

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tadej Abina

**Razvoj spletne aplikacije za podporo organizacije
dogodkov**

DIPLOMSKO DELO
VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE
STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: izr. prof. dr. Marko Bajec

Ljubljana 2012

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani/-a Abina Tadej,

z vpisno številko 63040001,

sem avtor diplomskega dela z naslovom:

Razvoj spletne aplikacije za podporo organizacije dogodkov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Marka Bajca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 26.9.2012

Podpis avtorja:



Št. naloge: 00337/2012

Datum: 04.09.2012

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

Kandidat: **TADEJ ABINA**

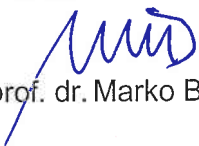
Naslov: **RAZVOJ SPLETNE APLIKACIJE ZA PODPORO ORGANIZACIJE
DOGODKOV**
**DEVELOPMENT OF A WEB APPLICATION FOR ORGANISATION OF
EVENTS**

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Organizacija kompleksnih dogodkov zajema veliko število aktivnosti, ki morajo biti izvedene v relativno kratkih časovnih rokih. V okviru diplomske naloge podrobno preučite področje organizacije raznovrstnih dogodkov ter pripravite načrt za ustrezno informacijsko podporo. V poljubnem razvojnem okolju razvijte spletno aplikacijo, ki bo podpirala organizacijo tako enostavnih kot tudi kompleksnih dogodkov.

Mentor:


prof. dr. Marko Bajec



Dekan:


prof. dr. Nikolaj Zimic

Zahvala

Za pomoč, hiter odziv na vprašanja ter nasvete pri pisanju diplomske naloge se zahvaljujem mentorju izr. prof. Dr. Marku Bajcu.

Posebej pa bi se zahvalil bližnjim, ki so mi bili ves čas študija v vsestransko oporo.

Kazalo

Povzetek	1
Abstract.....	2
1 Uvod.....	3
2 Organizacija dogodkov	4
2.1.1 Posebni dogodki	4
2.1.2 Pomembni pojmi in vprašanja, ki si jih moramo zastavljati.....	6
2.1.3 Časovnica organizacije dogodka	8
2.1.4 Pravni vidik	10
3 Spletna aplikacija za organizacijo posebnih dogodkov	13
3.1 Namen aplikacije	13
3.2 Opis in uporaba.....	13
3.3 Načrtovanje.....	19
3.4 Možne nadgradnje.....	28
4 Implementacija spletne aplikacije.....	29
4.1 SOLID (Objektno orientirano načrtovanje)	29
4.2 Microsoft .NET	30
4.3 Microsoft Visual Studio 2010.....	30
4.4 Shramba za nadzor verzij.....	31
4.5 Osnovna arhitektura aplikacije (ASP.NET MVC 4)	31
4.6 Testno voden razvoj (ang. Test Driven Development).....	35
4.7 ReSharper.....	39
4.8 Razor	40
4.9 Sprotna postavitve na strežniku s pomočjo Powershell in psake.....	41
4.10 SQL Dapper – Preslikava iz podatkovne baze v objekte.....	45
4.11 Večjezičnost.....	46
4.12 Validacija	48
4.13 Obveščanje po elektronski pošti	49
5 Zaključek	51
6 Literatura in viri	52

Seznam uporabljenih kratic in pojmov

Microsoft .NET	Ogrodje za razvoj programske opreme
MVC Model–View–Controller	Arhitekturni vzorec pri razvoju programske opreme
TDD Test Driven Development	Razvijalska tehnika, katere bistvo je pisanje testov skupaj z razvojem same aplikacije
SOLID Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion	Akronim SOLID predstavlja pet osnovnih principov objektno orientiranega programiranja in načrtovanja
ORM Object Relational Mapper	Preslikovalnik med objekti v objektno orientiranem programskem jeziku in relacijskim modelom
NUnit	Odprtokodno ogrodje za testiranje, napisano v programskem jeziku C#
Razor	Motor za poglede iz zadnje verzije MVC arhitekturnega vzorca
ReSharper	Razširitev za Microsoft Visual Studio, ki nam olajša refaktoriranje kode in poveča produktivnost

Povzetek

Organizacija dogodkov je hitro razvijajoče področje, ki je še posebej v vzponu v modernem času, čeprav ga poznamo že iz zgodovine. Dogodki in prireditve so vedno igrale glavno vlogo pri razbijanju monotonosti vsakodnevnega življenja in s tem postale pomemben del vsake družbe. Hiter tehnološki napredek je povečal zmožnosti za organizacijo dogodkov, s čimer pa so se povečale tudi želje in pričakovanja ljudi.

Diplomsko delo vsebuje pregled področja organizacije dogodkov, s svojimi prednostmi in slabostmi. Opisane so razdelitve in različne lastnosti dogodkov, ki jih moramo pri organizaciji upoštevati. Definirani so pojmi, ki jih srečujemo, kadar imamo opravki s prireditvami ter odgovori na vprašanja, ki so pomembna, če želimo zagotoviti uspešno izvedbo našega dogodka. Predstavljen je pravni vidik organizacije, ki nam opiše potrebna dovoljenja, zahteve in omejitve.

Sledi pregled aplikacije, katere namen je olajšanje prej opisanih nalog in zahtev pri organizaciji dogodka. Spletna aplikacija podpira večjezičnost in zagotavlja obveščanje vseh udeležencev preko elektronske pošte. Uporabnikom omogoča, da na enem mestu shranijo podatke o potencialnih nastopajočih, lokacijah dogodkov in možnih načinih promocije. Omogoča nam zbiranje in urejanje vseh entitet, ki bodo sodelovale pri produkciji dogodka. Na enem mestu lahko zberemo povezave na vso potrebno dokumentacijo (dovoljenja, pogodbe,...) ter si organiziramo posamezne dejavnosti s pomočjo opomnikov, ki nam jih aplikacija ponuja.

Na koncu si ogledamo še tehnične vidike implementacije. Uporabili smo programska orodja in principe, ki nam bodo zagotovili dolgoročno vzdrževanje in nadgradnje aplikacije, s čimer se bo le-ta lahko prilagajala spremembam na področju organizacije prireditev.

Ključne besede: organizacija dogodkov, posebni dogodki, .NET, ASP.NET MVC 4, testno voden razvoj

Abstract

Event organization is a fast developing field that has especially been rising in the modern times, even though it has been around for a long time. Special events have always played a leading role in breaking the monotony of everyday life and have therefore become a significant part of every society. Fast technological advancement has introduced new possibilities for event organization, but also increased the wishes and expectations of people.

This document includes an overview of the event organization area with its good and bad sides. It describes the categories and different characteristics of events, that need to be taken into account when organizing one. It defines different terms that we are to encounter when dealing with events and answers questions that are important, if we want to guarantee a successful one. We take a look at the legal side of the issue, which presents the needed permissions, demands and limitations.

The thesis then describes the application, which is meant to make the event organization tasks easier. The web application supports the multilanguage feature and provides means of email notifications for users. It enables the users to store vital information about the artists, venues and advertisement options in one place. It lets us list and edit all the entities that will be included in the production aspect of the event. It also serves as a place where we can gather links to all the necessary documentation (contracts, permits, ...) and let's us organize tasks using the reminders the application provides.

Finally, we take a look at the technical views of the implementation. We used programming tools and principles that will guarantee long term possibilities for maintenance and upgrades. This will ensure our application can adjust to the changes in the field of event organization.

Key words: event organization, special events, .NET, ASP.NET MVC 4, test driven development

1 Uvod

Posebni dogodki so fenomen, ki izhaja iz nerutinskih trenutkov, ki imajo kulturne, osebne ali organizacijske cilje, odmaknjene od normalnih vsakodnevnih aktivnosti, in katerih namen je izobraziti, proslaviti, zabavati ali obogatiti izkušnjo skupine ljudi. [1]

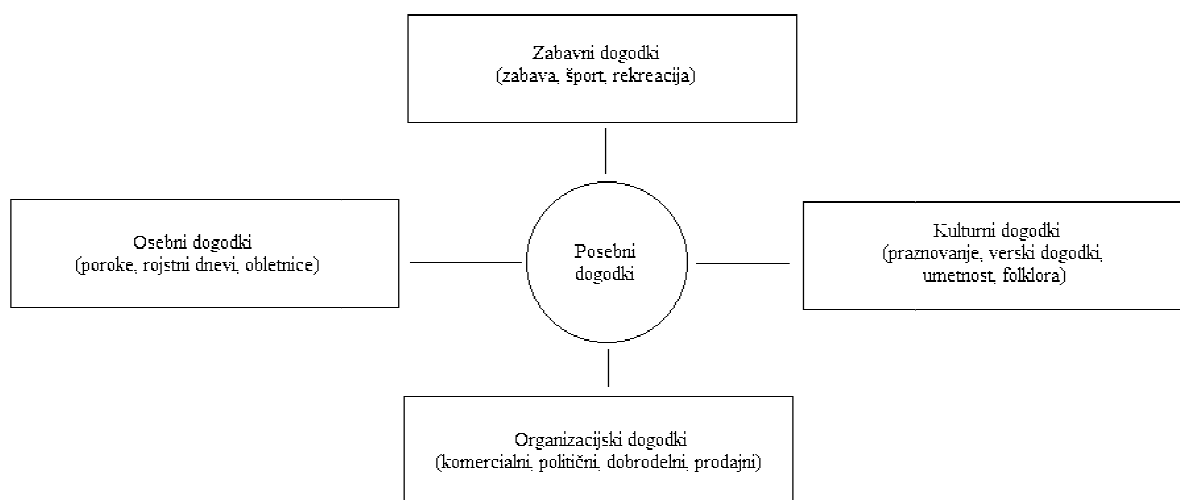
Prireditve že dolgo igrajo pomembno vlogo v človeški družbi. Monotonost vsakodnevnega življenja, polnega truda in skrbi, so ljudje prekinjali s pomočjo prireditev vseh tipov. V večini družb je bil že najmanjši izgovor lahko razlog za proslavitev, čeprav so imele tradicionalne prireditve natančno določena pravila in rituale. V Evropi, še posebej pred industrijsko revolucijo, so bila rutinska vsakodnevna opravila pogosto prepletena s festivali in karnevali. Osebni ali lokalni dogodki, povezani s proslavljanjem verskih praznikov ali določenih trenutkov v letu, so bili prav tako pogosti. Ta vloga družbi je bila, in je še vedno, zelo pomembna. V modernem svetu so se zgodovinske gonilne sile za prireditve spremenile. Verski razlogi za prirejanje velikih festivalov so, na primer, izgubili na pomembnosti, a ne glede na to še vedno vidimo karnivale, sejme in festivale na najrazličnejših mestih, v različnih časih leta. Veliko teh dogodkov, ne glede na to, ali so tradicionalnega ali verskega izvora, igrajo veliko vlogo pri privabljanju turistov, in posledično tudi prihodka, na določen kraj. Nekateri večji dogodki pa se še vedno vrtijo okrog dni kot so Božič in Velika noč v krščanskem koledarju in mesta širom Evrope na te dneve pogosto priredijo ogromne festivale. Z ozirom na vse lahko razumemo, kako pomembni so bili zgodovinsko taki posebni dogodki v socialnem vidiku vsakodnevnega življenja. V dandanašnjih časih smo tako navajeni posebnih dogodkov, da jih ne vidimo več v takem kontekstu. Prav tako je včasih težko za ljudi, ki bi se radi priučili organizacije prireditev, razumeti poln nabor teh aktivnosti, njihovo raznolikost, vlogo in potek. Pri organizaciji prireditev, ne moremo, kot v mnogih drugih industrijah govoriti o vrednosti te industrije na letni ravni. Pravzaprav je skoraj nemogoče definirati vrednosti prireditev v smislu industrije. Taka kalkulacija bi bila zelo problematična, saj je razpon prireditev огromen, od mednarodno organiziranih športnih spektaklov, kot so Olimpijske igre, do manjšega koncerta v vaškem lokalu. Edino, kar lahko razumno pogledamo, je posamezen izoliran dogodek, in za tak dogodek ugotovimo, kakšen je bil skupen izkupiček. Res je, da imajo nekateri dogodki namen ustvariti dobiček ali neko dodano vrednost, kot tudi zabavati in izobraziti občinstvo, vendar to še zdaleč niso edini razlogi za organizacijo.

V naši diplomski nalogi se bomo osredotočili bolj na dogodke zabavnega tipa, kot so glasbeni koncerti, nastopi komikov, večdnevni festivali ipd. Organizacija omenjenih dogodkov vključuje identifikacijo ciljnega občinstva, razvijanje koncepta dogodka, načrtovanje logistik, koordinacijo tehničnih vidikov, analizo po samem dogodku, upravljanjem s proračunom, pridobivanje dovoljenj, pogajanja in še mnogo. Nedavna rast festivalov in dogodkov kot industrije po celem svetu pomeni, da organizacija ni več samo hobi. Dogodki, predvsem tisti večjega tipa, imajo lahko velik vpliv na lokalno skupnost, ali celo na celotno državo. Organizacija dogodkov kot kariera je relativno novo področje. Sedaj obstajajo tečaji, ki pomagajo organizatorjem obvladovati pritisk in jim pomagajo pri učinkoviti izvedbi dela. To področje dela se smatra za hitro, stresno in zahtevno, a hkrati razburljivo. Organizator se srečuje z ogromno koordinacije, komunikacije in časovnimi roki, prav tako pa je veliko število nadur in dela za vikende nekaj običajnega. [3]

2 Organizacija dogodkov

2.1.1 Posebni dogodki

Posebne dogodke lahko načeloma razdelimo v štiri širše kategorije (prikazane na sliki 1), definirane glede na cilje posameznega dogodka [1]. Vendar pa moramo razumeti, da se te kategorije pogosto prepletajo. Maturantski ples je, na primer, za dijaka in njegovo družino osebni dogodek, medtem ko je za šolo organizacijski. Vaški karnival je tako kulturni dogodek, ki lahko praznuje nek vaški praznik, kot tudi zabavni dogodek, tako za lokalne prebivalce, kot za turiste. Zato moramo na ta prekrivanja gledati kot neizogibna, ne kot na izjeme.



Slika 1 - Priporočena kategorizacija posebnih dogodkov [1]

Posebni dogodki imajo veliko lastnosti, ki so skupne vsem vrstam storitev, še posebej pa so blizu različnim tipom storitev, namenjenim zabavi. Glavne karakteristike posebnih dogodkov so naslednje [1]:

Unikatnost

Dogodek mora ljudi privabiti s svojo posebnostjo in deviacijo od vsakodnevnega dogajanja. Pri tem pazimo, da se ne pojavljamo, dobro premisliti, katere nastopajoče bomo povabili, kakšno temo dogodka si bomo izbrali, poskusimo pridobiti lokacije, ki jih ljudje niso vajeni, program zastaviti na način, da bo ljudi pritegnil in tudi držal njihovo pozornost. Pomembna je seveda tudi izbira datuma, pri čemer pazimo, poleg zasedenosti zelenih lokacij in nastopajočih, tudi na ostale dogodke na isti dan. Le-ti naj se ne prekrivajo po tematiki, saj na ta način lahko izgubimo potencialne obiskovalce.

Ambient

Med vsemi karakteristikami dogodkov, je ambient najbolj pomemben za dober splošen vtis. Dogodek z dobrim ambientom je lahko zelo uspešen, dogodek s slabim pa polomija. Pri osebnih dogodkih, kot je rojstnodnevna zabava, je ambient ponavadi ustvarjen že s strani samih udeležencev, brez potrebe po dodatnem trudu. Pri večjih, javnih dogodkih, pa je potrebno za dober ambient poskrbeti za še za primerne vizualne dobrate, glasbo ter dodatne

opcije za zabavo. Na žalost tudi prisotnost teh elementov sama po sebi še ne zagotavlja dobrega končnega rezultata. Ravno zaradi tega je ta vidik dogodka eden izmed pomembnejših izzivov organizatorja.

Interakcija

Pri storitvah, med katere spada tudi organizacija dogodkov, imajo stranke, v našem primeru obiskovalci, pogost stik z osebjem, kar pogosto določi kvaliteto obiskovalčeve izkušnje. Ljudje, ki obiskujejo dogodke, so velikokrat tudi sami del tega procesa. Množica na koncertu, na primer, ne le spremlja dogodek, ampak tudi sama sodeluje pri ustvarjanju atmosfere - množica sodeluje z nastopajočimi in je sama del celotne izkušnje. Zato se mora organizator pri načrtovanju zavedati, da je uspeh dogodka v veliki meri odvisen od akcij in reakcij obiskovalcev.

Trdo delo

Večji in bolj unikaten, kot je dogodek, večja je verjetnost, da bo s strani organizatorjev potrebno veliko trdega dela, tako v smislu priprave, kot tudi same izvedbe. Potrebno je veliko načrtovanja in dobre komunikacije med sodelujočimi, zato se velikokrat posamezne naloge zaupa skupini strokovnjakov. Za samo izvedbo je potrebno veliko število ljudi. Če naprimer organiziramo koncert, potrebujemo tehnike, fizične delavce, natakraje, ljudi, ki bodo skrbeli za hrano in še mnogo drugih. Količino dela in potrebne pomoči je pri posebnih dogodkih težko napovedati, zato se organizatorji pri teh odločitvah zanašajo predvsem na izkušnje.

Fiksen urnik

Organizacijske aktivnosti pred in na dan dogodka tečejo po točno določenem urniku. Organizatorji in osebje se morajo držati dogovorjenih rokov in časovnih intervalov, da zagotovijo uspešno izvedbo dogodka. Urnik mora biti za dan dogodka premišljen tako, da imamo nekaj prostora v primeru, da pride do nenačrtovanih situacij.

