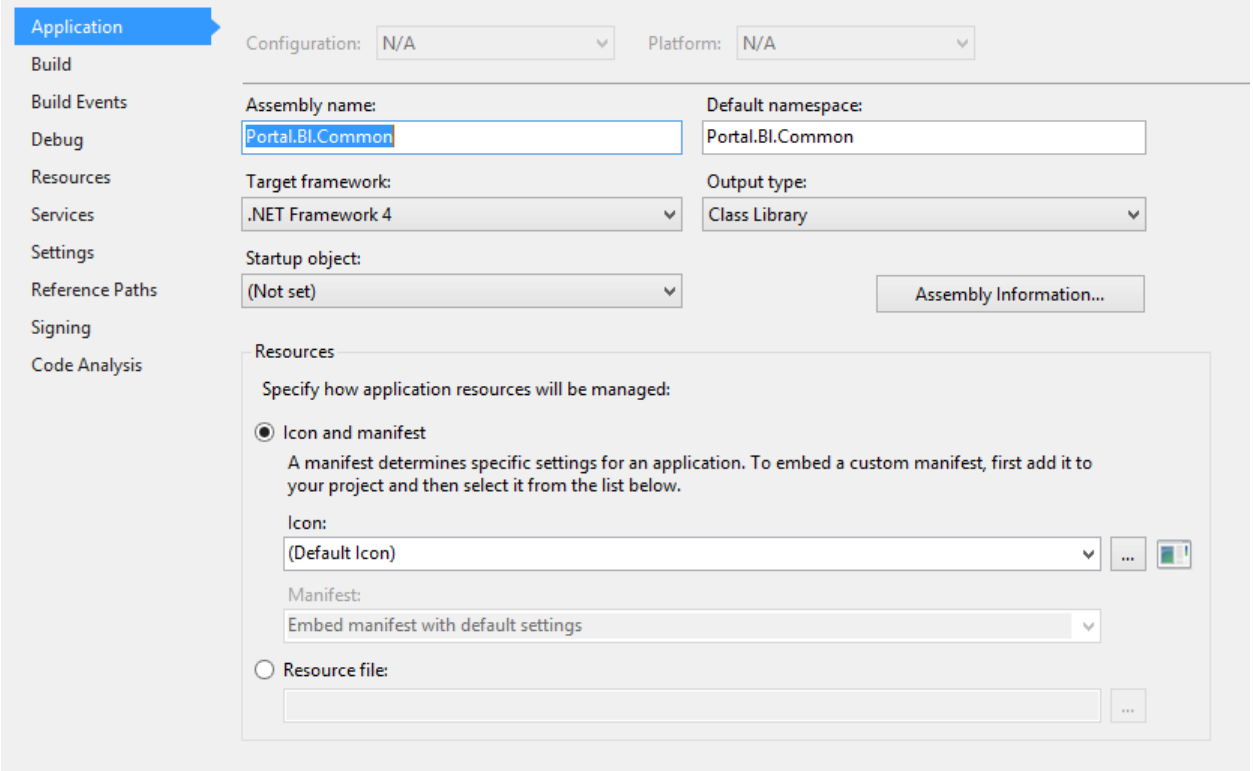
Solution Explorer izgleda tako:



Slika 3.4-2: Solution explorer

Ker naročnik zahteva kompatibilnost s platformo Windows XP SP1, smo omejeni na izvajalno okolje (ang. *Common Language Runtime*, *CLR*) .NET Framework 4.0 ali starejši.

Vsi projekti v rešitvi naj imajo ciljno izvajalno okolje nastavljeno na .NET Framework 4.0:



Slika 3.4-3: Izbira izvajalnega okolja

3.4.1 IMPLEMENTACIJA PODATKOVNEGA NIVOJA (DAL)

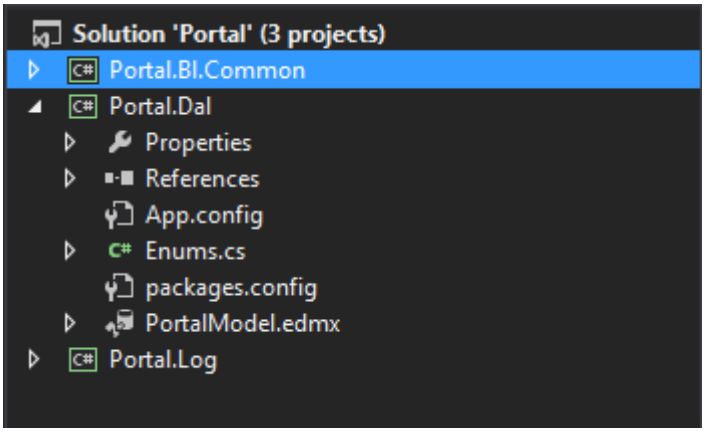
S pomočjo orodja za upravljanje programskih paketov NuGet (ang. *NuGet Package Manager*) v projekt dodamo sklic na knjižnico Entity Framework (v nadaljevanju EF) [20], ki nam zagotavlja ogrodje za interakcijo s podatkovno bazo MySQL. Zadnja verzija EF, ki po avtorjevih izkušnajh še pravilno deluje v okolju .NET Frameworku 4.0, je 5.0. Paket EF dodamo v rešitev na naslednji način:

```
PM> Install-Package EntityFramework -Version 5.0.0
```

Entity Framework je Microsoftovo ogrodje za manipulacijo podatkov po vzoru nHibernate, kar pomeni, da se razvijalcu ni potrebno ukvarjati z nadzorom nad povezavami na bazo podatkov in upravljanjem transakcij. EF se uporablja v navezi s knjižnico LINQ [21]. LINQ prevajalniku .NET omogoča preverjanje sintaktične pravilnosti poizvedb (ang. *query*) v času prevajanja programske kode (ang. *compile-time check*) in s tem učinkovito preprečuje tipkarske napake, ki se sicer lahko pojavijo pri pisanju poizvedb v jeziku SQL. Poizvedbe LINQ se med izvajanjem programa pretvorijo v poizvedbe SQL glede na izbranega ponudnika (ang. *provider*). V našem primeru je ponudnik knjižnica MySQL Connector [22], ki je prosto dostopen na spletni strani MySQL; paket namestimo ter ponovno zaženemo VS.

V projektu *Portal.Dal* ustvarimo podatkovni model (*Add -> New Item -> Data -> EF 5.x DbContext Generator*) in ga poimenujemo *PortalModel.edmx*. V čarovniku za pripravo modela se povežemo na bazo MySQL. Čarovnik iz baze uvozi imena tabel, stolpcev, pogledov in funkcij. Vrednost polja *Entity Container Name* nastavimo na *PortalEntities*. Na identifikator *PortalEntities* se bomo sklicevali na mestih v programski kodi, kjer bomo dostopali do podatkovne baze. Za vsa polja, poimenovana *Created* smo v poglavju 3.3.1 – Pravila za poimenovanje podatkovnih struktur predvideli privzeto vrednost ob ustvarjanju novega zapisa. Vsem poljem *Created* nastavimo *StoreGeneratedPattern* na *Computed*. EF posledično v polja *Created* ne bo zapisoval vrednosti medtem ko je branje še vedno dovoljeno.

S tem je delo na podatkovnem nivoju končano vsaj dokler se ne pojavi zahteva po spremembi podatkovnega modela s strani naročnika. Solution Explorer izgleda tako:



Slika 3.4-4: Priprava podatkovnega nivoja

Ko in če bo potrebno narediti spremembo na podatkovnem modelu, se ponovno poslužimo orodja MySQL Workbench. Dopolnimo podatkovni model in uporabimo funkcijo za pretvarjanje podatkovnega modela v SQL, *Database -> Forward Engineer*. V orodju VS odpremo datoteko *PortalModel.edmx* in uporabimo funkcionalnost za osveževanje podatkovnega modela iz baze *Update Model from Database*.

3.4.2 IMPLEMENTACIJA POSLOVNEGA NIVOJA (BL)

Rešitvi v VS dodamo novo knjižnico z *Add Project -> Class Library* in jo poimenujemo *Portal.Bl*.

Glede na rezultate vsebinske analize vemo, katere module potrebujemo v okviru Windows Forms odjemalca in katere za spletnega odjemalca. Glej priloženo kodo.