Vsak, ki se, ali bi se rad ukvarjal z organizacijo dogodkov si mora zapomniti predvsem eno stvar: *Nihče se ne rodi za organizatorja dogodkov.* [2] S tem v mislih se moramo zavedati, da nihče na pamet ne ve, če prostor ima podporno konstrukcijo za obešanje luči ali zvočnega sistema, prav tako ne ve, kako najbolje skonfigurirati dvorano. Poleg tega je človek lahko kaj kmalu preobremenjen s strani količine dela, ki je potrebno za organizacijo uspešnega dogodka. Posel organizacije prirediteljev se spreminja iz trenutka v trenutek in nihče, ne glede na to, kako dolgo se že s tem ukvarja, ne more vedeti vsega. Ogromno časa, dela in truda je potrebno, že preden se dogodek potrdi. Ponudbe in nasprotne ponudbe, spremembe datumov in zasedenost lokacij so samo nekatere skrbi, s katerim moramo biti seznanjeni. Toda pravo delo se začne šele po sami potrditvi dogodka. Takrat se pojavi celo morje odgovornosti in podrobnosti, za večino katerih je potrebno poskrbeti kar nekaj časa pred datumom dogodka. Ne glede na to, če je nekogaršnje delo okrog organizacije zaključeno že tedne pred dogodkom, pa si velja zapomniti, da na dan dogodka delajo vsi sodelujoči, saj ni pomoči nikoli premalo.

Nastopajoči in sami dogodki postajajo s časom vse dražji. To lahko pomeni manj priložnosti za organizatorje, lahko pa tudi pomeni manj datumov za nastope na strani nastopajočih. Vendar to ne pomeni nič drugega, kot to, da si moramo vzeti več časa za načrtovanje proračuna, določitve zelenih ciljev ter opredeliti stroške za produkcijski del organizacije. Preden se odločimo, katere nastopajoče bi radi in kako bi radi, da dogodek izgleda, moramo vzeti v zakup stroške produkcije dogodka. To je pri organizaciji trenutno najbolj pogosto vprašanje in hkrati tudi najbolj pomembno. Ne glede na to, če imamo dobre namene in cilje,

brez finančne moči, da podpremo ta vidik organizacije, ne bomo imeli veliko opcij. Na žalost na to vprašanje ne obstaja konkreten odgovor. Vsak dogodek je različen, kot so različne tudi lokacije in zahteve nastopajočih. V vsakem pogledu pa je produkcija največji strošek organizacije. Pod njo spadajo vsi fizični vidiki našega dogodka. Strošek produkcije je tesno povezan z lokacijo. Dogodki v zaprtih prostorih so načeloma veliko cenejši od dogodkov na prostem. Odvisno od tipa zaprtega prostora imamo lahko na voljo že oder, sedišča, v najboljšem primeru tudi sanitarije, zvočno in svetlobno opremo. Stroški produkcije se nam v takih primerih lahko zmanjšajo tudi samo na najem takega prostora. Nasprotno pa v primeru dogodka na prostem naš proračun dodatno obremenijo gradnja ali najem odra, luči, glasbene opreme, sanitarij, generatorja električne energije, po možnosti premična streha ipd. Poleg produkcije moramo vzeti v zakup še stroške nastopajočih, ki niso direktno povezani s samim nastopom. Poskrbeti je potrebno za njihove dodatne zahteve, kot tudi za hrano in pijačo, kar lahko predvsem pri skupinah z njihovo spremljevalno ekipo postane velik izdatek. Cena varnostni na dogodku je odvisna od velikosti in vrste le-tega. Ponavadi je priporočljivo ali celo obvezno najeti profesionalne varnostnike. Pri velikih dogodkih zabavne narave (glasbeni koncerti), kjer je pričakovano število obiskovalcev v tisočih je pri nas obvezna tudi prisotnost DrogArt ekipe in vsaj enega reševalnega vozila. Nazadnje pa je tu še strošek promocije. Ta je zopet povsem odvisen od velikosti dogodka, ciljnega občinstva in uporabljene strategije. Če upoštevamo še nepričakovane stroške in nekaj dodatnega denarja za dan dogodka, je okvirna cena za dogodek večjega tipa je nekje med 15 in 20 tisočimi evri brez stroškov nastopajočih.

Tu se pojavi vprašanje, kakšne nastopajoče izbrati. V veliki meri je naša izbira pogojena že s strani izbrane lokacije, saj le-ta mogoče ne ponuja dovolj prostora ali tehničnih sposobnosti za dogodke določenega tipa. Druga omejitev je datum našega dogodka, saj ni nujno, da bo izbrana skupina ali drugi nastopajoči ta termin imel prost. Prilagoditi se je potrebno tudi na želje ciljnega občinstva, saj moramo v primeru, da želimo privabiti veliko število ljudi, izbrati bolj znano točko kot v primeru dogodka zaprtega tipa. Posledično narasta tudi cena. Pri promociji veliko pomeni ustvarjalnost in iznajdljivost. TV ali radio oglasi znajo biti zelo dragi, lahko pa se z pristojnimi dogovorimo za kompenzacijo v obliki nekega števila vstopnic za naš dogodek. Poskusimo si zagotoviti čim večjo medijsko pokritost za čim nižjo ceno. Dogodek lahko promoviramo tudi z nagradnimi igrami za brezplačen vstopnice. Tako kot pri kompenzacijah pa moramo paziti na število vstopnic, ki jih podelimo brezplačno, predvsem v primeru omejenih kapacitet. Kar se tiče vstopnic, ki jih bomo prodali, moram zagotoviti, da lahko potencialni obiskovalci do njih pridejo na čim lažji način. Če jim bomo dostop do vstopnic otežili, obstaja velika verjetnost, da bo obiska občutno manj. Zato poskušamo našo mrežo prodajalnih čimbolj razširiti - od lokalov, glasbenih trgovin, študentskih servisov ipd. Najlažje in najbolj učinkovit način (vendar ponavadi tudi najdražji) pa je distribucija vstopnic preko interneta. Spletna prodajalna bo sicer vzela svoj delež denarja od vsake prodane vstopnice, vendar pa bo posledično lahko vsak kupec le-to pridobil kar od doma. Pri kritju vseh stroškov organizacije nam poleg prodaje vstopnic lahko pomagajo sponzorji, javni razpisi za sredstva, uporaba prostovoljcev, kjer je to le možno, kompenzacije itd.

2.1.2 Pomembni pojmi in vprašanja, ki si jih moramo zastavljati

V nadaljevanju bo opisano nekaj pomembnih pojmov, ki se pojavljajo pri pri produkciji komičnih, glasbenih in drugih dogodkov [2].

Prepustnice: Prepustnice so način za organizatorje, novinarje, nastopajoče in druge vpletene, da se na dan dogodka lahko prosto premikajo po lokaciji, seveda odvisno od vrste prepustnice. Prepustnice s polnim dostopom nam omogočajo dostop do vsega, zato jih moramo razdeliti razumno samo ljudem, ki jih res potrebujejo. Prepustnice za zaodrje

dovoljujejo dostop do zaodrja. Načeloma se izdajo celotni ekipi, ki sodeluje pri organizaciji na sam dan dogodka. Novinarske prepustnice se izdajajo novinarjem in fotografom, ki bodo pokrivali naš dogodek.

Bonusi: Po dogovoru dodaten denar, ki ga dobi nastopajoči, glede na prodajo vstopnic ali ustvarjenega zaslužka.

Ponudbe: Denar, ki je ponujen nastopajočim v zameno za nastop. Ponudbe naj se podajajo v pisni obliki z vsemi podrobnostmi dogodka. Paziti moramo na to, da so ponudbe v velikih primerih obvezujoče in da sama ponudba še ne pomeni, da jo bo želeni nastopajoči tudi sprejel.

Nična točka (ang. break even point): Točka, ko je zaslužek od prodanih vstopnic enak vsem izdatkom, ki so se pojavili pri organizaciji.

Dokumentacija: Zbirka vseh dokumentov, ki jih potrebujemo pri organizaciji. Pomembno je, da imamo čimveč stvari v pisni obliki in na enem mestu.

Maksimalen izkupiček: Najvišji možni zaslužek dogodka. Pomnožena najvišja cena vstopnic s kapaciteto izbrane lokacije. Podatek je pomemben pri določanju opsijskega bonusa za nastopajoče.

Lokacija: Lokacija ali lokacije našega dogodka. Od vrste le-te je odvisna večina odločitev, ki jih moramo sprejeti pri organizaciji.

Prodajna mesta vstopnic: Mesta, kjer ljudje lahko kupijo vstopnice za naš dogodek, če so te seveda mišljene. Ponavadi je cilj čim večje število prodajnih mest.

Produkcija: Fizični vidiki organizacije, kot so odri, luči, elektrika, varnostne ograje ipd.

Pogodba: Legalno zavezujoč dokument, v katerem so navedene podrobnosti našega dogodka v povezavi z nastopajočimi. Pogodbe se lahko spreminjajo samo, če se strinjata obe vpleteni stranki.

Tekač: Oseba, ki je ves dan na voljo za urejanje opravkov. Večina nastopajočih, predvsem glasbenih skupin, zahteva, da jim je na dan nastopa na voljo tekač z vozilom.

Zahteve nastopajočih (and. rider): Te ponavadi spremljajo pogodbo z nastopajočimi in navajajo posebne zahteve le-teh. Vsebujejo podatke o potrebni produkciji, hrani, ki ji nastopajoči zahteva in druge podrobnosti. Podobno kot pogodbe, se lahko zahteve spremenijo samo ob sporazumu obeh vpletenih strank. Vseeno pa so te zahteve v nekaterih primerih bolj želje in so stvar pogajanj.

Varnost: Varnost se pojavlja v različnih oblikah, od rediteljev, do profesionalnih varnostnikov, lahko tudi policije. Ponavadi imajo posebne zahteve po varnosti lahko tudi nastopajoči ali lastniki lokacije, kjer se dogodek odvija.

Organizacija večjega dogodka je dolg proces, ki zahteva veliko posameznikovega časa. Glede na to, da je ponavadi proračun omejen, je potrebno sprejemati pametne odločitve, ki ga ne bodo presegle in bodo hkrati izpolnile vse cilje organizatorjev. Zato si je treba cilje natančno določiti, narediti načrt za njihovo doseg, raziskati čim večje število možnosti in svoj načrt tudi izvesti. Večina organizatorjev ima v mislih naslednje cilje: prirediti uspešen in odmeven dogodek, zabavati in izobraziti občinstvo ter tudi samega sebe, ali pa samo doseči nično točko. Identifikacija primernih in sekundarnih ciljev nam pomaga pri izbiri nastopajočih in pri odločitvah, kje porabiti denar, ki je na voljo. Vprašati se moramo tudi naslednje [2]:

Tip dogodka: Ali bo dogodek brezplačen, ali bomo prodajali vstopnice? Bo dogodek zaprtega tipa, ali bo odprt za širše občinstvo? Je potrebno z dogodkom zaslužiti, je dovolj pokritje vseh stroškov, ali finance niso problematične? Ali je potreba po točno določenem tipu dogodka (koncert rock glasbe, veselica, predstava...)?

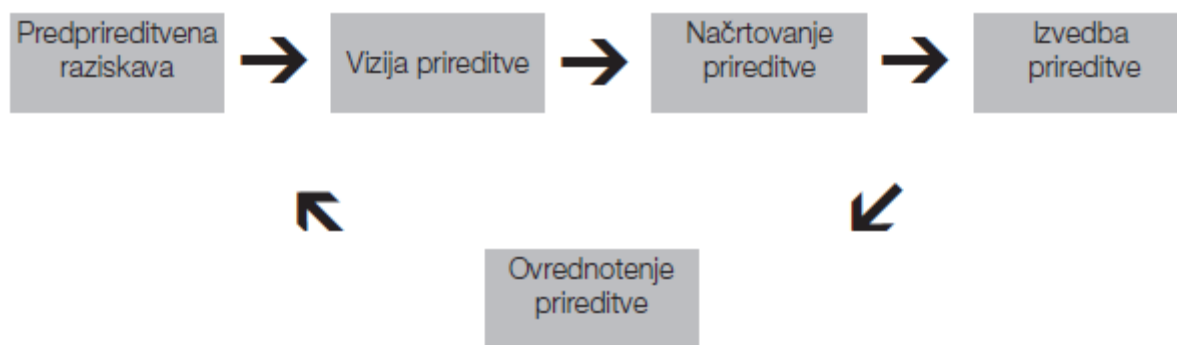
Izbira datuma: Več kot imamo na voljo datumov, večje imamo možnosti, da dobimo nastopajoče, ki smo jih izbrali. Poskušamo pridobiti čim več možnih datumov na naši željeni lokaciji in pridobiti pisno potrditev le-teh.

Lokacija: Ali je oder že na lokaciji, ali ga bo potrebno najeti? Je dovolj električne energije, ali bo potrebno zagotoviti dodatne generatorje? Ali je na lokaciji na voljo električar ter ali bo na voljo na dan dogodka? Kakšno je največje število ljudi, ki lahko vstopijo na izbrano lokacijo? Če imamo potrebo po sedežih, je potrebno vedeti, če so le-ti na voljo, v nasprotnem primeru jih je potrebno zagotoviti. Ali ima lokacija garderobe? V primeru, da organiziramo dogodek na prostem nas zanima tudi, če imamo alternativno lokacijo v primeru slabega vremena.

Proračun: Upoštevati moramo vsaj naslednje - cena nastopajočih in njihovih dodatnih zahtev, produkcije, hrane in pijače za vpletene, oglaševanja, varnosti, tiskanje vstopnic, najem lokacije ter vse druge pričakovane in nepričakovane stroške.

2.1.3 Časovnica organizacije dogodka

Priporočljivo je, da se organizacija začne veliko pred samim dogodkom, odvisno od velikosti le-tega. Pri večjih dogodkih lahko ta proces traja mesece. Osnovni proces organizacije dogodka je prikazan spodaj (slika 2).



Slika 2 - Osnovni proces organizacije prireditve, oziroma dogodka [4]

V smislu aktivnosti organizatorja pa lahko organizacijo dogodka razdelimo na štiri faze [2].

Predprireditveno načrtovanje

Najprej izberemo datume za naš dogodek. Kadarkoli je to možno, določimo tudi rezervne datume. Poskusimo pridobiti pisne potrditve rezervacij zelenih lokacij za vsakega od možnih datumov in se prepričamo, da to pomeni, da imamo lokacijo na voljo tekom celega dneva. Preverimo proračun in potrdimo vsa sredstva pri za to pristojnih ljudeh (ponavadi sponzorjih). Pustimo si levji delež denarja, ki je na voljo za produkcijske stroške, kot so zvočna oprema, odri, hrana itd. Naredimo si seznam nastopajočih, ki jih želimo za naš dogodek in jih razvrstimo po naših željah. Glede na naše pridobljeno znanje o lokaciji dogodka (velikost odra, elektrika, dovoljenja ipd.) ugotovimo možne težave s tehničnimi zahtevami izbranih nastopajočih. Nato v pisni obliki pošljemo ponudbo izbranim nastopajočim in pridobimo potrditev, da je bila naša ponudba sprejeta.

Po potrditvi datuma

Naredimo načrt naše promocije. Uporabimo vse vire, ki so nam na voljo, in naredimo intenzivno in dolgo trajajočo kampanjo. Izberemo in organiziramo naše osebe. Prepričamo

se, da je osebje dovolj veliko in sposobno. Pri organizaciji dogodkov ni sodelujočih nikoli preveč. Naročimo izdelavo vstopnic (če seveda dogodek ni brezplačen) in poskrbimo za njihova prodajna mesta. Poskrbimo za varnost na dan dogodka preko rediteljev, profesionalnih varnostnikov, ali celo policije, ter pridobimo vsa potrebna dovoljenja. Ko prispejo pogodbe s strani nastopajočih ali katerekoli druge vpletene stranke, jih podrobno pregledamo in se odločimo ter pogajamo o potrebnih spremembah, ponavadi predvsem glede dodatnih zahtev nastopajočih.

Pred dogodkom

Redno spremljamo, kako poteka prodaja vstopnic in glede na pridobljene podatke prilagajamo promocijo. Pazimo na vstopnice, ki ne bodo šle v prodajo (vstopnice, ki jih zahteva nastopajoči ali katera druga vpletena stranka v zameno za svoje storitve). Prepričamo se, da bomo na dan dogodka imeli na voljo dovolj denarja, da plačamo nastopajoče. Naredimo si časovni načrt za dan dogodka in potrdimo prihode prodajalcev vstopnic, varnostnikov, električarjev, tehnikov in drugih ob dogovorjenem času. Gostimo podroben organizacijski sestanek z našim osebjem.

Na dan dogodka

Seznamimo osebje z vsemi podrobnostmi, jim zagotovimo dostop do potrebnih informacij v vsakem danem trenutku ter razdelimo propustnice. Ustavimo prodajo vstopnic na vseh prodajnih mestih (razen na vhodu na dogodek) ter neprodane vstopnice shranimo v blagajno. Sledimo napredku produkcije ter skrbimo, da gre vse po načrtu. Prepričamo se, da je osebje za pospravljanje na voljo po koncu dogodka. Ves čas pa moramo biti pripravljeni na prilagoditev naših načrtov z ozirom na nepričakovane prijetljaje.

Postopek si pogledjmo še malce bolj podrobno v smislu časovnih okvirov [2].