Kodiranje izvedemo po sledečem algoritmu:

1. Pripravimo t.i. ogrodje (ang. *frame*) za modul, npr. *News*. Ogrodje zajema imena razredov in metod knjižnice, vhodno-izhodne parametre in podatkovne tipe; vse metode naj vračajo podatek tipa *boolean*, ki bo klicočim metodam podal informacijo o uspešnosti izvajanja. Pripravimo sledeče metode:
   1.1. *GetNews*, namenjeno filtriranemu branju zapisov tipa *News*
   1.2. *UpdateNews*, namenjeno posodabljanju zapisov tipa *News*
   1.3. *DeleteNews*, namenjeno odstranitvi oz. deaktivaciji zapisov tipa *News*
2. Implementiramo metode modula, npr. *News*, kot je prikazano v priloženi kodi.
3. Vsak naslednji modul ustvarimo tako, da:
   3.1. pripravimo kopijo datoteke prvega modula, v našem primeru *News*,
   3.2. kopijo datoteke primerno preimenujemo glede na ime entitete, katero naj modul upravlja ter

3.3. izvedemo točko 2 za novi modul.

Priprava poslovnega nivoja naj poteka sinhrono s fazo priprave spletne storitve in nadzorne aplikacije. Pomembno je, da funkcionalnosti v rešitev dodajamo modularno [23]. Na ta način se razvijalec ukvarja s pripravo natanko enega modula naenkrat, preprečeno je miselno preskakovanje med vsebinami, izvedba je učinkovitejša, razumljivost programske kode je večja.

3.4.2.1 PRAVILA ZA PISANJE PROGRAMSKE KODE

V priloženi kodi so upoštevana pravila za pisanje programske kode v jeziku C# [25], ki jih priporoča podjetje Microsoft. Upoštevamo tudi naslednja pravila:

1. Vsaka metoda vrača podatek tipa *boolean*, ki vrne *True*, kar se metoda uspešno izvede, in *False*, kadar se ne. Pravilo razvijalcu omogoča enostavno razumevanje delovanja kode in enostavno testiranje.
2. Vsi razredi poslovne logike so statični (ang. *static*). S tem dosežemo minimalno porabo pomnilnika.
3. Vsi razredi poslovne logike ne hranijo stanja (ang. *stateless*). Z izjemo nespremenljivih konstant poslovna logika ne hrani stanja o delovanju, kar nam omogoča paralelizem pri izvajanju zahtev. Statična instanca razreda zadošča za izvajanje večih hkratnih klicev, saj bo CLR vsakemu klicu zagotovil ločen sklad (ang. *stack*) in izvajanje ne bo povzročilo neskladnosti lokalnih spremenljivk.
4. Vsak sklop programske kode, ki dostopa do zunanjih virov (baza podatkov, datotečni sistem, zunanji procesi, itd.), naj bo zajet v *try-catch*. Če se izvede stavek *catch*, naj metoda vrne vrednost *False*.
5. Vsaka izjema v izvajanju programa (ang. *exception*) naj bo zapisana v dnevniško datoteko s klicem metode *LogInstance.Log(...)*.

Včasih je potrebno podatkovni model in poslovni nivo prilagoditi novim zahtevam naročnika. Pomembno se je izogniti skušnjavi, da se namesto spremembe podatkovnega modela izvede prilagoditev obstoječe programske kode (ang. *hack* ali *kludge*) [24]. V kolikor se pretirava s takšnimi posegi, se sčasoma pojavijo problemi, ki otežijo vzdrževanje in v skrajnem primeru tudi onemogočijo ali zelo otežijo izvedbo novih naročil. Primer takšne napake je dvoumno ali napačno poimenovanje podatkovne strukture, ki ni popravljeno takoj, ko je napaka zaznana. S časom se lahko pojavi zahteva po novi podatkovni strukturi, ki bi morala nositi isto ime, vendar je le-to že zasedeno. Potrebno je popraviti programsko kodo na vseh mestih, kjer se uporablja napačno poimenovanje, ter ponovno testiranje. Večinoma pa razvijalci raje izberejo na kratek rok cenejšo pot in napačno poimenujejo novo podatkovno strukturo, s čimer se zmanjša razumljivost kode. Drug primer slabe prakse je dodajanje programske logike, ki ne

odpravi problema, ampak ga zaobide. Pogosta napaka takšne vrste je zapisovanje podatkov v polje istega tipa, ki pa izvorno ni namenjeno specifičnemu podatku. Primer: v tabeli *Address* hranimo v polju *Address* celoten naslov osebe. Naročnik nato zahteva obdelavo podatkov glede na kraj bivanja. Problem lahko rešimo na dva načina. Pravilna rešitev je, da v tabeli *Address* pripravimo stolpce *HouseNumber*, *Street*, *PostalNumber* in *Town* ter obstoječe podatke porazdelimo v te stolpce. Nepravilni pristop pa je, da se zanašamo na pravilni vrstni red vnosa podatkov s strani uporabnikov in privzamemo da je zadnji niz znakov v polju vedno kraj. Samo vprašanje časa je, kdaj se bo pojavil naslov, kjer zadnji znakovni niz ne bo predstavljal kraja. V praksi je avtor že zasledil primere, kjer programska koda specifičen podatek v tabeli obravnava kot izjemo, kar na dolgi rok seveda ni rešilo problema. S stroškovnega vidika je bilo potrebno problem reševati dvakrat, k strošku pa lahko prištejemo še čas ugotavljanja, ali je bila prvotna (nepravilna) rešitev morda na mestu ali gre samo za nedoslednost razvijalca. Primerov slabe prakse je še precej, vendar niso relevantni za pričujoče delo.

3.4.2.2 MODULI NADZORNE APLIKACIJE

Seznam modulov in funkcionalnosti, ki naj jih posamezen modul zajema:

1. Modul za avtorizacijo dostopa *AccessControlBl*
   1.1. Upravljanje dovoljenja za dostop uporabnika do funkcionalnosti, ki pritičejo uporabniškim vlogam, dodeljenim temu uporabniku.
2. Modul za upravljanje vsebin *AdministrationBl*
   2.1. Upravljanje z uporabniki:
      2.1.1. Pregled uporabnikov
      2.1.2. Dodajanje uporabnika
      2.1.3. Urejanje uporabnika
      2.1.4. Brisanje uporabnika
      2.1.5. Prijava uporabnika v sistem
      2.1.6. Odjava uporabnika iz sistem
   2.2. Upravljanje šifrantov z operacijami *beri*, *dodaj*, *uredi*, *briši* za:
      2.2.1. Šifrant držav
      2.2.2. Šifrant vsebinskih zaznamkov
      2.2.3. Šifrant kategorij izdelkov
      2.2.4. Šifrant uporabnikov sistema
   2.3. Upravljanje z novicami:
      2.3.1. Pregled novic
      2.3.2. Dodajanje novice
      2.3.3. Urejanje novice

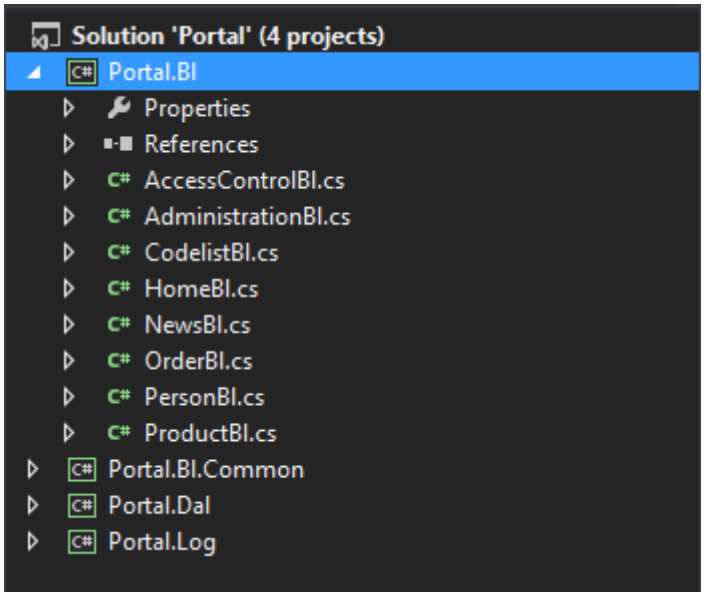
2.3.4. Brisanje novice
2.4. Upravljanje z izdelki:
2.4.1. Pregled izdelkov
2.4.2. Pregled izdelkov glede na kategorijo
2.4.3. Dodajanje izdelka
2.4.4. Urejanje izdelka
2.4.5. Brisanje izdelka
2.5. Upravljanje s povezavami na domači strani:
2.5.1. Branje seznama vseh povezav
2.5.2. Dodajanje povezave
2.5.3. Urejanje povezave
2.5.4. Brisanje povezave

3.4.2.3 MODULI SPLETNE APLIKACIJE

Seznam modulov in funkcionalnosti, ki naj jih posamezen modul zajema:

1. *CodelistBl*: modul za branje šifrantov
2. *HomeBl*: modul za branje povezav na domači strani
3. *NewsBl*: modul za branje novic
4. *ProductBl*: modul za branje kategorije izdelka, izdelkov in posameznega izdelka
5. *OrderBl*: modul za upravljanje naročil s strani končnega uporabnika:
   5.1. Urejanje naročila
   5.2. Pregled naročila
   5.3. Preklic/izbris naročila
6. *PersonBl*: modul za upravljanje s kupci
   6.1. Zapisovanje kupca ob sprejemu naročila
   6.2. Brisanje podatkov o kupcu ob odstranitvi naročila

Če je bralec sledil navodilom, rešitev v Solution Explorer-ju izgleda izgleda tako:



Slika 3.4-5: Poslovna logika spletne aplikacije

3.4.2.4 PRIMER MODULA

Spodnja programska koda predstavlja implementacijo modula *HomeBl*.

```
using System.Data;
using System.Linq;
using System.Collections.Generic;

using Portal.Log;
using Portal.Dal;

namespace Portal.Bl
{
        public static class HomeBl
   {
      private const string CallerId = "HomeBl";

      public static bool GetHomeItems(SupportedLanguage language, bool preview, ref
IList<view_index_homeitem> list)
      {
         list = null;

         try
         {
            using (PortalEntities context = new PortalEntities())
            {
               list = (
```

```
                    from hi in context.view_index_homeitem
                    where
                        (hi.LanguageId == (byte)language) &&
                        (preview ? true : hi.Active)
                    orderby hi.Order
                    select hi
                )
                .Take(8)
                .ToList();
            }

            return true;
        }
        catch (DataException ex)
        {
            LogInstance.Log(ex, CallerId);
            return false;
        }
    }
  }
}
```

3.5 SPLETNA APLIKACIJA

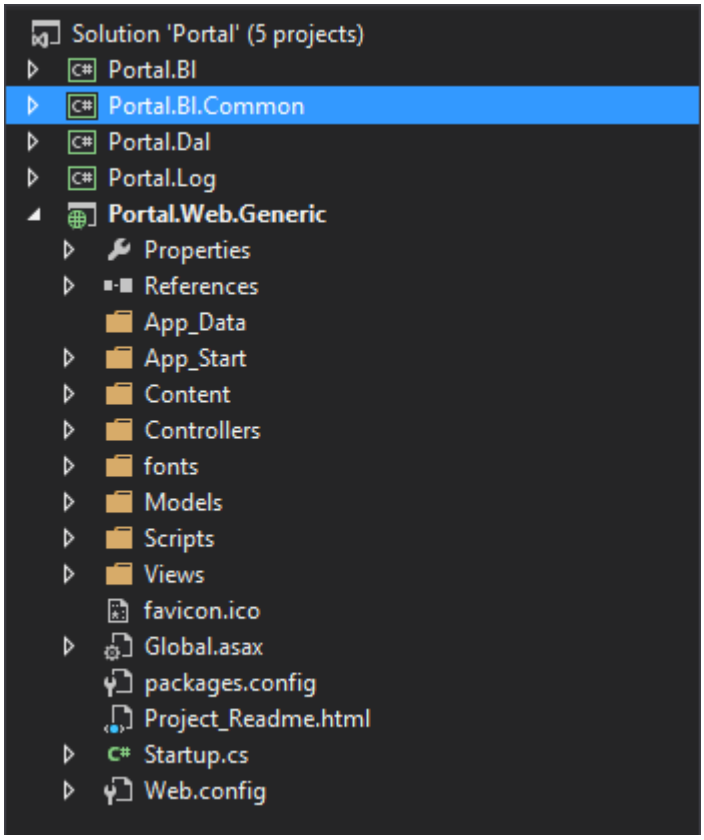
Ko je ogrodje poslovnega nivoja pripravljeno, pričnemo s pripravo spletnega odjemalca. V podjetjih proces pisanja programske kode praviloma poteka sinhrono s pripravo podatkovnega modela, kjer sta podatkovni model in poslovna logika vhodna parametra za postopek priprave spletne aplikacije. V našem primeru bomo ob pripravi podatkovnega modela vzporedno začrtali tudi ogrodji spletne in nadzorne aplikacije. Vsakič ko določimo podatkovne strukture nekega modula poslovne logike, pripravimo pripadajoči ogrodji tudi v obe aplikaciji. S takšnim pristopm je mogoče na enostaven način sproti testirati pravilnost delovanja aplikacije, pravilnost delovanja poslovne logike ter primernost podatkovnega modela.

3.5.1 PRIPRAVE

Rešitvi v VS dodamo MVC spletno aplikacijo [26], *Add Project -> ASP.NET Web Application*, projekt poimenujemo *Portal.Web.Generic*, izberemo možnost *MVC* in obkljukamo možnost *Web API* [27]. Z orodjem NuGet namestimo zadnjo stabilno različico knjižnice NewtonSoft.Json [28]:

```
PM> Install-Package Newtonsoft.Json
```

Solution Explorer izgleda tako:



Slika 3.5-1: Datoteke spletne aplikacije

Projekt poženemo in v brskalniku se odpre domača stran aplikacije. Do aplikacije lahko dostopamo tudi neposredno z zahtevo na naslov URL: *http://localhost:31097/Portal.Web.Generic/*.

Za projekt tipa MVC se avtor odloči, ker:

- zagotavlja preglednost in urejenost programskih komponent
- gre za moderen in dobro dokumentiran pristop k razvoju spletnih aplikacij, ki je prenosljiv na druge tehnologije (npr. Spring MVC),
- je relativno preprost za učenje in razumevanje.

Eden od alternativnih pristopov je ASP.NET Web Forms, ki pa je namenjen predvsem večjim poslovnim aplikacijam s kompleksnimi vnosnimi formami. Zahteva tudi temeljitejše poznavanje ravojnih pristopov.

Koncept Web API je posplošitev koncepta MVC. Koncepta se razlikujeta v vsebini odgovora (ang. *response*) na zahtevo HTTP. Zahteva HTTP za MVC pogled izriše (ang. *rendering*) kodo spletne strani (HTML in Javascript). Zahteva Web API pa vrača serializirano splošno podatkovno strukturo, katere namembnost ni vnaprej določena. Web API odjemalec je lahko katera koli aplikacija ali storitev, MVC odjemalec pa je načeloma samo brskalnik. V našem primeru bo Web API koriščen s strani Windows Forms odjemalca, t.j. nadzorne aplikacije.

Aplikacija se v času razvoja izvaja v okolju Visual Studio Development Server, ki aplikacijo namesti in poganja na strežniku IIS Express.

Kratek opis map v projektu:

- V mapi *App_Start* se nahaja konfiguracijska koda, ki se izvede ob instanciranju aplikacije. Vsaka instanca aplikacije izvede to kodo samodejno ob zagon.
- V mapi *Controllers* se nahajajo krmilniki (ang. *controller*s) spletne aplikacije. Vsak modul spletne aplikacije naj ima svoj krmilnik, razen *PersonBl*, ki je v celoti upravljan v okviru modula za sprejemanje naročil *OrderControllerBl*.
- V mapi *Models* hranimo podatkovne strukture, ki predstavljajo dinamične vsebine pogledov. Vsak pogled ima svoj model.
- V mapi *Views* hranimo poglede (ang. *views*). Pogledi so datoteke tipa *.cshtml*. Ko strežnik prejme zahtevo HTTP (ang. *request*), jo posreduje aplikaciji. Aplikacija pogled izriše (ang. *render*) s pomočjo t.i. poglednega pogona (ang. *view engine*) Razor, ki datoteko *.cshtml* prevede v niz znakov HTML in Javascript. Ta niz nato kot del odgovora HTTP (ang. *HTTP response*) posreduje nazaj strežniku, ta pa odjemalcu.

- V mapi *Content* se nahajajo oblikovna slogi (ang. *style sheets*) pogledov, zapisani v CSS 3.0, in slike (logotip, gumbi, etc.), na katere se sklicujejo slogi.
- V mapi *Images* bomo hranili slike in druge vsebine, ki jih bo v sistem naložil skrbnik z uporabo nadzorne aplikacije in se bodo prikazovale v pogledih.
- V mapi *Scripts* hranimo skripte, zapisane v jeziku Javascript, ki se uporabljajo v programski kodi pogledov.

Več o posameznih gradnikih, ki se nahajajo v naštetih mapah, bomo povedali v nadaljevanju.

3.5.2 USMERJANJE UPORABNIŠKIH ZAHTEV ZNOTRAJ SPLETNE APLIKACIJE

Strežnik IIS bo zahteve HTTP preusmeril do aplikacije. Aplikacija v mapi *App_Start* vsebuje datoteko *RouteConfig.cs*, ki se izvede ob zagonu nove instance aplikacije. Datoteka vsebuje pravila usmerjanja zahtev HTTP do posameznih modulov. Kriterij za usmerjanje je oblika naslova URL. Iz naslova URL strežnik IIS zazna, kateri aplikaciji naj posreduje zahtevo. Nato aplikacija iz naslova URL razbere, kateri pogled ali klic Web API je zahteval pošiljatelj zahteve.

Primer: Usmeritev na modul za izdelke v *RouteConfig.cs*:

```
routes.MapRoute(
   name: "Product",
   url: "Product/{action}/{id}",
   defaults: new { controller = "Product", action = "RandomProduct", id = UrlParameter.Optional }
);
```

Primer naslova URL za prikaz podrobnosti izdelka v trgovini je prikazan v poglavju *3.3.6.1 – Primer povezave na izdelek*.

Posebej pozorni moramo biti na privzeto preusmeritev. Kot odgovor na zahteve, ki jih ne bo mogoče usmeriti v okviru obstoječih pravil, zapisanih v *RouteConfig.cs*, bo vrnila privzeto stran.

Primer usmerjevalnega pravila iz datoteke *RouteConfig.cs*:

```
routes.MapRoute(
   name: "Default",
   url: "{controller}/{*AllValues}",
   defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

V mapi *App_Start* odpremo poiščemo *WebApiConfig.cs* in v metodo *Register* dodamo še pravilo za usmerjanje klicev Web API, ki ob napačno formirani zahtevi HTTP preusmeri odjemalca na privzeto stran:

```
public static void Register(HttpConfiguration config)
{
   config.Routes.MapHttpRoute(
      name: "Administration",
      routeTemplate: "api/{controller}/{action}/{linkId}",
      defaults: new
      {
         controller = "Home",
         action = "Index",
         linkId = ""
      }
   );
}
```

Zahteve HTTP za klic Web API, ki bodo preusmerjene na privzeto stran, je dovoljeno tudi zavreči. Na ta način dodatno razbremenimo omrežno infrastrukturo.

3.5.3 KRMILNIKI

Krmilniki implementirajo akcije (ang. *action*), ki jih sme priklicati dohodna uporabniška zahteva. Vsaka akcija je dosegljiva samo navedenemu tipu zahtev HTTP (GET, POST, HEAD, DEBUG, etc.). Vsaka zahteva HTTP, ki jo IIS posreduje aplikaciji in ustreza enemu od usmerjevalnih pravil bo priklicala zahtevano akcijo na zahtevanemu krmilniku.

3.5.3.1 PRIMER KRMILNIKA

Spodnja koda prikazuje primer krmilnika z akcijo, imenovano *Index*, ki je vidna zahtevam HTTP GET. Krmilnik bo pred izvedbo akcije s klicem metode *GetLanguageId* določil jezik, ki ga zahteva brskalnik, in ga shranil v polje *_language*. Akcija *Index* iz podatkovne baze prebere povezave *HomeItem* za izbrani jezik in ustvari pogled */Views/Home/Index.cshtml*. Če branje povezav uspe, v model zapišemo seznam povezav, sicer v model zapišemo sporočilo o napaki. Model nato posredujemo pogledu *Index*.

```
public class HomeController : Controller
{
   private SupportedLanguage _language;

   protected override void OnActionExecuting(ActionExecutingContext filterContext)
   {
      _language = ControllerHelper.GetLanguageId(Request);
```

```
            base.OnActionExecuting(filterContext);
        }

        [HttpGet]
        public ActionResult Index(string id)
        {
            IList<view_index_homeitem> list = null;
            if (!HomeBl.GetHomeItems(_language, !string.IsNullOrEmpty(id), ref list))
            {
                ViewData["Error"] = Strings.Common.Error_Server_ReadData;
            }

            ViewData["LanguageId"] = (byte)_language;

            return View(list);
        }
    }
```

3.5.3.2 PRIMER IZVAJANJA ZAHTEVE HTTP

Vzemimo primer, ko uporabnik prvič obišče spletno aplikacijo. Izvajanje poteka po sledečih korakih:

- uporabnikov brskalnik strežniku IIS pošlje zahtevo HTTP GET *http://www.domena.com*,
- IIS bo zahtevo posredoval aplikaciji
- aplikacija bo zahtevo usmerila na pot *default* in s tem na krmilnik *Home* ter akcijo *Index* s parametrom *id = null* (glej *HomeController.cs*),
- akcija bo pridobila zahtevane podatke in jih posredovala poglednemu pogonu
- pogledni pogon bo obdelal pogled *Views/Home/Index.cshtml* in izrisal kodo v odgovor HTTP (ang. *HTTP response*),
- IIS bo odgovor HTTP posredoval brskalniku uporabnika.

3.5.4 MODELI

Večina pogledov prikazuje dinamične podatke, kot so npr. novice v izbranem jeziku, sporočila o napakah, itd. Dinamično podatkovno strukturo, ki hrani te podatke, imenujemo model. Krmilniške akcije pridobijo podatke iz podatkovne baze, sestavijo model in ga posredujejo poglednemu pogonu. Pogledni pogon bo nato iz modela poskušal prebral podatke, na katere se sklicuje programska koda v pogledu.

3.5.4.1 PRIMER MODELA

Spodnja koda prikazuje primer uporabe modela *ContactModel* v akciji *Contact*, ki se proži, kadar uporabnik na spletni strani obišče naslov URL *Views/Home/Contact.cshtml*, izpolni kontaktni obrazec in ga pošlje na strežnik z zahtevo HTTP POST.

Model *ContactModel* vsebuje podatke, ki jih vpiše uporabnik v razdelku strani *Contact*:

- sporočilo ,
- kontakt uporabnika,
- poročilo o uspehu/neuspehu operacije posredovanja sporočila skrbniku sistema.

```
public class ContactModel
{
   public string Message { get; set; }
   public string Contact { get; set; }

   public string Error { get; set; }
   public string Success { get; set; }
}
```

Implementacija akcije *Contact* v krmilniku *HomeController.cs*:

```
[ValidateInput(false)]
[HttpPost]
public ActionResult Contact(ContactModel model)
{
   if (CommonBl.EmailContact(model.Message, model.Contact))
   {
      model.Success = Strings.Contact.Success;
   }
   else
   {
      model.Error = Strings.Contact.Error;
   }
   return View(model);
}
```

3.5.5 POGLEDI

Pogled služi kot predstavitveni nivo modelu. Pogled se izvaja v poglednem pogonu Razor [29, 30]. Datoteka s pogledom ima končnico *.cshtml* in vsebuje kodo HTML in Javascript, dovoljeno je tudi označevanje (ang. *markup*). Oznaka je sklop programske kode, napisane v jeziku C#. Poglednemi pogon izvede kodo, zajeto v oznaki, in rezultata izriše na mesto, kjer je oznaka umeščena v pogled. Ko pogledni pogon zaključi obdelavo oznak, zapiše pogled v odgovor HTTP. S tem je obdelava posamezne zahteve HTTP v okviru spletne aplikacije zaključena.

3.5.5.1 PRIMER POGLEDA

Koda prikazuje vsebino pogleda *Store.cshtml*. Model, ki ga pogled predstavi, je *Portal.Web.Models.StoreModel*. Izrisani pogled prikaže vstopno stran spletne trgovine (ang. *store*):

```
@{
    ViewBag.Title = Strings.Product.Title;
}
@using Portal.Dal
@inherits WebViewPage<Portal.Web.Models.StoreModel>

@if (!string.IsNullOrEmpty(Model.Error))
{
    <div class="error">@Model.Error</div>
}
else if (Model.Categories.Count == 0)
{
    <div class="empty">Trenutno ni na voljo noben izdelek.</div>
}
else
{
    <link rel="stylesheet" type="text/css" href="~/Content/jquery-ui-1.9.1.custom.min.css" />
    <link rel="stylesheet" type="text/css" href="~/Content/Product/Store.css" />
    <link rel="stylesheet" type="text/css" href="~/Content/Product/ProductThumbs.css" />
    <link rel="stylesheet" type="text/css" href="~/Content/Product/Details.css" />
    <script type="text/Javascript" src="~/Scripts/UI/jquery.ui.core.min.js"></script>
    <script type="text/Javascript" src="~/Scripts/UI/jquery.ui.widget.min.js"></script>
    <script type="text/Javascript" src="~/Scripts/UI/jquery.ui.button.min.js"></script>
    <script type="text/Javascript" src="~/Scripts/Product/cart.js"></script>
    <script type="text/Javascript" src="~/Scripts/Product/store.js"></script>
    <script type="text/Javascript">
        $(function () {
            $('#categoryList img').click(function () {
                selectCategory($(this), '@Url.Action("ProductThumbs", "Product")/');
            });
        });
    </script>