4 mesece pred dogodkom: Odločimo se, kakšen dogodek bomo sploh organizirali in si zastavimo cilje.

3-4 mesece prej: Pridobimo ekipo ljudi, ki nam bodo pomagali pri organizaciji ter si zagotovimo pisno rezervacijo lokacije našega dogodka. Poskušamo čim bolj razdelati okvirne stroške organizacije in izvedbe, da vidimo, če je dogodek sploh finančno smiseln. Na podlagi ugotovitev določimo cene vstopnic.

3 mesece prej: Ponovno potrdimo rezervacijo lokacije in še enkrat preverimo proračun. Naredimo si načrt za promocijo. V tem trenutku tudi pošljemo ponudbe za nastopajoče, v katerih poskusimo čimbolj podrobno opisati naš dogodek, tudi kar se tiče podrobnosti. V ponudbah natančno definiramo roke za podpise pogodb in možne spremembe ter tudi navedemo možne težave, da se s tem izognemo morebitnim nesporazumom kasneje. Ko potrdimo nastopajoče, o tem takoj obvestimo lastnika lokacije našega dogodka, naše osebje ter začnemo s promocijo. Pridobimo si vsa potrebna dovoljenja s strani vseh pristojnih strank ter uredimo pogodbe. Pridobimo vse potrebne kontakte za uspešno izvedbo produkcije. To pomeni, da poiščemo ponudnike odrov, luči, glasbene opreme, kulis, varnostnih ograj itd. Na enem mestu zberemo kontakte omenjenih oseb. Poskrbimo še za varnost tako, da se odločimo, koliko in kakšne tipe varnosti potrebujemo ter nato ta vidik tudi uredimo.

1-2 meseca prej: Medtem, ko se bodo obiskovalci mogoče vozili eno uro ali več na sam dogodek, tega zagotovo ne bodo storili za nakup vstopnic, zato zagotovimo, da so te čim lažje dostopne. Glede na potek prodaje vstopnic se odločimo za morebitne spremembe v promociji. Še enkrat pregledamo vse pogodbe in se ponovno prepričamo, da vsi naši dogovori in rezervacije še veljajo.

1 mesec prej: Če katerih pogodb še nismo prejeli, pošljemo pisno opozorilo s točno določenimi roki. Začnemo določati čase za posamezne dele organizacije na dan dogodka ter sestavimo ekipo za delo na ta dan.

2-3 tedne prej: Ponovno potrdimo vse dogovore, pregledamo pogodbe in glede na prodajo vstopnic prilagodimo promocijo.

1-2 tedna prej: Z nastopajočimi potrdimo čas prihoda, tonskih vaj in ostale podrobnosti ter priskrbimo vse potrebne propustnice.

1-2 dni prej: Potrdimo prihode našega osebja in poskrbimo za komunikacijo med njimi ter si priskrbimo si nekaj dodatnega denarja za izredne situacije. Še enkrat preverimo vse vidike, ki bodo igrali vlogo na dan dogodka:

- Ali so luči, odri, elektrika, varnostne ograje itd. na mestu, ali pa vsaj rezervirane in bodo na mestu, ko bo to potrebno?
- Prihodi osebja in nastopajočih.
- Imamo denar za nastopajoče in nekaj dodatnega za izredne situacije?
- Je hrana in pijača pripravljena?
- Je varnosti dovolj in kdaj bodo varnostni elementi prispeli?
- Ali se bodo vstopnice prodajale tudi na dan dogodka? Ali imamo dovolj menjalnine?
- Imamo pripravljen plan dela za dan dogodka in ali se ga bo naše osebje držalo?
- Ali imamo urejena vsa potrebna dovoljenja in podpisane ter potrjene vse pogodbe?

Na dan dogodka: Predvsem ostanimo mirni in bodimo pripravljeni na improvizacijo.

2.1.4 Pravni vidik

Zakon o javnih zbiranjih (ZJZ) je temeljni predpis, ki ureja način uresničevanja ustavne pravice do mirnega zbiranja in javnih zborovanj na javnih shodih. Poleg tega to področje urejajo tudi ostali nacionalni pravni viri, ki opredeljujejo področje javnih zbiranj (shodov in prireditev) [4]:

1. Zakon o javnih zbiranjih
2. Zakon o varnosti cestnega prometa
3. Zakon o volilni in referendumski kampanji
4. Zakon o varstvu pred utopitvami
5. Zakon o varnosti na smučiščih
6. Zakon o veterinarski skladnosti
7. Zakon o varstvu podzemnih jam
8. Pravilnik za izvajanje zakona o javnih zbiranjih
9. Uredba o splošnih ukrepih v športnih objektih na športnih prireditvah

Javni shod oz. javna prireditev je vsako organizirano zbiranje ljudi na prostem ali v zaprtem prostoru, ki ima določenega organizatorja in vnaprej določen program, udeležba pa je brezpogojno ali pod določenimi pogoji dovoljena vsakomur.

Javno prireditev se organizira z namenom izvajanja kulturne, športne, zabavne, izobraževalne, verske ali druge aktivnosti. [4]

Za legalno organizacijo našega dogodka, je le-tega potrebno prijaviti in pridobiti dovoljenje. Organizator prijavi shod ali prireditev najmanj tri dni pred dnevom shoda oz. najmanj pet dni pred dnevom prireditve na policijski postaji, na območju katere se omenjena prireditev organizira. Za prijavo shoda ali prireditve ni potrebno plačati upravne takse. Vlogo za izdajo dovoljenja pa organizator poda pri upravni enoti, na območju katere se prireditev organizira, in sice najmanj sedem dni pred dnevom le-te. Če pa gre za prireditev na cesti, pa je ta rok 30 dni pred prireditvijo. Za vlogo za izdajo dovoljenje je potrebno plačati upravno takso. Za vse

shode in prireditve velja, da je prijava obvezna. Edina izjema so prireditve, za katere Zakon o javnem zbiranju določa, da jih ni treba prijaviti (npr. spontani ulični nastopi, svečanosti ob rojstvu in poroki, jubileji in podobna zbiranja). V primeru, ko gre za prireditev, za katero je potrebno dovoljenje, pristojna policijska postaja ne sprejme prijave, temveč organizatorja napoti k pristojnemu organu. Če je prijavo shoda ali prireditve, za katero je potrebno dovoljenje, organizator poslal po pošti, jo pristojna policijska postaja nemudoma pošlje pristojnemu organu in o tem obvesti organizatorja. Shodi in prireditve, za katere je potrebno dovoljenje, so naslednji:

- shodi in prireditve na cesti
- prireditve, kjer obstaja nevarnost utopitve
- prireditve z zbiranjem živali
- mednarodne športne prireditve
- tekmovanja klubov v prvi državni ligi članic, oziroma članov v kolektivnih športih
- prireditve, na katerih se uporablja odprt ogenj
- prireditve, na katerih se uporabljajo predmeti, zaradi katerih je lahko ogroženo življenje ali zdravje ljudi, oziroma premoženje
- prireditve, na katerih se uporabljajo naprave, zaradi katerih je lahko ogroženo življenje ali zdravje ljudi, oziroma premoženje
- prireditve, na katerih se pričakuje več kot 3.000 udeležencev

V prijavi shoda je potrebno navesti:

- podatke o organizatorju shoda ali prireditve (osebno ime, EMŠO ali datum rojstva in spol, če mu EMŠO ni določen, državljanstvo, naslov stalnega ali začasnega prebivališča, oziroma firma, sedež, matična številka in osebno ime zastopnice ali zastopnika).
- kraj, čas in trajanje shoda ali prireditve, program shoda ali prireditve in predvideno število udeležencev.
- osebne podatke vodje shoda ali prireditve in vodje rediteljev (osebno ime, EMŠO ali datum rojstva in spol, če mu EMŠO ni določen, državljanstvo, naslov stalnega ali začasnega prebivališča); če rediteljsko službo opravlja subjekt, ki se ukvarja z dejavnostjo varovanja ljudi in premoženja, pa tudi njegove podatke (firma, sedež, matična številka, osebno ime odgovorne osebe).
- osebne podatke pooblaščenke ali pooblaščenca organizatorja za vročitev - na sedežu pristojnega organa (osebno ime, naslov stalnega ali začasnega prebivališča).
- način varovanja shoda ali prireditve z navedbo števila rediteljev ter ukrepov za zagotovitev reda, varnosti življenja in zdravja udeležencev in drugih oseb, varnosti premoženja, da ne bo ogrožen javni promet in da ne bo nedopustno obremenjeno okolje.

Prijavi je treba priložiti še:

- soglasje lastnice ali lastnika, upravljavke ali upravljavca zemljišča ali prostora, na katerem se organizira shod ali prireditev;
- dokazilo, da je o shodu ali prireditvi pisno obveščena lokalna skupnost;
- če se na shodu ali prireditvi z zvočnimi ali drugimi napravami povzroča čezmeren hrup, tudi dovoljenje pristojnega občinskega organa.

Ko organiziramo koncert ali kakršno koli prireditev, kjer se bo izvajala živa glasba, moramo v roku osmih dni pred koncertom ali prireditvijo le-to prijaviti Združenju SAZAS (Združenje skladateljev in avtorjev za zaščito glasbene avtorske pravice Slovenije).

Združenje SAZAS nam bo na podlagi prijave izdalo dovoljenje, ki nas obvezuje, da v roku petnajstih dni po prireditvi Združenju SAZAS dostavimo pravilno izpolnjen spored izvedenih glasbenih del na prireditvi in podatke o ustvarjenem prihodku od prireditve. Na podlagi

sporočenih podatkov bo Združenje SAZAS izračunalo našo obveznost nadomestila za avtorske pravice.

Združenju SAZAS je potrebno poslati naslednje obrazce [5]:

- Obrazec koncerti, prireditve, s katerim koncert prijavimo. Ta obrazec moramo poslati najpozneje osem dni pred koncertom, oziroma prireditvijo.
- Spored izvedenih del za javno izvajanje del na koncertih in prireditvah (obrazec SAZAS-1). V ta obrazec je treba vnesti spisek vseh izvajanih skladb. Obrazec izpolnijo člani posamezne glasbene skupine na dan prireditve. Nato ga (prvi delovni dan po prireditvi) pošljemo priporočeno na sedež Združenja SAZAS. Pri izpolnjevanju obrazca bodimo pozorni, da za vsako glasbeno skupino izpolnimo svoj obrazec (to sicer ni pravilo, je pa zelo priporočljivo, če ne želimo zapletov) ter za glasbo in besedilo skladbe navedemo imena in priimke glasbenikov, za aranžma pa ime glasbene skupine.
- Osnove za obračun avtorskega honorarja (obrazec SAZAS-3). Ta obrazec je potrebno poslati v petnajstih dneh po prireditvi.

3 Spletna aplikacija za organizacijo posebnih dogodkov

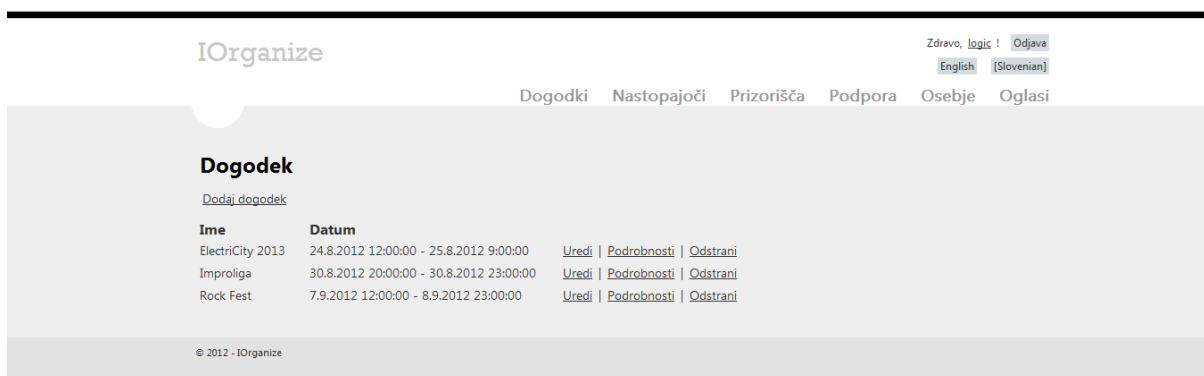
Naša spletna aplikacija je namenjena organizatorjem dogodkov, ne glede na njihovo vlogo v konkretnem primeru dogodka. Učinkovito jo lahko uporablja celotna ekipa, od oblikovalcev do oglaševalcev. Ker ponavadi organizacija večjega dogodka traja dalj časa, lahko mesece, se pomembne informacije vmes lahko izgubijo ali pozabijo, do še več takih neprijetnih pripetljajev pa pride po koncu dogodka. Prav tako je pomembno, da imajo zavoljo usklajevanja vsi udeleženi pri organizaciji v vsakem danem trenutku popoln pregled nad trenutnim stanjem organizacije.

3.1 Namen aplikacije

Z našo aplikacijo smo želeli zagotoviti čimbolj pokriti vse vidike organizacije nekega konkretnega dogodka. Pregled smo hoteli omogočiti celotni ekipi, ki sodeluje pri organizaciji, ter s tem zagotoviti transparentnost in čim tesnejšo povezanost. Tako imajo vsi člani pregled nad možnimi nastopajočimi, ki jih lahko tudi dodajajo, urejajo in brišejo na podlagi pridobljenih informacij. Tu so še vsi podporni elementi - lokacije, različne vrste podpornih dejavnosti, osebje, prostovoljci, dokumenti, oglaševanje in opomniki. Vse skupaj zagotavlja vse potrebne informacije na enem mestu za vse sodelujoče. Le-ti imajo pregled tudi nad trenutnim finančnim stanjem projekta z vsemi izdatki in prihodki. S pomočjo opomnikov in obveščanja prek elektronske pošte o le-teh se prepreči pozabljanje na pomembne sestanke, opravila ipd. To je še posebej pomembno na sam dan dogodka, saj je takrat kritičnega pomena, da vsak od vpletenih izpolnjuje svoje naloge ažurno, drugače lahko kaj hitro pride do težav. Kar se tiče same implementacije aplikacije, smo skušali uporabiti čim novejšo tehnologijo ter aplikacijo narediti berljivo in čim lažje nadgradljivo.

3.2 Opis in uporaba

Uporabniki naše aplikacije bodo prihajali iz različnih področij (stiki z javnostjo, oblikovanje, oglaševalci...), zato smo poskušali aplikacijo narediti karseda razumljivo in preprosto za uporabo. Prva stran aplikacije je prikazana na sliki 3.

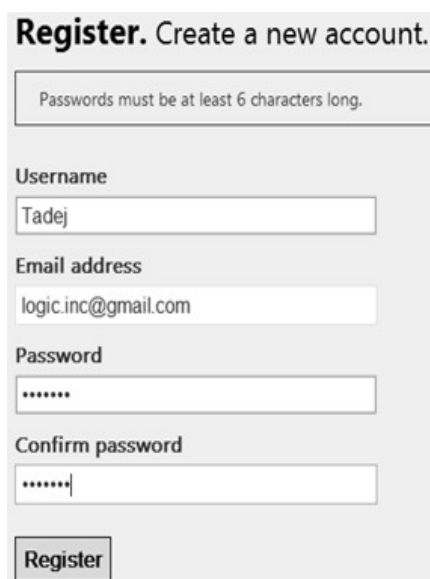


Slika 3 - Prva stran aplikacije

Registracija in prijava

Ob prvem obisku naše aplikacije se mora uporabnik registrirati (slika 4) ter ob vseh nadaljnjih dostopih prijaviti. S tem smo preprečili neavtorizirane dostope do naših podatkov. Aplikacija nas ob vsakem takem dostopu preusmeri na stran za prijavo. Tako delovanje smo dosegli z

uporabo že vgrajene avtentikacije v ogrodju ASP.NET MVC 4, ki bo podrobneje opisan v kasnejših poglavjih.



Slika 4 - Registracija novega uporabnika

Izbira jezika

Da bi bila aplikacija čimbolj dostopna za vse udeležence pri organizaciji, smo se odločili implementirati podporo za več jezikov. Uporabnik lahko kadarkoli v desnem zgornjem kotu izbere želen jezik (slika 5). V trenutno verziji sta podprta in prevedena dva jezika, in sicer slovenščina ter angleščina, ki je tudi privzet jezik aplikacije.

Hello, [logic](#) ! [Odjava](#)
[\[English\]](#) [Slovenian](#)

Slika 5 - Izbira jezika

Nastopajoči (ang. Artists)

Glavna atrakcija za obiskovalce našega dogodka bodo nastopajoči. Aplikacija ponuja popoln pregled nad vsemi potencialnimi nastopajočimi, katerih informacije smo pridobili (slika 6). Ta pregled nastopajočih ne pomeni tudi dejanskih nastopajočih na konkretnem dogodku, ampak zbirko informacij, ki jih hranimo za uporabo pri trenutnih ali nadaljnjih organizacijah.

Nastopajoči				
Dodaj nastopajoče				
Ime	Tip	Kontakt	Cena	
Jay Lumen	Dj	jay@lumen.hu	1500	Uredi Podrobnosti Odstrani
NDKj	Dj	ndkj@heatflow.com - 003912345678	800	Uredi Podrobnosti Odstrani

Slika 6 - Pregled vseh možnih nastopajočih

Uporabnik lahko za vsakega izmed nastopajočih pregleda podrobnosti, ga uredi, odstrani ali pa doda novega. Uporabnik pri urejanju ali dodajanju vnese ime nastopajočega, njegov tip

(tukaj so mu na voljo izbire DJ, skupina, komik ipd.), ceno nastopa, potne stroške, stroške in zahteve nastavitve, možne dodatne stroške, državo, od koder nastopajoči prihaja, kontaktne informacije in opombe (slika 7).

Dodaj/Uredi

Ime: Jay Lumen

Tip: 0j

Cena: 1500

Potni stroški: 250

Stroški nastavitve: 80

Zahteve nastavitve: 4*

Drugi stroški:

Država: Madžarska

Email: jay@lumen.hu

Telefon:

Opombe: Edini let na voljo je do Zagreba, tako da ga bo potrebno pobrati na zagrebkem letališču v petek ob 13ih.

Shrani

[Nazaj na seznam](#)

Slika 7 - Obrazec za vnos novega nastopajočega

Prizorišča (ang. Venues)

Podobno kot pri nastopajočih, nam podstran Prizorišča ponuja pregled vseh možnih prizorišč za naše dogodke. Pri dodajanju ali urejanju vnesemo ime, tip (izbira med lokalom, klubom, na prostem...), ceno najema in ostale stroške, točen naslov, kar je še posebej pomembno pri opomnikih, kontaktne informacije ter opombe.

Podpora (ang. Support)

Med podporne elemente pri organizaciji dogodka spada veliko stvari. Potrebno je zagotoviti zadostno število sanitarij, primerno varnost, tako s strani varnostnikov, kot tudi fizično zaščito objektov in drugih dragocenosti, pijačo in po možnosti tudi hrano za obiskovalce, takšno in drugačno opremo ter še marsikaj. Tukaj nam je v pomoč pogled Podpora, kjer imamo vse potrebne kontakte zbrane na enem mestu. Uporabniki dodajajo posamezne podporne elemente na podoben način kot nastopajoče in prizorišča. Izbirajo lahko med različnimi tipi podpore, npr. glasbena oprema, osvetlitev, distribucija vstopnic... Število potrebnih elementov se drastično poveča, kadar imamo opravka s prireditvijo na prostem. Takrat se pojavijo potrebe še po dodatnih varnostnih ukrepih, varnostnih ograjah, postavitvijo odrov, kemijskih sanitarij itd.

Osebj (ang. Staff)

Za vsak dogodek potrebujemo določeno število ljudi, ki skrbijo za strežbo, varnost, zabavo in podobno. Uporabnik izbira med različnimi tipi osebja, od natakarejev, rediteljev, promotorjev, prodajalcev vstopnic, maskot do fizičnih delavcev. Za razliko od običajnega atributa cene, se pri osebju vnaša urna postavka ter pričakovano število delovnih ur.

Oglasi (ang. Advertisements)

Eden izmed pomembnejših faktorjev za organizacijo uspešnega dogodka je dobro oglaševanje le-tega. Uporabnik lahko doda oglaševalske možnosti za televizijske in radijske oglase, objave v revijah ali časopisih, spletne objave ter pripadajoče cene, kontaktne informacije in opombe.

Dogodki (ang. Events)

Glavni pogled v naši spletni aplikaciji in hkrati prva stran se imenuje Dogodki. To je izhodiščna točka za vse informacije, ki se tičejo organizacije našega dogodka. Pri dodajanju samega dogodka uporabnik vnese samo osnovne informacije o njem, ostalo se tekom organizacije na dogodek dodaja kasneje. Ob kreaciji novega dogodka uporabnik vnese ime dogodka, opis, elektronski naslov, povezavo na dogodek na spletu, če le-ta obstaja, ceno vstopnice, pričakovano število obiskovalcev ter seveda datum dogodka (slika 8).

Dodaj/Uredi	
Ime	Rock Fest
Opis	Dvodnevni festival rock glasbe v Radovljici. Že sedmo leto zapored vas bo dva dni na idilični lokaciji pod milim nebom zabavalo kar 20 rock skupin iz Slovenije in tujine. Obiskovalcem bo na voljo tudi brezplačno kampiranje na organiziranem prostoru, dogajanje pa bodo popestrili še plesalci in hostese.
Email	rockfest@radovljica.si
Url do dogodka na Facebook-u	http://www.facebook.com/RockFest7
Cena vstopnice	15
Ocena števila obiskovalcev	700
Datum od	2012-09-07 12:00
Datum do	2012-09-08 23:00
Trenutni proračun:	0
Shrani	
Nazaj na seznam	

Slika 8 - Dodajanje novega dogodka

Uporabnik potrdi kreacijo novega dogodka s klikom na akcijo Shrani. Nato se odpre nova povezava, kjer lahko dogodek urejamo bolj podrobno (slika 9).

Current balance: 7740

Add document					
Name	Url	Remove	Edit	Add reminder	
Dovoljenje župana občine Škofja Loka		Remove	Edit	Add reminder	
Razpis za sredstva		Remove	Edit	Add reminder	
Zapisnik razprave na policijski enoti Škofja Loka		Remove	Edit	Add reminder	
Pogodba z varnostno službo		Remove	Edit	Add reminder	

Add advertisement					
Name	Contact	Type	Cost	Remove	Add reminder
Uptech stran	uptech.info@gmail.com	Web	0	Remove	Add reminder
Pop TV	info@poptv.si - 013456789	Web	1700	Remove	Add reminder

Add income					
Name	Contact	Amount	Remove	Edit	Add reminder
Razpis	razpisi@slo.si	2000	Remove	Edit	Add reminder

Add support					
Name	Contact	Type	Cost	Remove	Add reminder
Frelih s.p.	frelih@oprema.si - 0411233456	MusicalEquipment	1500	Remove	Add reminder
Sanitarje Kranj	wo@kranj.si	Bathrooms	500	Remove	Add reminder

Add artist					
Name	Contact	Type	Cost	Remove	Add reminder
Jay Lumen	jay@lumen.hu	Dj	1500	Remove	Add reminder
NDKj	ndkj@heatflow.com - 003912345678	Dj	800	Remove	Add reminder

Add staff					
Name	Contact	Type	Cost	Remove	Add reminder
Kalan Miha	miha@kalan.com - 040345654	PhysicalWorker	60	Remove	Add reminder
Nina Gorjek	nina@gorjek.si	Bartender	40	Remove	Add reminder

Add venue					
Name	Contact	Type	Cost	Remove	Add reminder
Mestni trg		OpenAir	0	Remove	Add reminder
Vahtn'ca		Pub	0	Remove	Add reminder

Add reminder

No reminders have been added.

Save

Back to list

Slika 9 - Podrobnosti dogodka

Uporabnik lahko sedaj na dogodek dodaja posamezne entitete, ki bodo igrale vlogo pri organizaciji in izvedbi tega dogodka. Doda lahko nastopajoče, ki jih je prej že vnesel v sistem, ali pa jih vnese naknadno. Isto lahko stori z vsemi ostalimi entitetami, ki so bile bolj podrobno opisane v prejšnjih poglavjih. Ob kliku na Dodaj (ang. Add) za katerokoli entiteto se uporabniku prikaže seznam vseh entitet tistega tipa, ki so bile vnešene v aplikacijo (slika 10). Uporabnik nato željeno entiteto na dogodek doda s klikom na Dodaj k dogodku (ang. Add to event).

Advertisements					
Add advertisement					
Name	Type	Contact	Cost		
Uptech stran	Web	uptech.info@gmail.com	0	Edit	Details
Loški Utrip	Web	loski@utrip.si - 045123456	120	Edit	Details Add to event
Pop TV	Web	info@poptv.si - 013456789	1700	Edit	Details
Radio Sora	Web	radio@sora.si - 0451456789	200	Edit	Details Add to event
Back to event					

Slika 10 - Dodajanje entitete na dogodek

Novost so entitete z imenom Dokumenti. Dokumenti se ne dodajajo neodvisno, tako kot ostale entitete, saj so vezani na obravnavani dogodek. Na tem mestu lahko uporabnik doda povezave na pomembne dokumente, ki jih je prej nekje shranil (FTP, Dropbox...). S klikom na povezavo Url bo uporabnik preusmerjen na izbran dokument. Druga stvar, ki je vidna samo na tem mestu, so Opomniki (ang. Reminders). Omogočeno je dodajanje enega ali več opomnikov za vsako entiteto, vezano na naš dogodek. Če želimo dodati opomnik, ki je vezan samo na dogodek, in ne na točno določeno entiteto, pa to storimo s klikom na Dodaj

opomnik (ang. Add reminder), ki se nahaja nad seznamom opomnikov. V tem seznamu so opomniki, katerih nastavljen datum je prekoračen, obarvani z rdečo (slika 11).

Add reminder					
	Name	Reminder for	Time	Notify by email	
	Potrdi termine	Pop TV	29.8.2012 15:00:00	☒	Remove
	Vprašaj za popust	Glasbena oprema	31.8.2012 11:00:00	☒	Remove

Slika 11 - Seznam opomnikov

Pri dodajanju opomnikov (slika 12) uporabnik vnese ime ter vsebino opomnika, izbere datum in uro le-tega ter po želji izbere opcijo obveščanja po elektronski pošti. V primeru, da je ta izbrana, bo ob vnešenem datumu in času na elektronski naslov dogodka poslano obvestilo z vsebino opomnika in vsemi nujnimi informaciji o entiteti in dogodku, za katerega je bil ta opomnik dodan.

Add/Edit

Name

Time

Notes

Pokliči g. Janez Novaka in se dogovori ter potrdi termine za naš oglas.

Notify by email
☒

[Back](#)

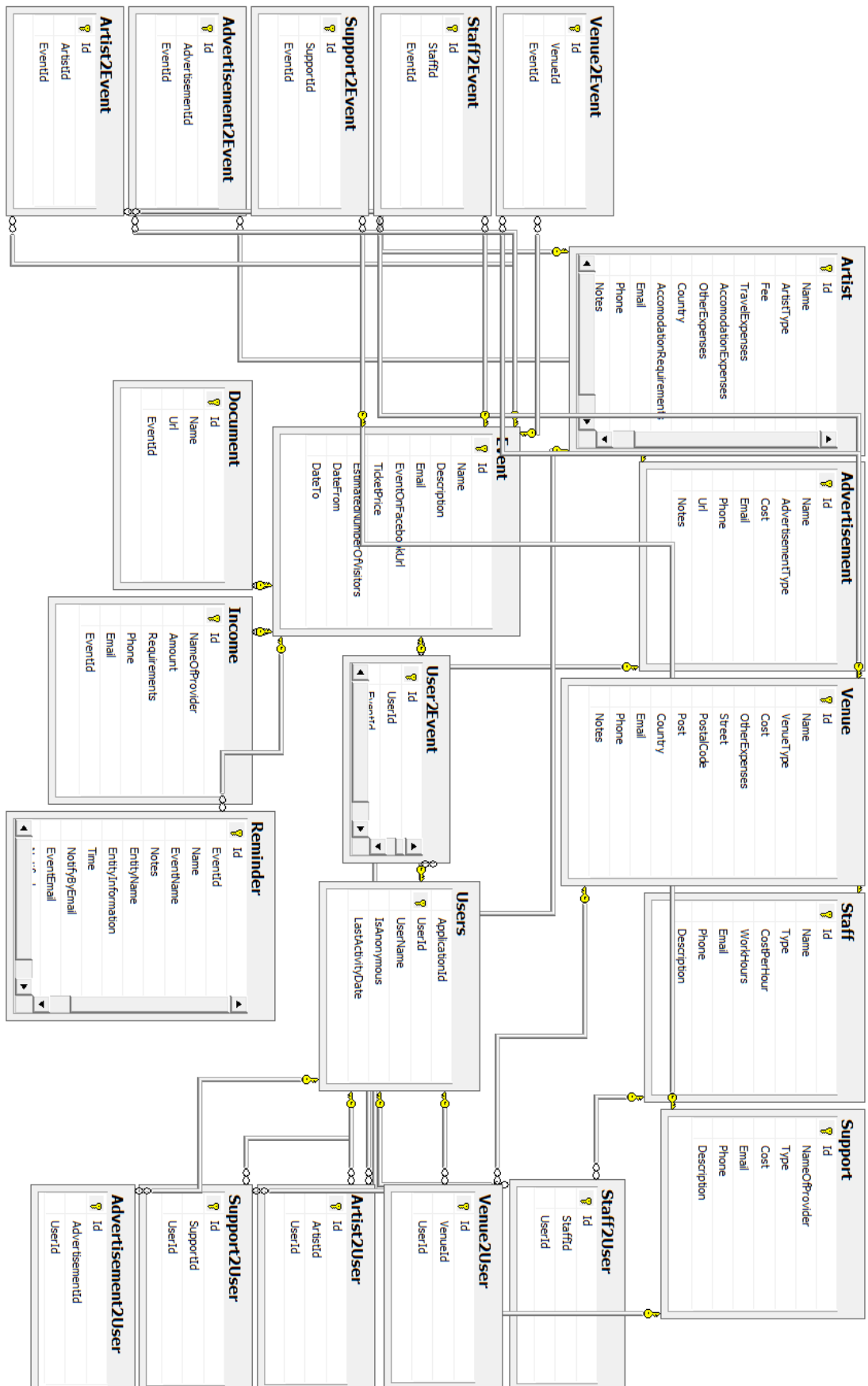
Slika 12 - Dodajanje opomnika za oglas

Zadnja entiteta, ki jo vidimo samo na pregledu dogodka pa je Dohodek (ang. Income). Tu vnesemo vsa sredstva, ki smo jih pridobili za organizacijo (sponzorji, razpisi, ipd.), in katera prispevajo h končnem proračunu dogodka.

3.3 Načrtovanje

Model podatkovne baze

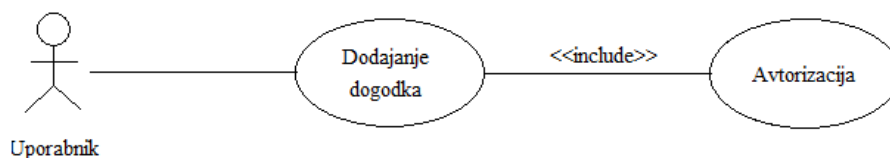
Naša glavna entiteta je Dogodek (ang. Event). Vse ostale entitete, od Nastopajočih do Podpore so z dogodkom povezane preko pripadajoče povezovalne tabele. Vsaka od navedenih entitet se lahko namreč uporabi na večih dogodkih. Podobno velja tudi za Uporabnike (ang. Users) v kombinaciji s prej omenjenimi entitetami. Izjema so entitete Dokument, Dohodek in Opomnik, ki so vezane direktno na dogodek. Model podatkovne baze je prikazan na sliki 13.



Slika 13 - Model podatkovne baze

Diagrami primerov uporabe

Diagram primere uporabe predstavlja povezavo med uporabniki sistema in procesi, ki v sistemu tečejo, hkrati pa predstavljajo prenos objektnega razmišljanja v poslovno modeliranje. Diagram primera uporabe služi kot osnova pri definiranju funkcionalnih zahtev, identificiranju in določanju lastnosti objektov, definiranju interakcije med objekti in vmesniki objektov, načrtovanju uporabniških vmesnikov, testiranju in komunikaciji z uporabnikom [6].



Slika 14 - Diagram primera uporabe dodajanja dogodka

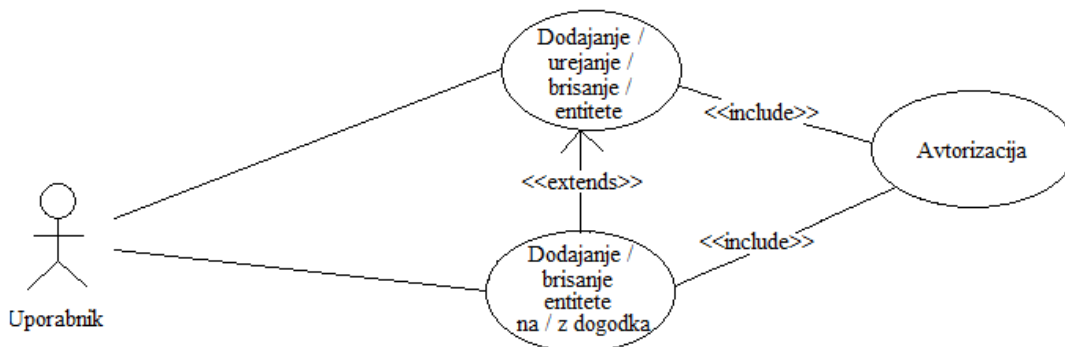
Diagram primera uporabe na sliki 14 prikazuje dodajanje novega dogodka v sistem. Uporabnik vnese informacije o dogodku in nato dogodek shrani ali pa akcijo prekliče.

Osnovni tok:

- A. Uporabnik izbere akcijo 'Dodaj dogodek' na seznamu dogodkov.
- B. Uporabnik vnese vse podatke o dogodku - ime, opis, elektronski naslov, povezavo do dogodka na spletu, ceno vstopnice, oceno števila obiskovalcev ter datum dogodka.
- C. Uporabnik vnose potrdi s pritiskom na 'Shrani'
- D. Dogodek se shrani v sistem
- E. Uporabniku se prikaže seznam njegovih dogodkov v sistemu

Alternative:

- Alt. 1 Uporabnik ni prijavljen v sistem. Sistem uporabnika preusmeri na prijavno stran, kjer se mora avtorizirati.
- Alt. 2 Uporabnik prekliče vnos dogodka s klikom na 'Nazaj na seznam'. Uporabniku se prikaže seznam njegovih dogodkov v sistemu. Dogodek se v sistem ne shrani.
- Alt. 3 Uporabnik ne vnese vseh obveznih podatkov. Pri poljih, ki so obvezna, a jih je uporabnik pustil prazna, se na desni izpiše obvestilo o manjkajoči vrednosti. Če te vrednosti uporabnik ne vnese, dogodka ne more shraniti v sistem.
- Alt. 4 Uporabnik vnese v določeno polje neveljavno vrednost. Na desni se ob tem polju izpiše obvestilo o nepravilnem vnosu. Dokler vnosa uporabnik ne popravi, dogodka ni možno shraniti v sistem.