    <ul id="categoryList">
    @foreach (view_product_productcategory category in Model.Categories)
    {
        <li>
            <img id="c_@category.Id" class="transparent @(Model.CategoryId == category.Id ? "selected":
    string.Empty)" src="@Url.Content(Portal.Bl.CommonBl.ImageUri("~/Images/Category", Model.LanguageId,
    category.Id) + ".png")" alt="" title="" />
        </li>
```

```
    }
    </ul>
    <div id="productListContainer">
        @Html.Action("ProductThumbs", "Product", new { id = Model.CategoryId, productId =
    Model.ProductId, preview = Model.Preview })
    </div>
} @* end content *@
```

Slika 3.5-2: Izris pogleda *Store.cshtml*

3.6 NADZORNA APLIKACIJA

Nadzorna aplikacija bo v skladu z dogovorom z naročnikom služila kot orodje za upravljanje vsebin spletne aplikacije. Na željo naročnika moramo zagotoviti upravljanje vsebin zajetih v točki 2 poglavja 3.4.2.2 – Moduli nadzorne aplikacije:

- upravljanje z uporabniki,
- upravljanje šifrantov,
- upravljanje z novicami,
- upravljanje z izdelki,
- upravljanje s povezavami na domači strani.

Naročniku smo predlagali, da nadzorno aplikacijo razvijemo kot Windows Forms aplikacijo. Kot osnovo za izvedbo imamo na voljo izvorno kodo aplikacije, ki jo je avtor razvil za naročnika. Drug razlog je cena razvoja spletnega odjemalca, ki je praviloma višja kot razvoj Windows Forms aplikacije. Naročnik zagotovi, da dostopa do nadzornega modula izven delovnega časa ni potrebno zagotavljati. Prav tako ni potrebno zagotavljati dostopa iz poljubne naprave.

Primerjava cene razvoja spletne in Windows Forms aplikacije:

1. HTML v navezi s knjižnico jQuery ponuja manjši nabor gradnikov uporabniškega vmesnika (ang. *controls*) kot Windows Forms, kar pomeni, da bi zapletenejše gradnike morali razviti sami. Če bi HTML in jQuery dodali še knjižnico Bootstrap in vtičnike za Bootstrap, se nabor gradnikov precej izenači.
2. Programske kode za spletni vmesnik je načeloma vedno potrebno spisati več kot primerjive kode v Windows Forms rešitvi. Višek kode nastane predvsem pri oblikovanju HTML. V Windows Forms je oblikovanje vnosne maske strogo ločeno (datoteka s končnico *.designer.cs*) od implementacije funkcionalnosti.
3. Windows Forms aplikacijo lahko izpeljemo iz že delujoče aplikacije s podobnimi funkcionalnostmi.
4. Zahteva za podporo starejšim brskalnikom lahko ustvari dodaten strošek pri razvoju spletne aplikacije.

Naročnik nam ugodi in podpre naš predlog.

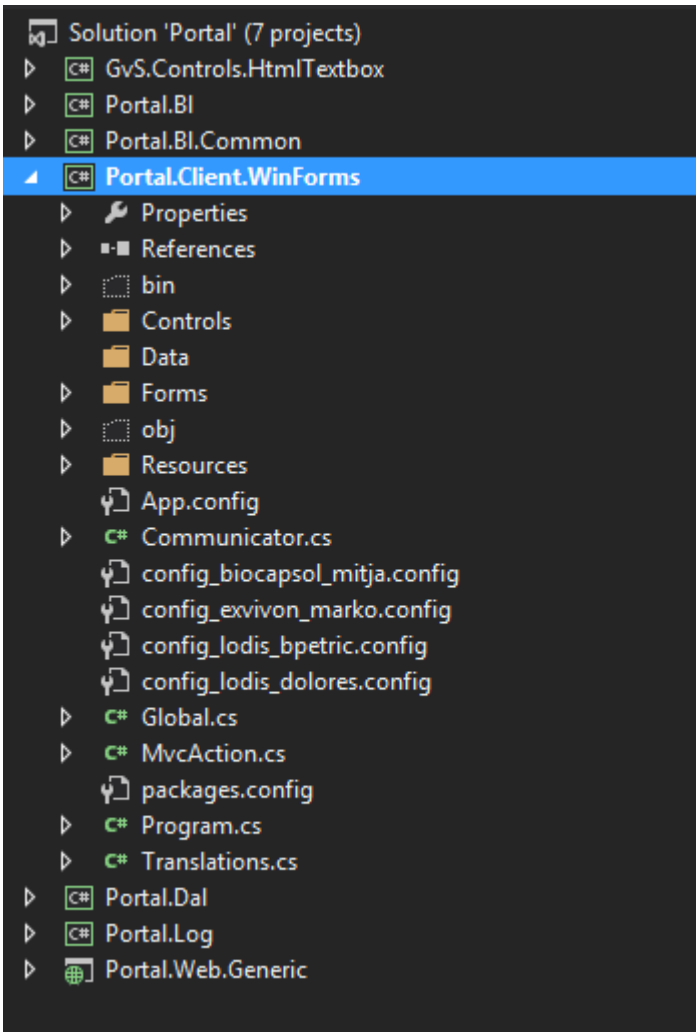
Nadzorna aplikacija naj sestoji iz:

- uporabniškega vmesnika,
- komponente za komunikacijo *Communicator*, ki z uporabo zahtev HTTP komunicira z Web API krmilnikom spletne aplikacije *AdministrationController.cs*.

3.6.1 PRIPRAVE

Rešitvi v VS dodamo Windows aplikacijo *Add Project -> Windows Forms appplication* in projekt poimenujemo *Portal.Client.Winforms*.

Solution Explorer izgleda tako:



Slika 3.6-1: Datoteke Windows Forms odjemalca

Če projekt poženemo, se prikaže nadzorna aplikacija.

Kratek opis gradnikov projekta:

- datoteka *App.config* je konfiguracijska datoteka za aplikacijo,
- v mapi *Controls* se nahajajo gradniki uporabniškega vmesnika (ang. *user controls*),
- v mapi *Forms* so definirane forme uporabniškega vmesnika (ang. *application forms*).
- mapa *Data* je prazna in je namenjena shranjevanju uporabniških vsebin na trdi disk med delovanjem aplikacije,

- v mapi *Resources* se nahajajo datoteke (slike, dokumenti, ipd.), ki jih potrebujemo v času razvoja,
- razred *MvcAction* je komponenta za neposredno komunikacijo s spletno aplikacijo, t.j. posredovanje zahtev in odgovorov HTTP,
- razred *Communicator* služi kot predstavitveni nivo razreda *MvcAction* in poenostavlja uporabo razreda *MvcAction*,
- razred *Translator* vsebuje slovar ključev s pripadajočimi prevodi, ki se uporabljajo za lokalizacijo aplikacijskih gradnikov.

Prevode hranimo v razredu *Translator*, priporočena praksa pa veleva, da se prevode hrani v datotekah z viri *.resx*. Utemeljitev odstopanja:

1. Vse prevode imamo v enem samem razredu, število *.resx* datotek pa je vezano na število vnosnih mask (ang. *forms*). Vsaka vnosna maska potrebuje eno *.resx* datoteko za vsak podprt jezik.
   1.1. Urejanje ene same datoteke je lažje in preglednejše.
   1.2. Ena sama datoteka pomeni manjšo možnost, da asociiramo isti prevod z več različnimi ključi.
   1.3. Ena sama datoteka pomeni manjšo možnost, da se pojavi več ključev z istim imenom.
2. Če bi naročnik zahteval podporo dodatnemu jeziku, bi razredu *Translator* dodali ključ za izbiro jezika in nov sklop prevodov. Uporaba datotek z viri *.resx* pa zahteva ponovno dodajanje *.resx* za nov jezik za vsako vnosno masko, kajti le na ta način zna Windows Forms samodejno poiskati prevode za jezik, ko se le-ta zamenja med izvajanjem aplikacije.

Več o posameznih gradnikih bomo povedali v nadaljevanju.

3.6.2 KOMUNIKACIJA S SPLETNO APLIKACIJO

Do spletne aplikacije bomo dostopali na dva načina:

- z naslovom URL za predogled podatkovne strukture (npr. *News*), ki dostopa do krmilnika spletne aplikacije enako kot brskalnik,
- z naslovom URL za upravljanje podatkovne strukture (npr. *News*), ki dostopa do Web API krmilnika spletne aplikacije.

Generiranju zahtev sta namenjena razreda *MvcAction* in *Communicator*. Vsaka metoda razreda *Communicator* vrača rezultat tipa *bool*, ki označuje ali je metoda uspela ali ne.

Zahteve HTTP za predogled so običajne zahteve HTTP, ki jih aplikacije odpre v brskalniku, kadar si nazornik želi ogledati, kako bo neka urejana vsebina izgledala v spletni aplikaciji, ko bo objavljena.

Zahteve HTTP za Web API so namenjene urejanju vsebin (dodajanje, urejanje, brisanje). Psamezni deli zahtev so definirani v konfiguracijski datoteki app.config, lepi pa jih metoda MvcAction.ActionUri:

```
string.Concat(
   Global.AdministrationUri,    // http://localhost:31097/Portal.Web.Generic/
   action, "/",
   Global.AdministrationLinkId, // BFE0BDB6AD4C4A7CA3F42CB7A1FAB5C0
   Communicator.User.Username.ToRequestParameter("username", true) +
   Communicator.User.Password.ToRequestParameter("passwordHash"),
   arguments
);
```

Primer zahteve za prikaz seznama kategorij:
http://localhost:31097/Portal.Web.Generic/api/Administration/GetProductCategories/BFE0BDB6AD4C4A7CA3F42CB7A1FAB5C0?username=skrbnik&passwordHash=%2flIGdrGh2T2rqyMZ7qA2dPNjLq7rFj0eiCRPXrHeEOs%3d

3.6.2.1 ZAGOTAVLJANJE VARNE KOMUNIKACIJE

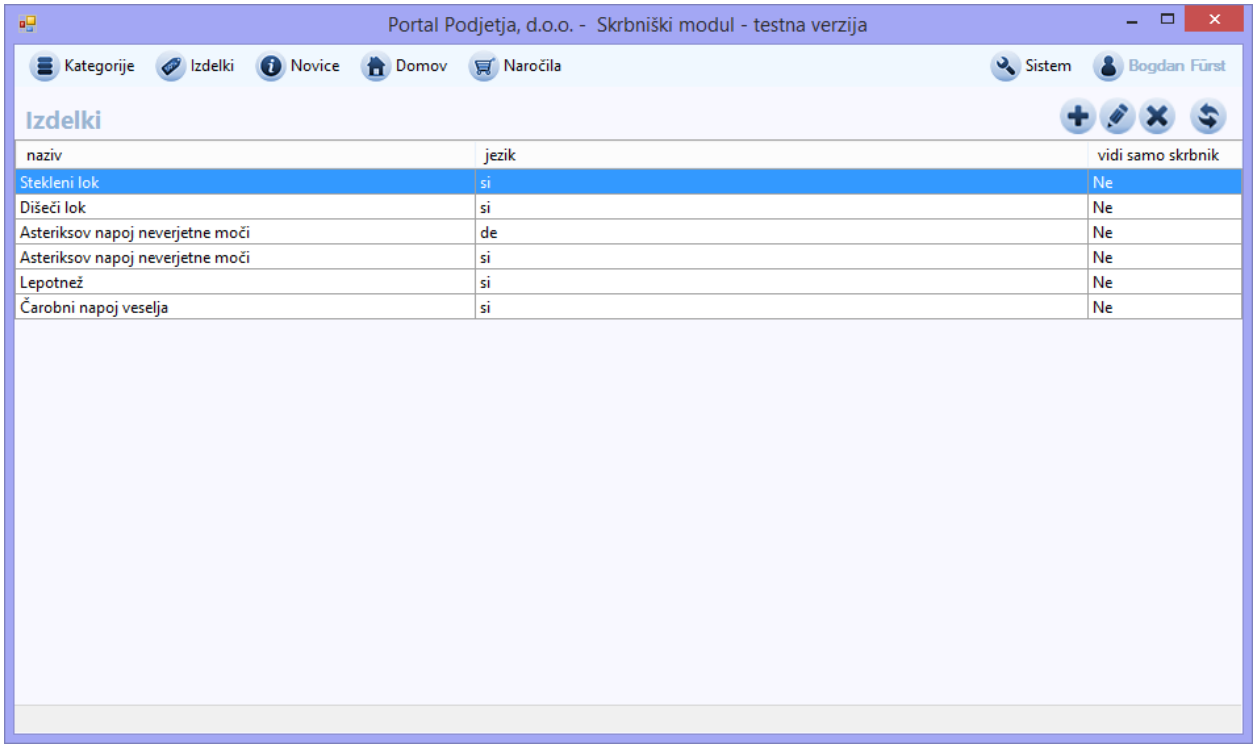
Naročnik sicer ni zahteval kriptiranja komunikacijskega kanala med strežnikom IIS in brskalnikom. Aplikacija nameščena v okolju, kjer so uporabniki slabo obveščeni o posledicah izbire šibkega gesla ali posredovanja gesla tretjim osebam. Vsakemu nadzorniku zato dodelimo unikatni ključ tipa *Guid* za dostop do spletne aplikacije in ga poimenujemo *LinkId* (glej poglavje 3.3.2 – Podatkovne strukture nadzorne aplikacije). Ključ shranimo v konfiguracijsko datoteko nadzorne aplikacije v procesu namestitve, od koder ga aplikacija prebere vsakič, ko poskuša komunicirati s spletno aplikacijo. Ključ bo del naslova URL, na neveljavni ključ ali ključ, ki pripada drugemu nadzorniku, se krmilnik Web API spletne aplikacije ne bo odzval. S tem preprečimo posameznemu nadzorniku, da v aplikacijo dostopa z uporabniškim imenom in geslom drugega nadzornika tudi, če si nadzornika izmenjata gesli. Z uporabo ključa *LinkId* otežimo tudi morebitni napad na našo spletno aplikacijo, kjer bi napadalec skušal zaznati naslove URL, na katere se spletna aplikacija odziva in mu omogočajo manipulacijo vsebin.

3.6.3 UREJANJE SKRBNIŠKIH VSEBIN

Večino vsebin urejamo po naslednjem postopku:

- prikaz seznama sorodnih vsebin (npr. seznam novic *News*)
- dodajanje, urejanje in odstranjevanje posamene vsebine (npr. posamezne novice *News*).

Modul za upravljanje posameznega tipa vsebine (npr. *News*) torej sestoji iz seznama sorodnih vsebin, ter vnosne maske za urejanje posamezne vsebine. Posamezno vsebino naj ima nadzornik možnost vpisati vnesti v vseh jezikih, v katere bo prevedena spletna stran. V aplikaciji, ki smo jo uporabili za izhodišče, smo uporabili ikone Luna Blue [31, 32]. Pri določanju velikosti vnosnih mask upoštevamo da bodo nadzorniki opremljeni s prenosnimi računalniki, ki zmorejo prikazati sliko ločljivosti največ 1024 × 768 točk.



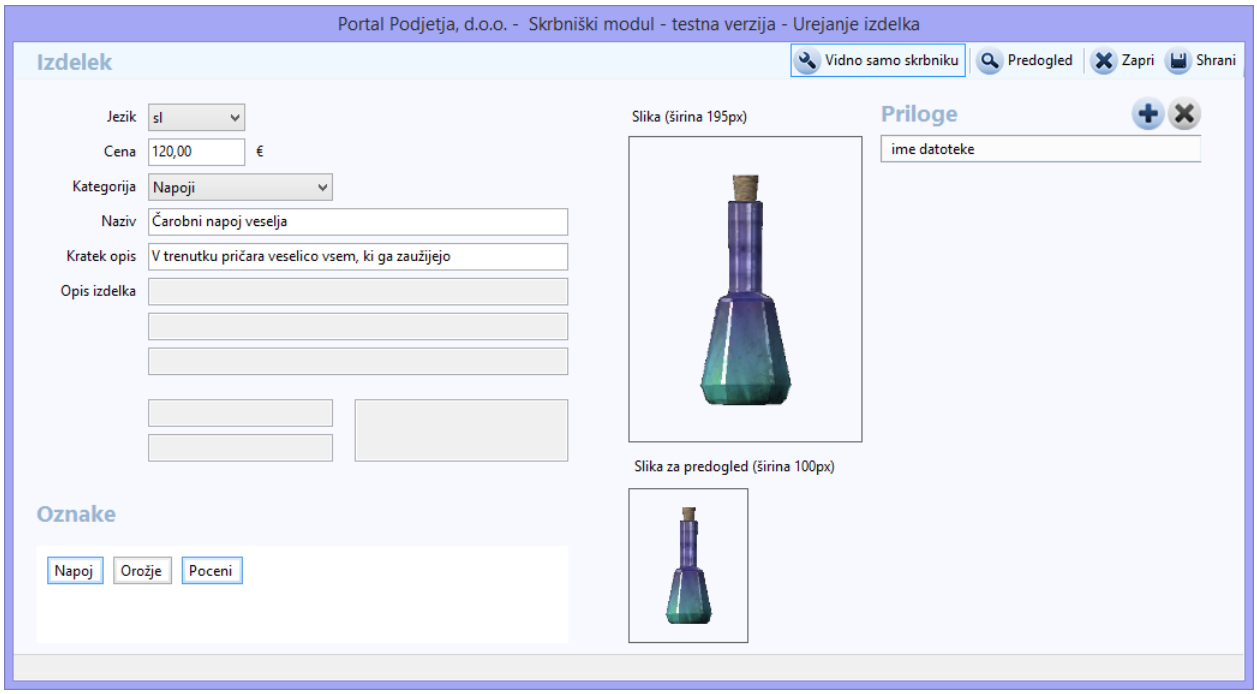
Slika 3.6-2: Urejanje uporabniških vsebin - Seznam izdelkov

Na vnosni maski za urejanje izdelkov naročnik zateva prikaz naslednjih podatkov:

- Jezik,
- Cena v evrih,
- Izbira kategorije izdelka,
- Naziv izdelka,
- Kratek opis izdelka,
- Podroben opis izdelka,
- Velika prikazna slika, ki se prikaže v podrobnostih izdelka,
- Majhna prikazna slika, ki se prikaže v seznamu izdelkov,
- Priponke.

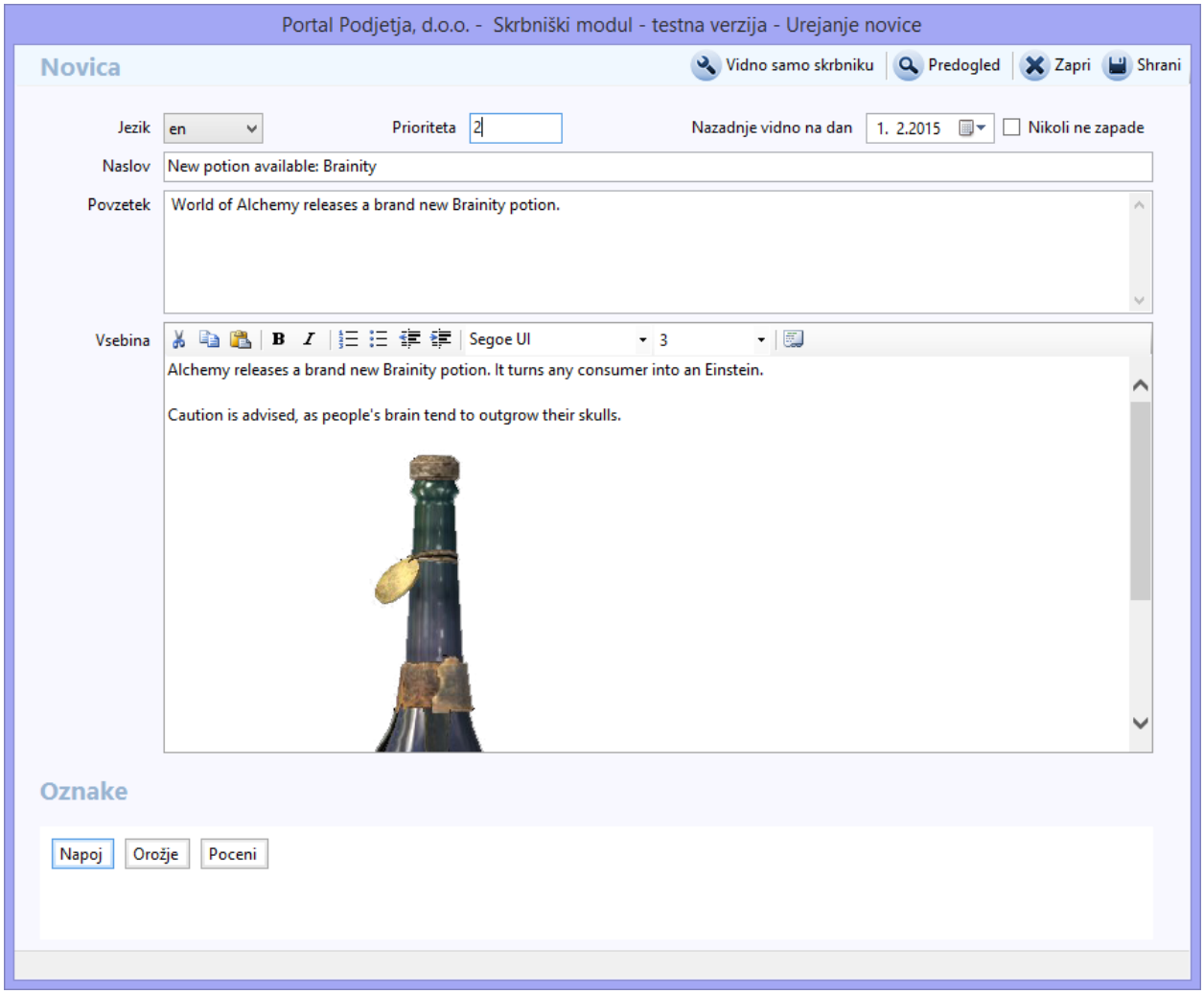
Izdelek je v spletni aplikaciji viden samo kadar se jezik vsebine ujema z nastavitvijo jezika v brskalniku obiskovalca strani.

Priponke zajemajo tehnično dokumentacijo izdelka, dokazila o neoporečnosti izdelka, dokazila o standardizaciji izdelka, ipd. Priponke hranimo v mapi *Data*.



Slika 3.6-3: Urejanje uporabniških vsebin - Podrobnosti izdelka

Naročnik po pregledu prototipa nadzorne aplikacije zahteva, da aplikacija omogoča napredno oblikovanje besedila novic. Zahtevi ugodimo kljub temu, da ne gre za pogodbeno obveznost in za nas pomeni dodatno delo. Za napredno oblikovanje vsebin uporabimo odprtokodno in brezplačno komponento *GvS.Controls.HtmlTextbox* [33].



Slika 3.6-4: Urejanje uporabniških vsebin - Podrobnosti novice

3.7 NAMEŠČANJE REŠITVE PRI NAROČNIKU

Z naročnikom se dogovorimo za datum namestitve. Razvoj je potekal brez večjih zapletov. Naročnik je vsa odstopanja od začetnih dogovorov sporočil pravočasno, zato ni prišlo do večjih časovnih razhajanj od načrta oziroma so se razhajanja izničila. Datum nameščanja na testni sistem je enak datumu, določenem v okviru časovnice iz poglavja 3.1.5. – Časovnica.

3.8 TESTNO OBDOBJE

Naročnik izvaja testiranje tako spletne kot nadzorne aplikacije. Potrebno je bilo odraviti nekaj manjših napak in narediti nekaj manjših popravkov na željo naročnika. Vse aktivnosti so potekale v okviru časovnice iz poglavja 3.1.5. – Časovnica.

3.9 PREVZEM REŠITVE

Izkaže se, da je testni sistem, na katerega smo nameščali testno verzije rešitve, tudi končni produkcijski sistem, zato ponovna postavitev okolja ni potrebna. Razvoj je uspešno zaključen.