Slika 15 - Diagram primera uporabe dodajanja, brisanja in urejanja entitete

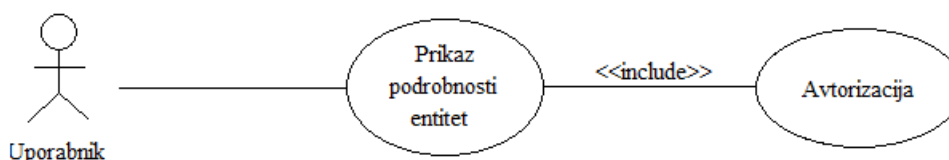
Diagram primera uporabe na sliki 15 prikazuje dodajanje, urejanje, ali brisanje entitete v sistem ter opsijsko dodajanje, ali brisanje entitete tudi na dogodku. Slednja akcija je razširitev prve.

Osnovni tok:

- A. Uporabnik izbere akcijo 'Dodaj' na seznamu izbrane entitete (Nastopajoči, Oglasi, Podpora, Lokacija, Osebj).
- B. Uporabnik vnese podatke o izbrani entiteti.
- C. Vnose potrdi s klikom na 'Shrani'.
- D. Entiteta se shrani v sistem.
- E. Uporabnik je preusmerjen na seznam entitet.
- F. Uporabnik na pogledu urejanje dogodka ob izbranem tipu entitete izbere 'Dodaj'
- G. Prikaže se seznam entitet izbranega tipa, ki pripadajo trenutno prijavljenemu uporabniku, in ki še niso bile dodane na dogodek.
- H. Uporabnik izbrane entitete doda na dogodek s klikom na 'Dodaj k dogodku'.
- I. Entiteta se doda k dogodku.

Alternative:

- Alt. 1 Uporabnik ni prijavljen v sistem. Sistem uporabnika preusmeri na prijavnno stran, kjer se mora avtorizirati.
- Alt. 2 Uporabnik prekliče vnos dogodka s klikom na 'Nazaj na seznam'. Uporabniku se prikaže seznam njegovih entitet določenega tipa v sistemu. Entiteta se v sistem ne shrani.
- Alt. 3 Uporabnik ne vnese vseh obveznih podatkov. Pri poljih, ki so obvezna, a jih je uporabnik pustil prazna, se na desni izpiše obvestilo o manjkajoči vrednosti. Če te vrednosti uporabnik ne vnese, entitete ne more shraniti v sistem.
- Alt. 4 Uporabnik vnese v določeno polje neveljavno vrednost. Na desni se ob tem polju izpiše obvestilo o nepravilnem vnosu. Dokler vnosa uporabnik ne popravi, entitete ni možno shraniti v sistem.
- Alt. 5 Uporabnik prekliče dodajanje entitete k dogodku s klikom na 'Nazaj na dogodek'. Uporabnik je preusmerjen nazaj na urejanje dogodka.
- Alt. 6 Entiteta, ki jo želi uporabnik dodati k dogodku, je bila k dogodku že dodana. Entiteta bo na seznamu možnih entitet za dodajanje k dogodku prikazana, a akcija 'Dodaj k dogodku' ne bo na voljo.
- Alt. 7 Uporabnik želi k dogodku dodati entiteto, ki pa v sistemu še ne obstaja. Na pregledu entitet, ki jih je možno dodati k dogodku, lahko izbere akcijo 'Dodaj'. Ponovijo se koraki primera uporabe od A. do D., v koraku E. pa je uporabnik preusmerjen nazaj na seznam entitet, ki jih je možno dodati k dogodku.
- Alt. 8 Uporabnik želi izbrisati entiteto. Izbere akcijo 'Odstrani' na seznamu entitet določenega tipa. Akcija je enaka za brisanje direktno na seznamu entitet določenega tipa, kot tudi na seznamu entitet, ki so bile dodane na dogodek. V prvem primeru se entiteta izbriše iz sistem, v drugem pa odstrani z dogodka, a v sistemu ostane. Uporabnik ostane na pogledu, na katerem je akcijo izbral.
- Alt. 9 Uporabnik želi urediti entiteto. Akcijo 'Uredi' lahko izbere direktno na seznamu entitet določenega tipa ali pa na seznamu entitet, ki so na voljo za dodajanje k dogodku. V obeh primerih se ponovijo koraki primera uporabe od B. do D., uporabnik pa je preusmerjen na seznam, na katerem je pred tem izbral akcijo.



Slika 16 - Diagram primera uporabe prikaza podrobnosti entitet

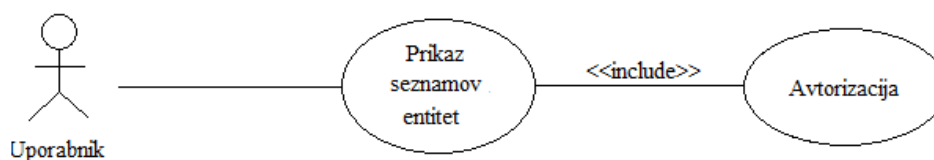
Diagram primera uporabe na sliki 16 prikazuje prikaz podrobnosti izbrane entitete.

Osnovni tok:

- A. Uporabnik na seznamu entitet izbere akcijo 'Podrobnosti' desno od imena želene entitete.
- B. Prikažejo se podrobnosti izbrane entitete.
- C. Uporabnik se vrne na prejšnji pogled s klikom na 'Nazaj' ali pa uredi trenutno ogledovano entiteto s klikom na 'Uredi'.

Alternative:

- Alt. 1 Uporabnik ni prijavljen v sistem. Sistem uporabnika preusmeri na prijavno stran, kjer se mora avtorizirati.



Slika 17 - Diagram primera uporabe prikaza seznamov entitet

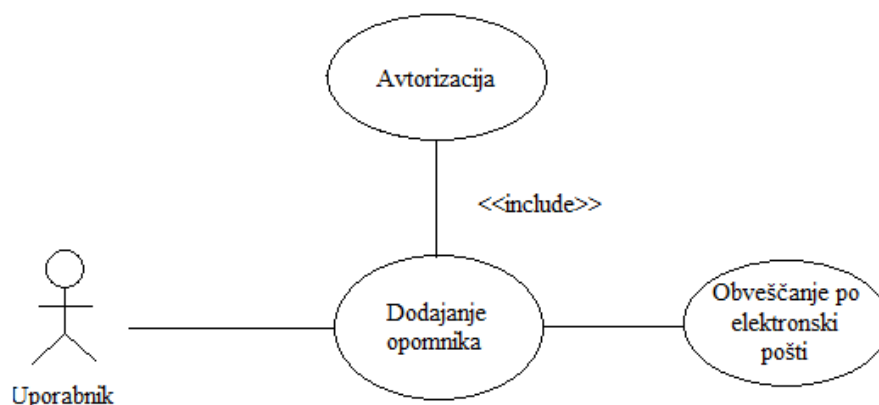
Diagram primera uporabe na sliki 17 prikazuje prikaz seznama izbranega tipa entitet.

Osnovni tok:

- A. Uporabnik v glavnem meniju izbere želen tip entitet.
- B. Prikažejo se seznam entitet izbranega tipa.
- C. Uporabnik lahko na seznam doda nove entiete ter ureja, briše in pregleduje podrobnosti obstoječih.

Alternative:

- Alt. 1 Uporabnik ni prijavljen v sistem. Sistem uporabnika preusmeri na prijavno stran, kjer se mora avtorizirati.



Slika 18 - - Diagram primera uporabe dodajanja opomnikov

Diagram primera uporabe na sliki 18 prikazuje dodajanje opomnikov in obveščanje o le-teh po elektronski pošti.

Osnovni tok:

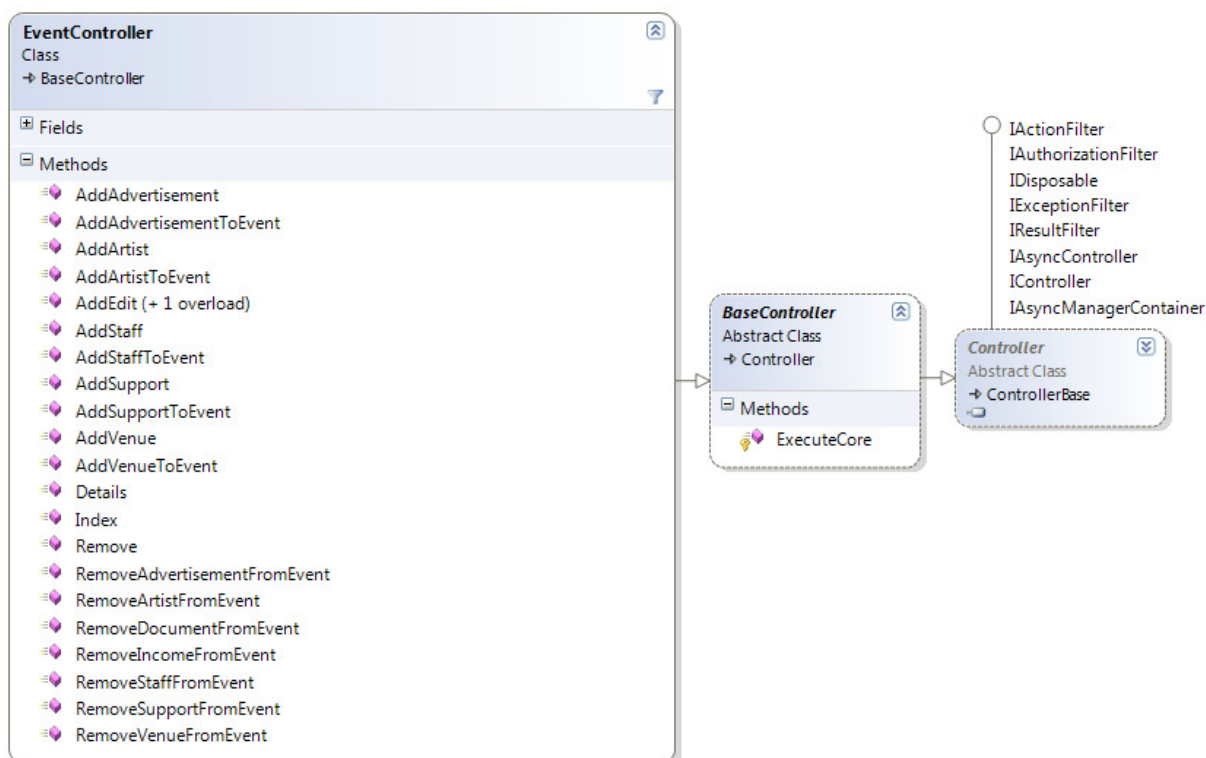
- A. Uporabnik na pogledu urejanje dogodka desno od izbrane entitete izbere akcijo 'Dodaj opomnik'.
- B. Prikaže se pogled za dodajanje opomnika.
- C. Uporabnik vnese Ime, Čas, Opombe in izbere opcijo Obvesti po elektronski pošti, če želi, da se obvestilo o opomniku pošlje na elektronski naslov dogodka.
- D. Vnos potrdi s klikom na 'Shrani'.
- E. Sistem uporabnika preusmeri nazaj na pogled urejanja dogodka, kjer v seznamu opomnikov sedaj vidi nazadnje dodan opomnik.
- F. Strežnik v časovnih intervalih preverja opomnike, ki so bili dodani v sistem. V primeru, da je bila izbrana opcija obveščanja po elektronski pošti, in da je čas opomnika manjši od trenutnega ob času preverjanja, strežnik preko ponudnika storitve elektronske pošte na elektronski naslov dogodka pošlje obvestilo o opomniku. Obvestilo vsebuje vse podatke, ki jih je uporabnik pri dodajanju opomnika vnesel, kot tudi vse informacije o entiteti, na katero se opomnik nanaša.
- G. V primeru, da je čas opomnika manjši od trenutnega časa, se na pregledu dogodka ta opomnik v seznamu opomnikov obarva rdeče.

Alternative:

- Alt. 1 Uporabnik ni prijavljen v sistem. Sistem uporabnika preusmeri na prijavnno stran, kjer se mora avtorizirati.
- Alt. 2 Uporabnik prekliče vnos s klikom na 'Nazaj'. Sistem ga preusmeri na pogled urejanja dogodka.
- Alt. 3 Uporabnik ne vnese vseh obveznih podatkov. Pri poljih, ki so obvezna, a jih je uporabnik pustil prazna, se na desni izpiše obvestilo o manjkajoči vrednosti. Če te vrednosti uporabnik ne vnese, entitete ne more shraniti v sistem.
- Alt. 4 Uporabnik vnese v določeno polje neveljavno vrednost. Na desni se ob tem polju izpiše obvestilo o nepravilnem vnosu. Dokler vnosa uporabnik ne popravi, entitete ni možno shraniti v sistem.
- Alt. 5 Elektronske pošte ni mogoče poslati. Strežnik bo neuspelo pošiljanje in razlog zanj zapisal v tekstovno datoteko, da bo lahko upravljalca ugotovil in popravil napako.

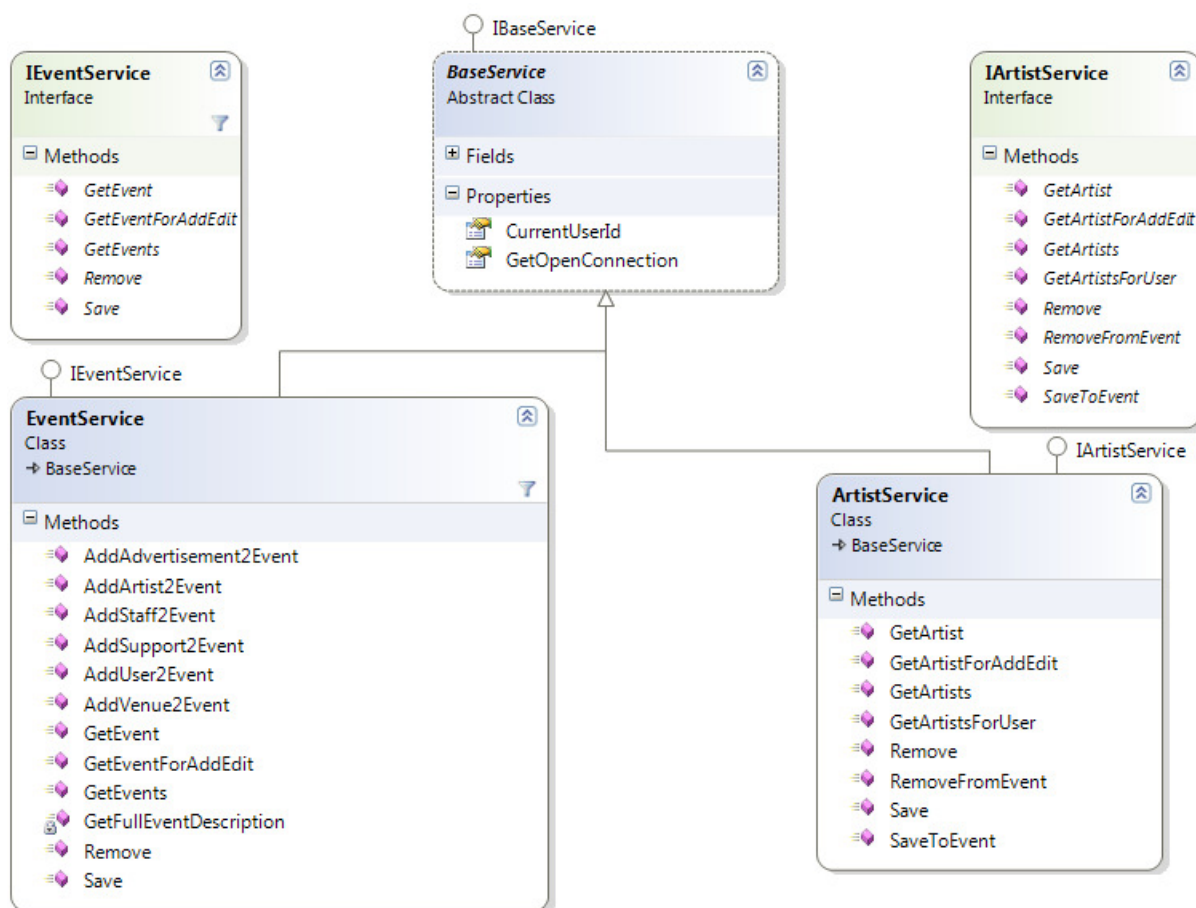
Razredni diagrami

Aplikacija je napisana v dveh nivojih. Najvišji je uporabniški vmesnik skupaj s kontrolerji, ki pogledom zagotavljajo potrebne podatke, drugi nivo pa je servisni, kjer se podatki pridobivajo iz podatkovne baze. Na sliki 19 je prikazan razredni diagram glavnega kontrolerja v aplikaciji, to je EventController. Vsebuje metode za dodajanje entitet v sistem, dodajanje entitet na dogodek ter brisanje le-teh z dogodka, dodajanje, urejanje in brisanje dogodka ter prikaz seznama dogodkov in podrobnosti. Razred deduje iz razreda BaseController, v katerem imamo implementirano logiko za prikaz lokaliziranih vsebin, le-ta pa deduje iz osnovnega MVC razreda Controller.



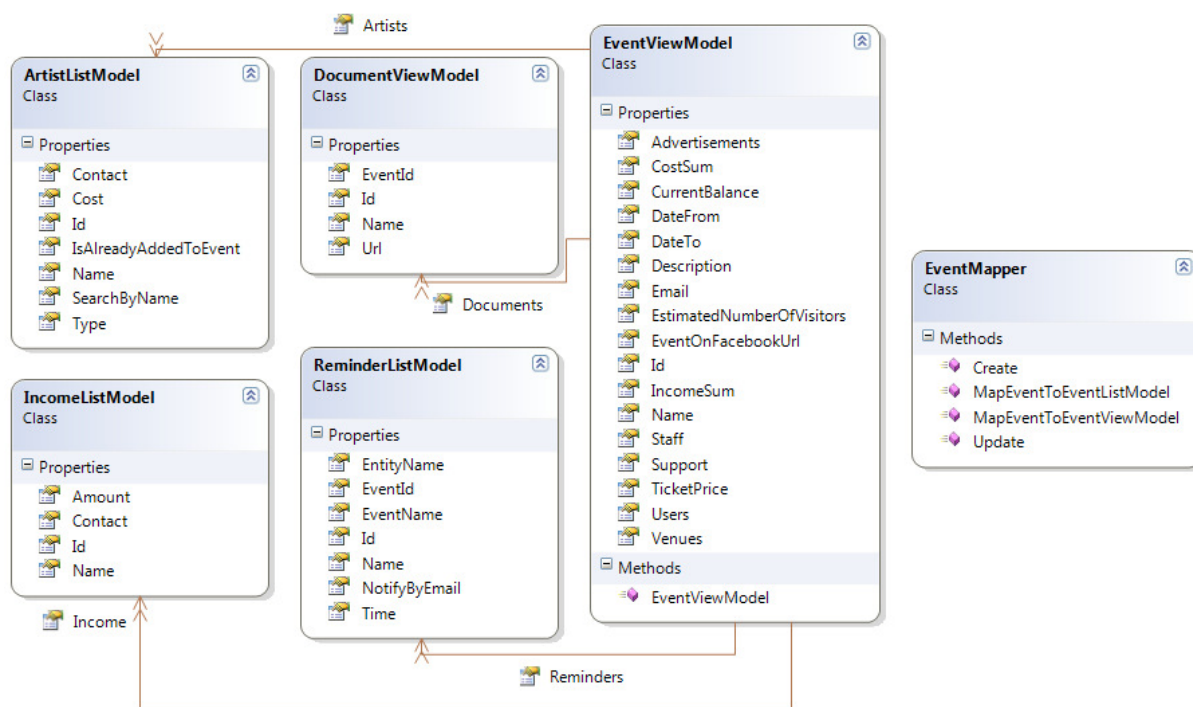
Slika 19 - Diagram razreda EventController

Na sliki 20 je prikazan diagram servisev ArtistService in EventService, prvi predstavlja servis za posamezne entitete v aplikaciji, drugi pa servis za najpomembnejšo izmed njih. Poleg drugih, vsebujeta tudi metode za pridobivanje, shranjevanje in brisanje posamezne entitete iz, ali v podatkovno bazo. Metode obeh so za druge razrede izpostavljene preko vmesnika (ang. interface). Servisa dedujeta iz razreda BaseService, ki nam ponuja trenutno odprto povezavo do podatkovne baze ter identifikator trenutno prijavljenega uporabnika v sistemu.



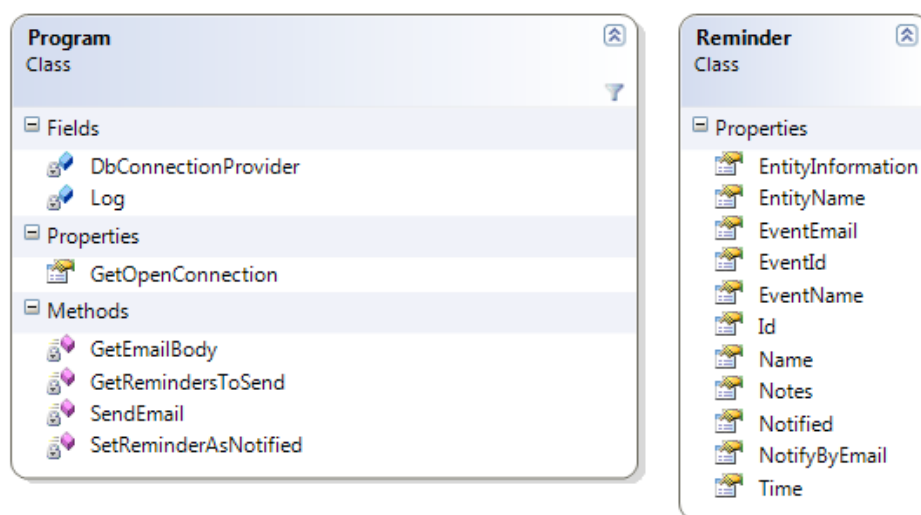
Slika 20 - Razredni diagram EventService in ArtistService

Na sliki 21 je orisan glavni razred za prikaz entitete Dogodek na uporabniškem vmesniku – EventViewModel. Poleg njega je prikazan tudi razred EventMapper, ki služi preslikovanju lastnosti med razredom Event in EventViewModel ter obratno. Naš obravnavani razred, poleg lastnosti dogodka, vsebuje sezname entitet, ki so bile dodane k dogodku (eno izmed njih na diagramu predstavlja ArtistListModel). Vsebuje pa tudi sezname entitet, ki so v naši aplikaciji vezane direktno na dogodek, na diagramu so vidne kot Dokumenti (ang. Documents), Opomniki (ang. Reminders) in Prihodek (ang. Income).



Slika 21 - Diagram razreda EventViewModel ter njegovih pomembnejših povezanih entiter ter preslikovalnika

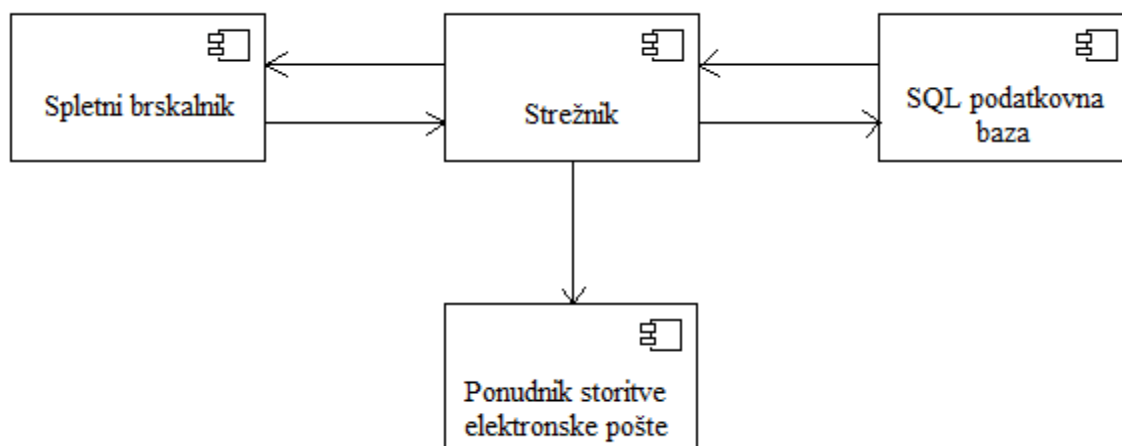
Zadnji razredni diagram na sliki 22 pa prikazuje dva glavna razreda v pomožni aplikaciji, ki se na strežniku zažene v določenih časovnih intervalih in poskrbi za obveščanje o opomnikih po elektronski pošti. Razred Opomnik (ang. Reminder) predstavlja opomnik, ki ga aplikacija pridobi iz podatkovne baze in nam zagotavlja vse potrebne podatke za vsebino elektronske pošte. Program pa je glavni razred aplikacije, ki poskrbi za pridobivanje opomnikov iz podatkovne baze, logiranje napak, sestavo vsebine elektronske pošte, pošiljanje le-te ter označevanje opomnikov kot poslane.



Slika 22 - Diagram glavnih dveh razredov v namizni aplikaciji za pošiljanje opomnikov preko elektronske pošte

Komponentni diagram

Komponentni diagram prikazuje odvisnost med programskimi komponentami (slika 23). Je graf, kjer so vozlišča komponente, povezave pa relacije med njimi. V našem primeru imamo odjemalca (spletni brskalnik), strežnik, na katerem gostujemo našo aplikacijo in pa podatkovno bazo, kjer se hranijo naši podatki. Strežnik v rednih časovnih intervalih preverja opomnike, ki so bili vneseni v sistem ter preko ponudnika storitve elektronske pošte pošilja obvestila o le-teh.



Slika 23 - Komponentni diagram

3.4 Možne nadgradnje

Poleg nadgradenj, ki jih bodo od nas zahtevale spremembe na področju organizacije dogodkov, imamo že v tem trenutku v mislih nekaj možnih izboljšav, ki bi nam še olajšala celoten postopek organizacije. Poleg obveščanja po elektronski pošti bi lahko dodali tudi obveščanje preko kratkih sporočil, oziroma SMS-ov. Problem, ki se pojavi pri tej rešitvi je ta, da bi bilo potrebno pridobiti dovoljenje izbranega ponudnika mobilnih storitev. Poleg tega bi ta ponudnik v zameno za svoje usluge zahteval plačilo na vsako poslano sporočilo. Glede na to, da smo aplikacijo naredili brezplačno, tak strošek ne pride v poštev. Na srečo smo v dobi, kjer prevladujejo t.i. pametni telefoni in nam zato obveščanje preko elektronske pošte ne otežuje doseganja naših ciljev. Večina omenjenih telefonov že namreč podpira elektronsko pošto in nas obvesti o prejetju le-te, kar je za našo želeno funkcionalnost dovolj.

Ker pa so hkrati taki telefoni postali del našega vsakdanjega, bi bilo vredno razmisliti v smeri prilagoditve naše spletne aplikacije za prikaz na mobilnih brskalnikih. Nove funkcionalnosti v ogrodju ASP.NET MVC 4 nam omogočajo prav to, zato je bila izbira tega orodja prava odločitev. Še bolj pa so zadnje čase v ospredju mobilne aplikacije. Možna nadgradnja naše aplikacije je torej tudi razvoj mobilne verzije le-te. Glavna prednost bi se pokazala predvsem pri opomnikih. V primeru, da uporabnik pride na območje, kjer mobilni internet ne deluje, namreč ne bo mogel prejeti opomnikov preko elektronske pošte. Če bi našo aplikacijo razvili kot mobilno, bi lahko v rednih intervalih preverjali za nove opomnike in jih shranili kot dejanske opomnike na samem telefonu. V tem primeru bi uporabnik prejeli obvestilo, oziroma opomnik, tudi če ne bi imel dostopa do interneta ali mobilnega omrežja. Uporabnikom bi mogoče lahko tudi omogočili direktno upravljanje z dokumenti iz same aplikacije. Trenutno se v sistem vnese samo povezava na dokument, možna nadgradnja pa bi bil prenos dokumentov v samo aplikacijo, ki bi jih nato shranila na želeno mesto, npr. Dropbox.

4 Implementacija spletne aplikacije

Pri razvoju naše spletne aplikacije smo uporabili še neizdano verzijo ogrodja MVC, in sicer verzijo 4, ki je ob času pisanja te diplomske naloge še v fazi kandidat za izdajo. Omenjeno ogrodje nam namreč ponuja večji nabor možnosti kot prejšnja verzija tega istega ogrodja, kar nam zagotavlja uporabnost aplikacije tudi na daljši rok. Ne glede na to smo aplikacijo zastavili na način, ki nam bom v prihodnosti omogočil lahko nadgrajevanje z malo možnosti napak. Pri tem nam je bila v veliko pomoč razvijalska tehnika testno vodenega razvoja (TDD), ki nam ob sprotnem pisanju testov zagotavlja čim večjo pokritost kode s strani le-teh in posledično večje zaupanja v delovanje same aplikacije ob spremembah, če so seveda testi po tej spremembi še vedno funkcionalni. V nasprotnem primeru pa je padel test znak, da nekaj v kodi ne deluje, kar nam v zgodnji fazi razvoja prihrani naporno iskanje napak kasneje. Uporabniki aplikacije bodo iz različnih področij, zato smo poskusili uporabniški vmesnik in logiko za njim narediti čimbolj preprosto in intuitivno.

4.1 SOLID (Objektno orientirano načrtovanje)

SOLID (Single responsibility, Open-closed, Liskov substitution, Interface segregation in Dependency inversion) je mnemonični akronim, ki ga je uvedel Robert C. Martin v začetku tega tisočletja in ponazarja pet osnovnih principov objektno orientiranega programiranja in načrtovanja. Ko so ti principi uporabljeni skupaj, je njihov namen omogočiti programerju, da razvije sistem, ki bo lahek za vzdrževanje in nadgradnjo tekom časa. Principi SOLID-a so vodila, ki jih lahko uporabimo tudi za izboljšanje slabe kode s pomočjo refaktoriranja, s čimer lahko kodi izboljšamo berljivost in možnosti nadgradnje. Ponavadi se uporablja skupaj s testno vodenim programiranjem (TDD) in je del splošne strategije agilnega in adaptivnega programiranja [7].

SRP (Single responsibility principle)

Vsak razred mora imeti samo eno odgovornost in ta odgovornost mora biti popolnoma enkapsulirana s strani razreda. [8]

Odgovornost je definirana kot razlog za spremembo, kar pomeni, da mora imeti razred ali modul samo en razlog za spremembo. Ta razlog je pomemben, saj je tako razred osredotočen na eno samo namembnost, kar ga naredi bolj robustnega.

OCP (Open/closed principle)

Entitete naj bodo odprte za razširitve, a zaprte za modifikacije. [9]

To pomeni, da lahko taka entiteta dovoli, da se spremeni njeno obnašanje, ne da bi se spremenila njena izvorna koda. To je še posebej uporabno v produkcijskem okolju, kjer spremembe kode ponavadi vključujejo tudi pregled kode, ponovno testiranje in druge procedure, ki zagotovijo, da je koda še vedno primerna za uporabo v produktu. Koda, ki upošteva to načelo, se ob razširitvi ne spremeni in posledično ne potrebuje vsega tega dodatnega napora.

LSP (Liskov substitution principle)

Objekti v programu se lahko zamenjajo s primerki njihovih podtipov, ne da bi s tem vplivali na pravilnost programa. [10]

Princip je razširitev OCP principa.

ISP (Interface segregation principle)

Več specifičnih vmesnikov je boljših kot en sam s splošnim namenom. [11]

Princip se uporablja za čist razvoj, njegov glavni cilj pa je programska oprema, ki jo je lahko spreminjati. ISP razdeli velike vmesnike na manjše, bolj specifične, tako da klienti vedo samo za metode, ki jih zanimajo.

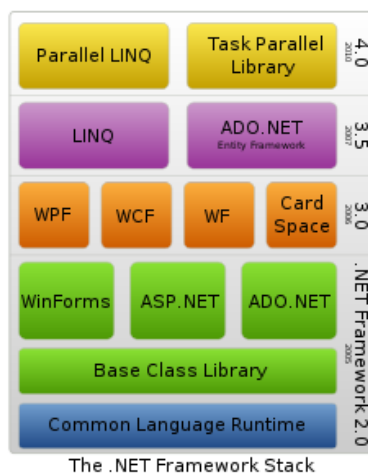
DIP (Dependency inversion principle)

Visokonivjski moduli ne smejo biti odvisni od nizkonivjskih. Oboji naj bodo odvisni od abstrakcij. Abstrakcije ne smejo biti odvisne od podrobnosti. Podrobnosti naj bodo odvisne od abstrakcij. [12]

Vstavljanje odvisnosti (Dependency injection) je ena od metod, ki sledijo temu principu.