Naročnik uporabi spletno trgovino za promocijo na večjem mednarodnem sejmu, kjer predstavi svoje izdelke in rešitev, ki smo jo razvili. Naročnik izpostavi, da oblikovanje ni zadovoljivo, vendar ne privoli v dodatno naročilo. Ker je za oblikovanje grafične podobe spletne tgrovine bil zadolžen naročnik in je bil pri izvedbi neuspešen, avtor ne privoli v brezplačno spremembo podobe trgovine, ki jo naročnik zahteva. Naročnik zamuja s plačilom, za katero je bilo dogovorjeno, da bo izvedeno v roku enega tedna od prevzema. Konflikt avtor reši z grožnjo z ovadbo zaradi goljufije in naročnik delno poravna svoje obveznosti. Nekaj mesecev po prevzemu naročnik preneha z dejavnostjo. Z dejavnostjo preneha tudi krovno podjetje.

4 SKLEPNE UGOTOVITVE

Razvoj je zaključen, reštive je bila prevzeta s strani naročnika. Uspešnost projekta določimo z uspešnostjo izpolnitve ciljev, zadanih v fazi načrtovanja.

Uspešno so bili izpolnjeni naslednji cilji:

1. Razvoj je potekal v okviru začrtane časovnice rokov in v predvidenem finančnem okvirju.

2. Izpolnjene so bile vse zahteve naročnika, ki so odstopale od začetnega načrta in so bile sporočene pravočasno, t.j. v času razvoja in testnega obdobja (npr. uvedba naprednega oblikovanja spetnih vsebin).

3. Izdelali smo platformo za upravljanje s spletno trgovino, ki jo zmore upravljati tehnično nevešča oseba.

4. Programsko ogrodje, ki smo ga razvili, je enostavno vzdrževati in ne omejuje kasnejšega razvoja. Za morebitne nove naročnike je predvidena replikacija programske kode. Vse različice rešitve bodo lahko slonele na trenutnem ali razširjenem podatkovnem modelu.

Večja odstopanja od začetnega načrta s strani naročnika:

1. Večje spremembe v grafičnem načrtu strani: neodzivnost in posledično odpoved sodelovanja zunanjemu sodelavcu, ki je bil zadolžen za dobavo osnutka grafične podobe, posledično smo morali oblikovanje strani zagotoviti sami.

2. Naročnik ni izplačal pogodbeno določenih sredstev v roku, določenem s pogodbo. Sodelovanje je bilo posledično zamrznjeno do izpolnitve obveznosti.

Večjih odstopanj od začetnega načrta z naše strani ni bilo.

4.1 PREDLOGI ZA IZBOLJŠAVE

Naročnik je z rešitvijo zadovoljen in rešitev uporablja za tržne namene. Po mnenju naročnika, bi v okviru razvoja lahko izboljšali:

- grafično podobo spletnih trgovin, kjer bi bilo potrebno najeti poklicnega oblikovalca, ki bi bolje ugodil željam naročnika.

Po avtorjem mnenju je prostor za izboljšanje predvsem v:

1. Pogajanjih za ceno končnega izdelka, kjer je bila izpogajana cena že v izhodišču prenizka glede na ocenjeno količino vloženega dela in kvaliteto.

2. Temeljitejšem informiranju glede plačilne sposobnosti pravne osebe, s katero sklepamo pogodbo (npr. tedensko izstavljanje računa, ločeno izstavljanje računa za vsak modul spletne trgovine, ...). Možna rešitev za problem plačilne nediscipline je pogodbeno določilo, ki zahteva dovolj visoko predplačilo. Na ta način se lahko prepričamo, da naročnik zmore zagotoviti dovolj finančnih sredstev za izvedbo naročila in vsaj delno pokrijemo stroške razvoja.

3. Podpora mobilnim napravam v spletni trgovini je slaba. V letu 2012 mobilne naprave še niso štele kot ena od primarnih ciljnih platform. V obstoječo kodo uporabniškega vmesnika bi bilo potrebno vpeljati katero od modernih knjižnic Javascript, ki zagotavljajo kompatibilnost z mobilnimi napravami, npr. Bootstrap.

4.2 SKLEPNA BESEDA

Pričujoče delo dokumentira razvoj programske rešitve za realnega naročnika. Poskuša slediti teoretičnim okvirom, vendar samo do točke, kjer obseg del, ki izhajajo iz teoretičnega pristopa k vodenju projekta in razvoju, ne zahteva prevelikega časovnega vložka. Dokumentirani so nekateri pomembnejši dejavniki, ki vplivajo na razvoj in lahko bralcu služijo kot oporne točke ali opozorila pri načrtovanju razvoja programske opreme. CMS, ki ga je predstavil avtor, je kot rešitev iz leta 2012 še vedno aktualen tudi v letu 2016, tako s tehnološkega vidika, kot z vidika načrtovanja in vodenja majhnih projektov na področju razvoja programske opreme. Programska koda rešitve upošteva razvojne smernice, ki jih je začrtalo podjetje Miscrosoft in prikazuje nekatere dobre prakse pri pisanju programske kode. Rezultat upoštevanja dobrih praks je, na primer, enostavnost namestitve rešitve v Azure oblak. Zaradi pravilno implementirane arhitekture je rešitev precej enostavno razširljiva, npr. z WCF storitvami. Zastaral pa je pristop k oblikovanju grafičnega vmesnika, kjer danes aktualne tehnologije slonijo na t.i. mobilnem (ang. *mobile-first*) pristopu k oblikovanju in razvoju spletnih uporabniških vmesnikov, kot ga omogočajo razširjene knjižnice/ogrodja Boostrap, AngularJS, itd. Vendar je zaradi pravilne implementacije arhitekrure mogoče rešitev nadgraditi z novim spletnim vmesnikom, ki sloni na obstoječi logiki, t.j. brez večjih posegov v druge arhitekturne nivoje reštive.