4.2 Microsoft .NET

.NET ogrodje je programsko ogrodje, razvito s strani podjetja Microsoft, ki primarno deluje na sistemih z Microsoft Windows operacijskim sistemom. Vsebuje veliko knjižnico in podpira delo v več računalniških jezikih. Diagram orodja je prikazan na sliki 24. Programi, napisani v .NET ogrodju se izvršujejo v programskem okolju, znanim pod imenom Common Language Runtime, krajše CLR, ki je virtualna naprava s servisi, kot so varnost, upravljanje s spominom in upravljanje z napakami. Osnovna knjižnica .NET ogrodja zagotavlja uporabniške vmesnike, dostop do podatkov, povezavo s podatkovno bazo, kriptografijo, razvoj spletnih aplikacij, numerične algoritme in komunikacijo prek omrežij. [18]



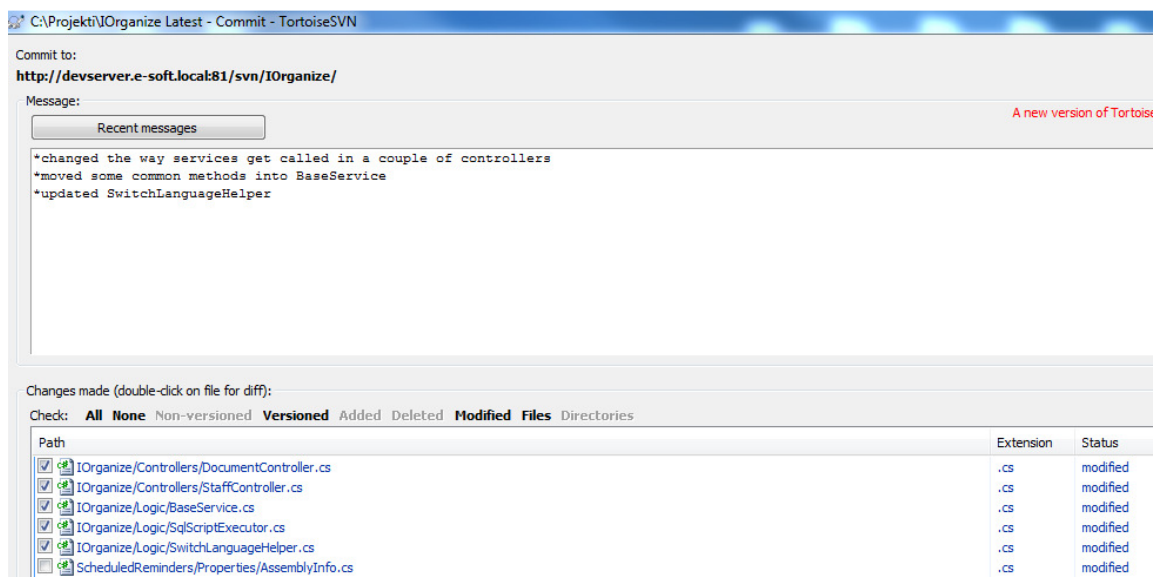
Slika 24 - Diagram ogrodja .NET [13]

4.3 Microsoft Visual Studio 2010

Visual Studio je najbolj razširjeno integrirano razvojno okolje za .NET ogrodje. V našem primeru smo uporabili verzijo 2010. Zagotavlja nam razvoj celotnega projekta, od začetnega načrtovanja do produkcije, od podatkovnega modela pa vse do uporabniškega vmesnika. Z njim lahko zgradimo spletne aplikacije, namizne aplikacije, XML spletne servise, pa tudi mobilne aplikacije.

4.4 Shramba za nadzor verzij

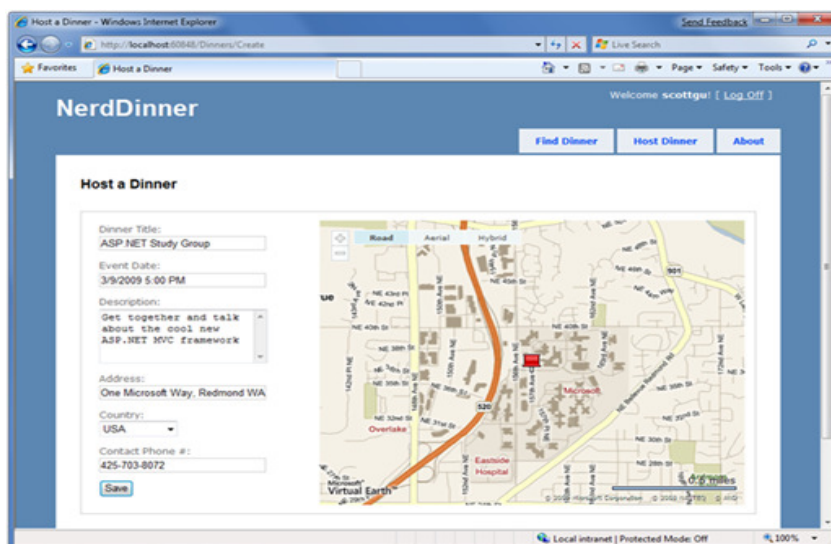
Za nadzor nad verzijami kode naše aplikacije smo uporabili shrambo za nadzor verzij (ang. version control repository). [26] Shramba nam omogoča takojšnjo postavitve delovnega okolja, ne da bi morali na novo iskati vse potrebne knjižnice ali verzij orodij. Prav tako nam omogoča, da se lahko v kateremkoli danem trenutku vrnemo na prejšnjo ali točno določeno verzijo naše kode, saj nam shramba omogoča sledljivost vseh sprememb od verzije 1 naprej. Priporočljivo je, da ob vsaki potrditvi našega dela v shrambo (ang. commit) dopišemo kratek opis sprememb (slika 25), kar nam na dolgi rok zagotovi popoln pregled celotne zgodovine projekta. Lažje je tudi iskanje vzrokov za nedelovanje aplikacije, saj za vsako potrditev vidimo tudi vse spremenjene datoteke.



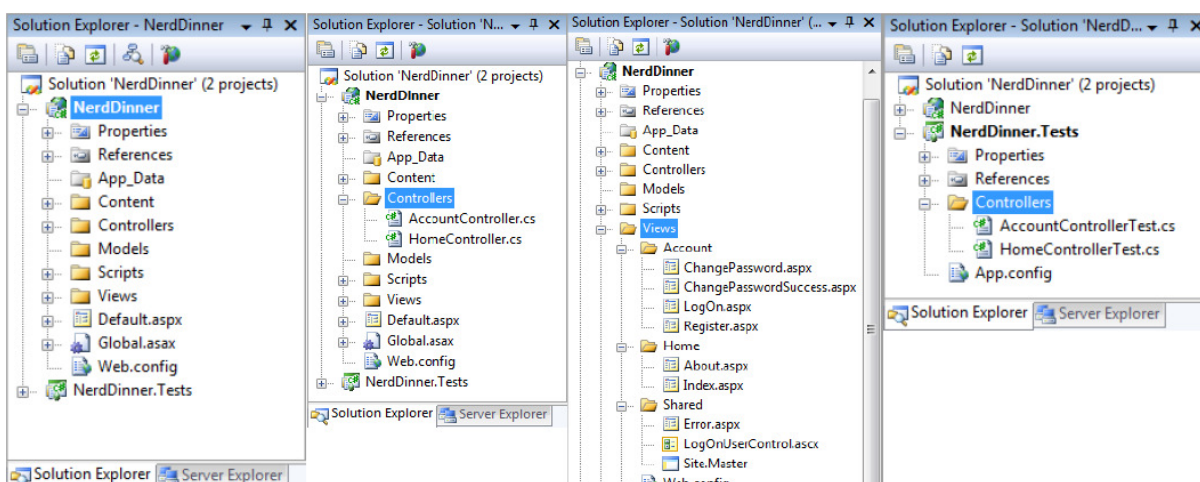
Slika 25 - Potrjevanje sprememb v shrambo

4.5 Osnovna arhitektura aplikacije (ASP.NET MVC 4)

MVC (ang. Model–View–Controller) je arhitekturni vzorec pri razvoju programske opreme. Glavni namen je ločiti logiko aplikacije od uporabniškega vmesnika in s tem omogočiti ločen razvoj, testiranje in implementacijo obojega. Model je predstavitev podatkov, na katerih aplikacija izvaja operacije. Ob spremembi stanja model obvesti uporabniški vmesnik ali pogled (ang. View), da le-ta osveži predstavljene podatke. Pogled prilagodi podatke na način, primeren za predstavitev uporabniku. Kontroler (ang. Controller) pa sprejema zahteve (glede na to, da se MVC pogosto uporablja v spletnih aplikacijah, so to ponavadi GET ali POST vnosi) in generira odgovor s pomočjo logike modela, ki zahtevo izvrši. Primer MVC spletne aplikacije je prikazan na sliki 26, zgradba takih aplikacij pa na sliki 27.



Slika 26 - Primer MVC spletne aplikacije [14]



Slika 27 - Primer zgradbe MVC projekta [14]

Izboljšave v ASP.NET MVC 4 kandidatu za izdajo

Tu so opisane spremembe in predvsem izboljšave, ki so bile vpeljane v zadnji verziji ASP.NET orodja. MVC 4 je v fazi kandidat za izdajo, kar pomeni, da se bodo spodnje funkcionalnosti verjetno še razširile in izboljšale do dejanskega izida MVC 4 [15].

Izboljšave v privzetih predlogah za projekte

Predloga, ki se uporabi za kreiranje novega ASP.NET MVC 4 projekta je bila nadgrajena na bolj moderen izgled spletne strani (slika 28).

IOrganize Zavrnj, Iga, I. [Oglej](#)

Dogodki Nastopajoči Prizorišča Podpora Osebe Oglasi

Dodaj/Uredi

Ime:

Opis:

Email:

Url do dogodka na Facebook-u:

Cena vstopnice:

Ocena števila obiskovalcev:

Datum od:

Datum do:

Številni proračun:

Dodaj dokument

Ime	Tip	Cena	Ostalo	Dodaj dokument
Primo 20115	Web	0	Ostalo	Dodaj dokument

Dodaj oglas

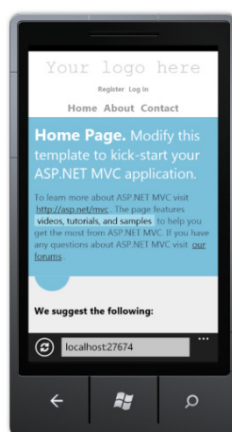
Ime	Kontakt	Tip	Cena	Ostalo	Dodaj oglas
Upnech	upnech.info@gmail.com	Web	0	Ostalo	Dodaj oglas

Dodaj nastopajoč

Ime	Kontakt	Tip	Cena	Ostalo	Dodaj nastopajoč
Iry	Lumen	0	3500	Ostalo	Dodaj nastopajoč

Slika 28 - Nova predloga za spletne strani v ASP.NET MVC 4 v naši aplikaciji

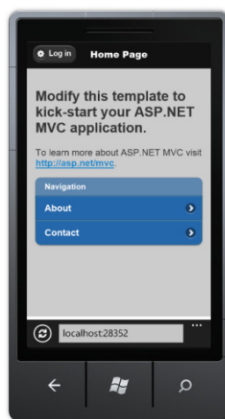
Poleg estetskih popravkov pa nova predloga vsebuje tudi izboljšane funkcionalnosti. Predloga vsebuje tehniko, imenovano prilagodljiv prikaz (ang. adaptive rendering), ki omogoči dober izgled tako v namiznih, kot tudi mobilnih brskalniki (slika 29), brez kakršnekoli prilagoditve.



Slika 29 - Primer spletne strani v mobilnem brskalniku [15]

Predloga 'Mobilna aplikacija'

Če bi radi naredili spletno stran prav za mobilne brskalnike, lahko uporabimo novo predlogo za projekte 'Mobilna aplikacija'. Le-ta je osnovana na jQuery Mobile, odprtokodni knjižnici za gradnjo uporabniških vmesnikov, optimiziranih, da delujejo na dotik. Izgled predloge je viden na sliki 30.



Slika 30 - Predloga 'Mobilna aplikacija' [15]

Ta predloga vsebuje isto aplikacijsko strukturo kot predloga internetne aplikacije (prav tako je koda v kontrolerju praktično identična), le da je ta stilizirana s pomočjo jQuery Mobile, da izgleda in se obnaša dobro na mobilnih napravah na dotik. V kolikor pa že imamo stran, namenjeno za prikaz na namiznih brskalnikih, kateri bi radi dodali poglede, optimizirane za mobilne naprave, ali pa bi radi ustvarili stran, ki postreže z drugačnim prikazom za namizne in mobilne brskalnike, pa lahko uporabimo novo funkcionalnost 'Prikazni načini', ki je opisana v nadaljevanju.

Prikazni načini

Nova funkcionalnost 'Prikazni načini' dovoli aplikaciji izbiro prikaza glede na brskalnik, ki je poslal prošnjo. Če naprimer namizni brskalnik zahteva domačo stran, aplikacija lahko uporabi predlogo *Views\Home\Index.cshtml*. Če pa isto stran zahteva mobilni brskalnik, pa lahko aplikacija vrne predlogo *Views\Home\Index.mobile.cshtml*.

Prav tako lahko prepišemo postavitve in delne poglede za določene tipe brskalnikov. Primer: Če vaš direktorij *Views\Shared* vsebuje tako *_Layout.cshtml* in *_Layout.mobile.cshtml* predlogi, bo aplikacijo privzeto uporabila *_Layout.mobile.cshtml* pri prošnjah s strani mobilnih brskalnikov ter *_Layout.cshtml* pri drugih prošnjah. Če direktorij vsebuje tako *_MyPartial.cshtml*, kot tudi *_MyPartial.mobile.cshtml*, bo navodilo *@Html.Partial("_MyPartial")* prikazalo *_MyPartial.mobile.cshtml* pri prošnjah s strani mobilnih brskalnikov in *_MyPartial.cshtml* pri drugih prošnjah.

Če bi radi ustvarili še bolj specifične poglede, postavitve ali delne poglede za druge naprave, lahko registriramo novo *DefaultDisplayMode* instanco, da navedemo, katero ime naj aplikacija poišče, ko prošnja zadovolji točno določene pogoje. Lahko bi naprimer dodali sledečo kodo v metodo *Application_Start* v datoteki *Global.asax*, ki bi registrirala tekst "iPhone" kot prikazni način, ki velja, kadar Apple iPhone brskalnik pošlje prošnjo:

```
DisplayModeProvider.Instance.Modes.Insert(0, new
DefaultDisplayMode("iPhone")
{
    ContextCondition = (context => context.GetOverriddenUserAgent().IndexOf
        ("iPhone", StringComparison.OrdinalIgnoreCase) >= 0)
});
```

Ko se ta koda zažene in Apple iPhone brskalnik pošlje prošnjo, bo naša aplikacija uporabila *Views\Shared_Layout.iPhone.cshtml* predlogo (če le-ta obstaja) [15].

4.6 Testno voden razvoj (ang. Test Driven Development)

Ker se glavna tema te diplomske naloge ves čas razvija in nadgrajuje, se bo morala spremembam na tem področju seveda prilagajati tudi sama aplikacija. Zato je bilo v ospredju pri razvoju tudi vzdrževanje in nadgradnja. Ti dve stvari smo si olajšali s sprotnim pisanjem testov, ki lahko ob kateremkoli času preverijo vse vidike aplikacije. Testno voden razvoj (ang. Test Driven Development ali krajše TDD) je razvijalska tehnika, ki prisili programerja, da piše teste skupaj z razvojem same aplikacije ter skrbi, da so testi elementarni in preprosti za razumevanje. Prihrani čas, saj namesto zaganjanja aplikacije in dejanske izvedbe določene funkcije ali odziva na dogodek, samo poženemo test. Ker so le-ti elementarni, je koda, napisana na ta način, precej bolj pregledna ter lažja za vzdrževanje. Poleg prednosti pri vzdrževanju, pa raziskave tudi kažejo, da ta tehinka poveča produktivnost programerja, kakor tudi njegovo zaupanje v samo kodo. Ker so testi elementarni in ker so napisani pred dejansko implementacijo posameznega sklopa, se programer bolj posveti sami funkcionalnosti, preden jo sploh napiše. Napake v kodi se na ta način lahko ujamejo v zgodnji fazi razvoja, kar nam prihrani dolgotrajno iskanje in popravljanje kasneje v življenjskem ciklu projekta. [17] Osnovni koraki pri tej tehniki so prikazani na sliki 31 in so naslednji [19]:

Dodaj test

V TDD se vsaka nova funkcionalnost začne s pisanjem testa. Ta test mora seveda spodleteti, ker je napisan preden se funkcionalnost implementira. Da razvijalec tak test napiše, mora točno vedeti kaj in kako bo ta nova funkcionalnost počela. To prisili razvijalce k boljšemu razumevanju zahtev in posledično boljše napisani funkcionalnosti ali pa vsaj kodi.

Poženi vse teste in preveri da zadnje napisani spodleti

Tako se prepričamo, da naše testiranje deluje pravilno in da naš novi test ni slučajno uspel, ne da bi sploh vseboval kakršnokoli kodo. Ta korak hkrati testira sam test in izloči možnost, da bi test vedno uspel in s tem izničil svojo uporabnost.

Napiši kodo

Sedaj napišemo kodo, ki bo povzročila, da bo test uspel. Ta koda v tej fazi še ne bo popolna saj je njen namen le, da test uspe. To je spremenljivo, ker so naslednji koraki namenjeni njenemu izboljšanju.

Poženi teste in preveri, da uspejo

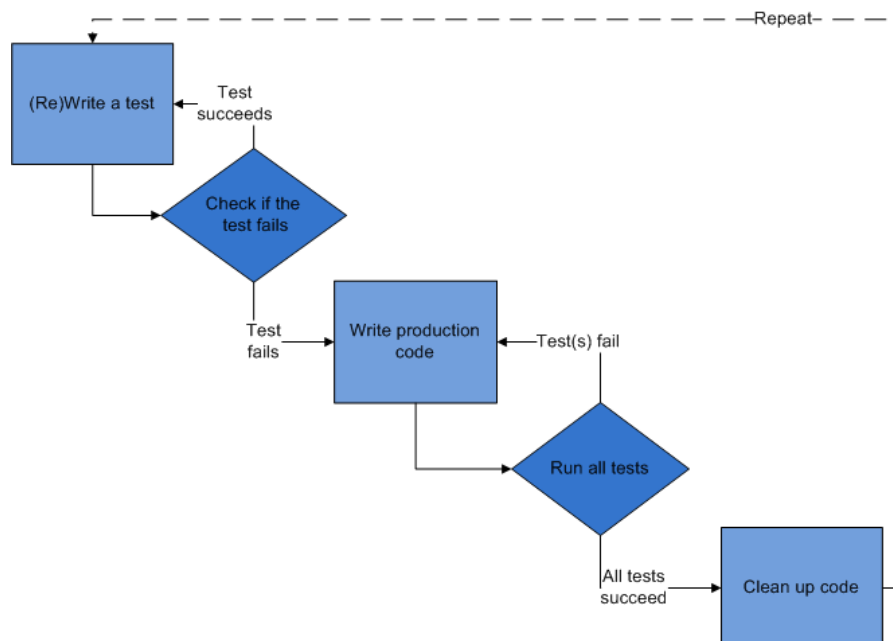
Če testi sedaj uspejo, je lahko programer prepričan, da koda dela, kar mora, to pa predstavlja podlago za naslednji korak.