LITERATURA

[1] Web content management system. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Web_content_management_system. [Dostopano marec 2016].

[2] Web application. [Online]. Dosegljivo: http://en.wikipedia.org/wiki/Web_application. [Dostopano marec 2016].

[3] Spletna stran [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/Spletna_stran. [Dostopano marec 2016].

[4] Windows Forms [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Windows_Forms#Windows_Forms_application. [Dostopano marec 2016].

[5] Model-view-controller [Online]. Dosegljivo: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller. [Dostopano marec 2016].

[6] Comparison of C Sharp and Java [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Comparison_of_C_Sharp_and_Java. [Dostopano marec 2016].

[7] The Best PHP Framework for 2015: SitePoint Survey Results [Online]. Dosegljivo: http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/. [Dostopano marec 2016].

[8] ACID [Online]. Dosegljivo: http://en.wikipedia.org/wiki/ACID. [Dostopano marec 2016].

[9] When are you required to have a commercial MySQL license? [Online]. Dosegljivo: http://www.xaprb.com/blog/2009/02/17/when-are-you-required-to-have-a-commercial-mysql-license/. [Dostopano marec 2016].

[10] MySQL vs PostgreSQL [Online]. Dosegljivo: http://www.wikivs.com/wiki/MySQL_vs_PostgreSQL. [Dostopano marec 2016].

[11] Open Database Connectivity [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Open_Database_Connectivity. [Dostopano marec 2016].

[12] Visual SVN Server Licensing [Online]. Dosegljivo: http://www.visualsvn.com/server/licensing/. [Dostopano marec 2016].

[13] SQL:2011 [Online]. Dosegljivo: https://en.wikipedia.org/wiki/SQL:2011. [Dostopano marec 2016].

[14] Agile software development [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Agile_software_development. [Dostopano marec 2016].

[15] Multitier architecture [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Multitier_architecture. [Dostopano marec 2016].

[16] Thin client [Online]. Dosegljivo: http://en.wikipedia.org/wiki/Thin_client. [Dostopano marec 2016].

[17] MVC Recommended Resources [Online]. Dosegljivo: http://www.asp.net/mvc/overview/getting-started/recommended-resources-for-mvc. [Dostopano marec 2016].

[18] Thin client [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Thin_client. [Dostopano marec 2016].

[19] Client–server model [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Client%E2%80%93server_model. [Dostopano marec 2016].

[20] Entity Framework [Online]. Dosegljivo: https://entityframework.codeplex.com/. [Dostopano marec 2016].

[21] LINQ (Language-Integrated Query) [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/bb397926.aspx. [Dostopano marec 2016].

[22] MySQL Connectors [Online]. Dosegljivo: https://www.mysql.com/products/connector/. [Dostopano marec 2016].

[23] Modular programming [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Modular_programming. [Dostopano marec 2016].

[24] Kludge [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Kludge. [Dostopano marec 2016].

[25] C# Coding Conventions (C# Programming Guide) [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/ff926074.aspx. [Dostopano marec 2016].

[26] Building Your First Web API with MVC 6 [Online]. Dosegljivo: http://docs.asp.net/en/latest/tutorials/first-web-api.html. [Dostopano marec 2016].

[27] Getting Started with ASP.NET Web API 2 [Online]. Dosegljivo: http://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api. [Dostopano marec 2016].

[28] Json.NET [Online]. Dosegljivo: http://www.newtonsoft.com/json. [Dostopano marec 2016].

[29] The Razor View Engine [Online]. Dosegljivo: http://www.codemag.com/article/1103041. [Dostopano marec 2016].

[30] ASP.NET Razor - Markup [Online]. Dosegljivo: http://www.w3schools.com/aspnet/razor_intro.asp. [Dostopano marec 2016].

[31] Luna Blue Icon Set [Online]. Dosegljivo: http://dryicons.com/free-icons/preview/luna-blue-icons/. [Dostopano marec 2016].

[32] DryIcons Free License Agreement [Online]. Dosegljivo: http://dryicons.com/terms/free/. [Dostopano marec 2016].

[33] Download HtmlTextbox for Windows.Forms [Online]. Dosegljivo: http://www.java2s.com/Open-Source/CSharp_Free_Code/HTML/Download_HtmlTextbox_for_Windows_Forms.htm. [Dostopano marec 2016].

PRILOGE

Priloga 1: Izvorna koda nadzorne aplikacije in treh različic spletne trgovine.