Izboljšaj kodo

Sedaj lahko počistimo kodo. Med optimizacijo lahko ponovno poženemo test in se s tem prepričamo, da izboljšave niso vplivale na končno funkcionalnost.

Ponovi

Spet začnemo z novim testom in ponovimo cikel ter tako izboljšujemo funkcionalnost. Velikost korakov naj bi bila vedno majhna, z manj kot desetimi popravki med vsakim testom. S tem si zagotovimo dobro testirano kodo, kar nam neprimerno olajša nadaljne vzdrževanje le-te.



Slika 31 - Koraki v testno vodenem razvoju [20]

V času razvoja smo imeli za hitro kreiranje baze pripravljeno skripto, ki se je sprti dopolnjevala, kadar se je dodala, ali spremenila kakšna entiteta. To isto skripto smo uporabili tudi pri testih. Pred zagonom vsakega testa smo klicali funkcijo, ki je na novo ustvarila testno bazo (PersistenceTest), tako da je imel vsak test na voljo prazno bazo. Napisali smo si tudi pomožne generične funkcije, ki so nam preverjale shranjevanje ali vračanje podatkov, tako da smo lahko teste, ki so bili skupni vsem entitetam, izvedli z enim samim klicem ene od teh funkcij. Podobne funkcije smo si pripravili tudi za testiranje pogledov in kontrolerjev (slika 32).

```

protected void AssertPartialViewRendered(ActionResult result)
{
    Assert.IsNotNull(result, "Did not render a partial view");
    Assert.AreEqual(((PartialViewResult)result).ViewName, "");
}

protected static void AssertRedirectsToAction(ActionResult result, string action)
{
    Assert.AreEqual(action, ((RedirectToRouteResult)result).RouteValues["Action"]);
}

protected void AssertValueForField<TModel, TValue>(Expression<Func<TModel, TValue>> func, TValue value)
{
    using (var connection = GetOpenConnection())
    {
        var fieldName = GetMemberName(func);
        var dbResult = connection.Query(string.Format("SELECT {0} FROM {1}", fieldName, typeof(TModel).Name)).Single();
        var dictionary = ((IDictionary<string, object>)dbResult);
        Assert.IsTrue(dictionary.ContainsKey(fieldName), "The record in the database does not contain a field named \"{0}\"", fieldName);
        var fieldValue = ((IDictionary<string, object>)dbResult)[fieldName];
        Assert.AreEqual(value, fieldValue);
    }
}

```

```

protected void AssertSingleRecord<TModel>()
{
    using (var connection = GetOpenConnection())
    {
        var results = connection.Query(string.Format("SELECT * FROM {0}", typeof(TModel).Name));
        Assert.AreEqual(1, results.Count(), "Expected a single record in table {0}", typeof(TModel).Name);
    }
}

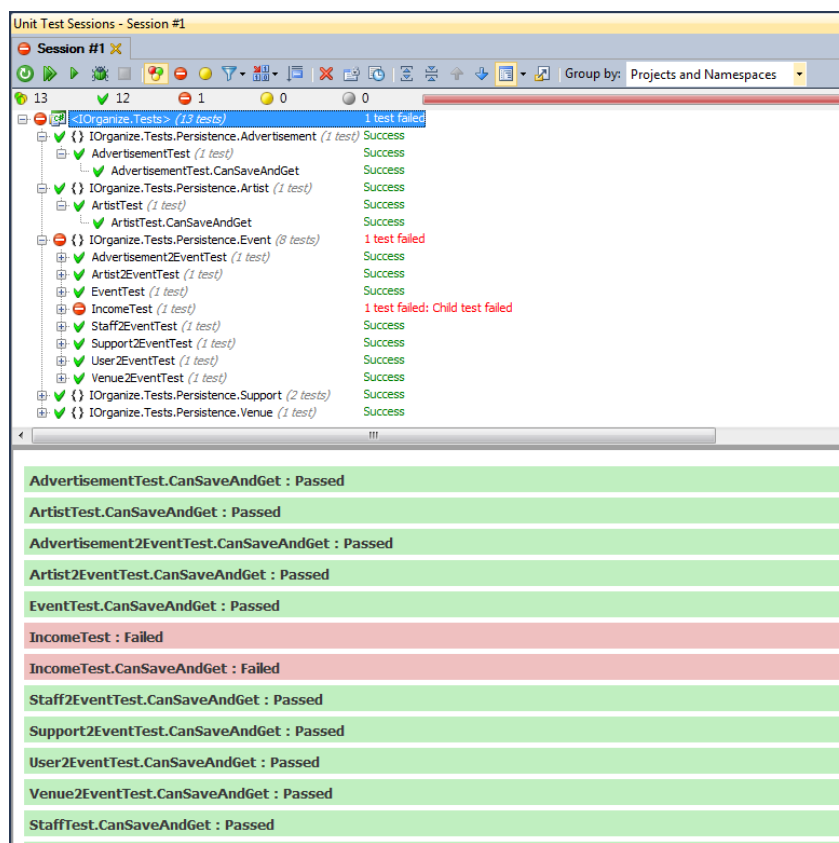
protected void AssertNoRecord<TModel>()
{
    using (var connection = GetOpenConnection())
    {
        var results = connection.Query(string.Format("SELECT * FROM {0}", typeof(TModel).Name));
        Assert.AreEqual(0, results.Count(), "Expected no records in table {0}", typeof(TModel).Name);
    }
}

```

Slika 32 - Nekatere generične funkcije za pomoč pri testiranju v aplikaciji

NUnit

Za hrbtnico testov smo uporabili NUnit. NUnit je osnova za enotsko testiranje za vse .NET programske jezike. Na začetku je bila narejena na podlagi JUnit, zadnjih nekaj verzij pa je narejenih izključno v jeziku C#. Od takrat je bila popolnoma spremenjena na način, ki lahko izkoristi številne prednosti .NET jezikov, med njimi tudi attribute po meri in druge zmožnosti, povezane z refleksijo (ang. reflection) [16]. Na sliki 33 je prikazan primer poganjanja in rezultati testov v orodju Visual Studio 2010.



Slika 33 - Poganjanje testov znotraj Visual Studia 2010

Rhino Mocks

Rhino Mocks je dinamično ogrodje mock objektov za .NET platformo. Njegov namen je poenostavitev testiranja tako, da dovoli razvijalcu kreiranje navideznih implementacij objektov po meri in preverjanje interakcij s pomočjo enotskega testiranja. Ponujajo možnost Uredi, Ukrepaj in Zagotovi (ang. Arrange, Act and Assert) ali Eksplicitna izvedba, ponovno zaženi in preglej (ang. Explicit record, replay and verify) modelov za pričakovanja, podporo generikom, C# Lambda izrazov ter druge C# in .NET funkcionalnosti. Omogočajo tudi pričakovane rezultate glede na ujemanje argumentov, ujemanje omejitev, vračanje podatkov po meri za preverjanje pričakovanih argumentov na podlagi naše lastne kode, nastavitev akcij na metodah, vračanje specifičnih vrednosti, upravljanje z izjemami in še veliko več. Slika 34 prikazuje uporabo tega ogrodja na entiteti Uporabnik.

```
-var membershipMock = new Mock<IStaticMembershipService>();
-var userMock = new Mock<MembershipUser>();
-membershipMock.Setup(s => s.GetUser()).Returns(user);
-userMock.Setup(u => u.ProviderUserKey).Returns(user.ProviderUserKey);
```

Slika 34 - Primer uporabe Rhino Mock

V naši aplikaciji smo uporabili dve vrsti testov, enotske in integracijske.

Enotski testi

Enotski (ang. unit) testi so elementarni testi, namenjeni testiranju ene same enote kode. To je lahko, na primer, shranjevanje posamezne entitete ali urejanje le-te. Enotski test je del kode (ponavadi metoda), ki kliče drug del kode in nato preveri pravilnost nekaterih domnev. Če se domneve izkažejo za napačne, test ni uspel. Uporabili smo jih pri testiranju naših entitet (Artist, Event, Staff itd.). S tem smo zagotovili, da se entitete pravilno shranijo in zopet pridobijo iz podatkovne baze, da se spremembe pri urejanju odražajo tudi v podatkovni bazi, ipd. Za pomoč smo napisali metode za kreiranje testnih objektov (ang. Factory Methods). Eden izmed enotskih testov naše aplikacije je prikazan na sliki 35, ena od metod za kreiranje testnih objektov pa na sliki 36.

```
-[TestFixture]
-public class Staff2EventTest : BasicPersistenceTest<Model.Event.Staff2Event>
-{
-    .....protected override Model.Event.Staff2Event CreateInstance()
-    .....{
-        .....var testEvent = FactoryMethods.CreateEvent();
-        .....SaveEntity(testEvent);

-        .....var staff = FactoryMethods.CreateStaff();
-        .....SaveEntity(staff);

-        .....var staff2Event = FactoryMethods.CreateStaff2Event(staff, testEvent);

-        .....return staff2Event;
-    .....}
-}
```

Slika 35 - Enotski test iz naše aplikacije

```

public static Advertisement CreateAdvertisement()
{
    var advertisement = new Advertisement
    {
        AdvertisementType = (int)AdvertisementType.Radio,
        Cost = 200,
        Email = "info@radio1.si",
        Name = "Radio 1",
        Notes = "3x na teden ob 15:00",
        Phone = "0145647578",
        Url = "www.radio1.si"
    };

    return advertisement;
}

```

Slika 36 - Metoda za kreiranje testnega objekta iz naše aplikacije

Integracijski testi

Kadar se v testu pojavi odvisnost od neke druge plasti ali modula, enotski test preide v integracijski test. Integracijski testi testirajo aplikacijo po njeni vertikalni strukturi, npr. kako kontroler kliče servisno plast ter kako nato servisna plast pridobi ali shrani podatke v podatkovno bazo. Prav tako smo z njimi testirali prikaz pravih pogledov in povezavo z zunanjimi moduli. Pri tem smo uporabili navidezne objekte, ki oponašajo servise ali zunanje module in nam vračajo vrednosti, ki jih od njih pričakujemo. Na isti način lahko testiramo tudi obravnavanje napak, ki nam jih omenjeni servisi ali modulu lahko vrnejo. Na sliki 37 je prikazano nekaj takih testov iz naše aplikacije.

```

[TestFixture]
public class ArtistControllerTest : ControllerTest<ArtistController>
{
    private Artist _artist;

    protected override ArtistController GetController()
    {
        return new ArtistController();
    }

    [SetUp]
    public void SetUp()
    {
        CreatePersistentUser();
        CreatePersistentArtist();
    }

    private void CreatePersistentArtist()
    {
        _artist = FactoryMethods.CreateArtist();
        SaveEntity(_artist);
    }

    [Test]
    public void Index_renders_view_for_models()
    {
        var controller = GetController();

        var result = controller.Index();

        AssertRendersViewModelCollection<ArtistListModel>(result);
    }
}

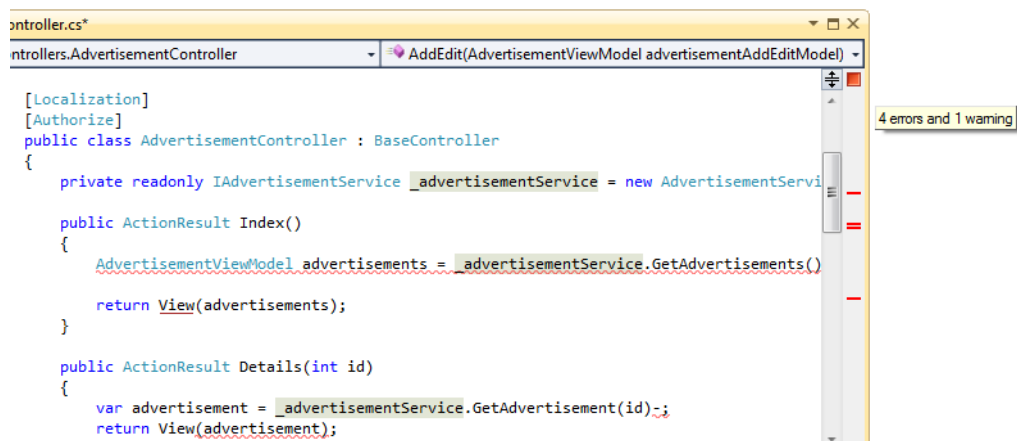
```

Slika 37 - Nekateri integracijski testi iz naše aplikacije

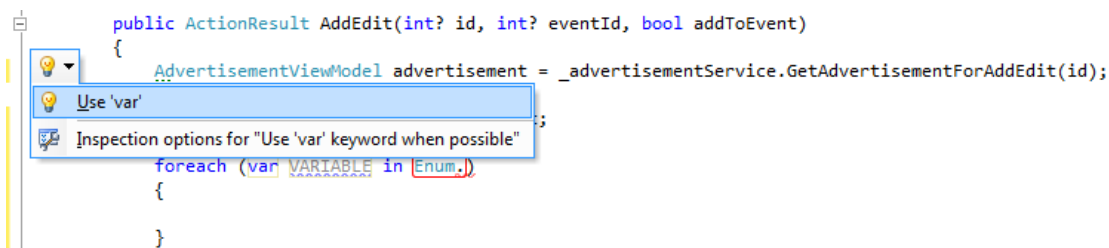
4.7 ReSharper

ReSharper je razširitev za Visual Studio, ki nam olajša refaktoriranje kode in poveča produktivnost. Razvili so ga v podjetju JetBrains. ReSharper izvaja analizo kode po celotni aplikaciji, detekcijo napak ob pisanju, brez potrebe po grajenju kode, zagotavlja dodatne funkcionalnosti za popravljanje napak, dokončevanje kode, navigacijo, iskanje, označevanje

sintakse, formatiranje, generacijo in optimizacijo kode ter podpira NUnit in MSTest za lažjanje enotskega testiranja. Vse te funkcionalnosti so v zadnji različici (7) na voljo za C#, VB11, JavaScript, CSS, HTML, ASP.NET, ASP.NET MVC, Razor, XML in še mnogo drugih [21]. Označevanje napak in opozoril je vidno na sliki 38, na sliki 39 so prikazane še druge zmožnosti orodja.



Slika 38 - Opozarjanje na napake in označevanje mest v kodi, kjer se napake pojavijo



Slika 39 - Uporaba orodja ReSharper za implicitno definiranje spremenljivk in lažje kreiranje zank

4.8 Razor

ASP.NET MVC je vedno podpiral koncept motorjev za poglede (view engines). To so moduli, ki se lahko vklopijo in ponujajo različne predloge sintaks. Privzeti ASP.NET MVC motor je do sedaj uporabljal iste .aspx/.ascx/.master predloge datotek kot ASP.NET Web Forms. Med drugimi popularnimi motorji za poglede, ki so v uporabi danes, sta še Spark in NHaml. Nov motor za poglede, Razor, pa je optimiziran za generacijo HTML-ja s pomočjo predlog na podlagi kode (slika 40). Glavne prednosti Razor-ja so [22]:

- Kompakten, izrazen in tekoč: Razor minimizira število znakov in pritiskov tipk, ki so potrebni v datoteki in s tem omogoči hitrejši in bolj tekoč način programiranja. Za razliko od večine predlog sintaks, nam ni potrebno prekiniti toka programiranja za dodatno definicijo delov kode, ki predstavljajo interakcijo s strežnikom znotraj našega HTML-ja, prevajalnik je namreč dovolj inteligenten, da bo te dele razbral iz naše kode. To omogoči kompaktno, čisto in izrazno sintakso.
- Lahko učenje: Razor je lahek za priučitev in nam omogoči, da smo produktivni v kratkem času z minimalnim naborom konceptov, pri čemer lahko uporabimo naše predhodno znanje zelenega programskega jezika in HTML-ja.

- Intellisense: Čeprav Razor sam po sebi ne potrebuje posebnega urejevalnika in ga lahko prav tako učinkovito uporabljamo v Beležnici, pa nam znotraj Visual Studia nudi odlično podporo za samodokončevanje izrazov.
- Podpora enotskem testiranju pogledov brez potrebe po kontrolerju ali spletnemu strežniku, saj ga lahko gostimo v vsakem enotnem testnem projektu.

```

@Html.ValidationSummary(true)
<fieldset>
    @Html.HiddenFor(model => model.Id)
    <legend>@IOrganize.App_GlobalResources.Resources.Staff</legend>
    <table>
        <tr>
            <td>
                <div class="display-label">
                    @Html.LabelFor(model => model.Name)</div>
                </td>
            <td>
                <div class="display-field">
                    @Html.TextBoxFor(model => model.Name)
                    @Html.ValidationMessageFor(model => model.Name)
                </div>
            </td>
        </tr>
        <tr>
            <td>
                <div class="display-label">
                    @Html.LabelFor(model => model.Type)</div>
                </td>
            <td>
                <div class="display-field">
                    @Html.DropDownListFor(model => model.Type, Model.Types)
                    @Html.ValidationMessageFor(model => model.Type)
                </div>
            </td>
        </tr>
    </table>

```

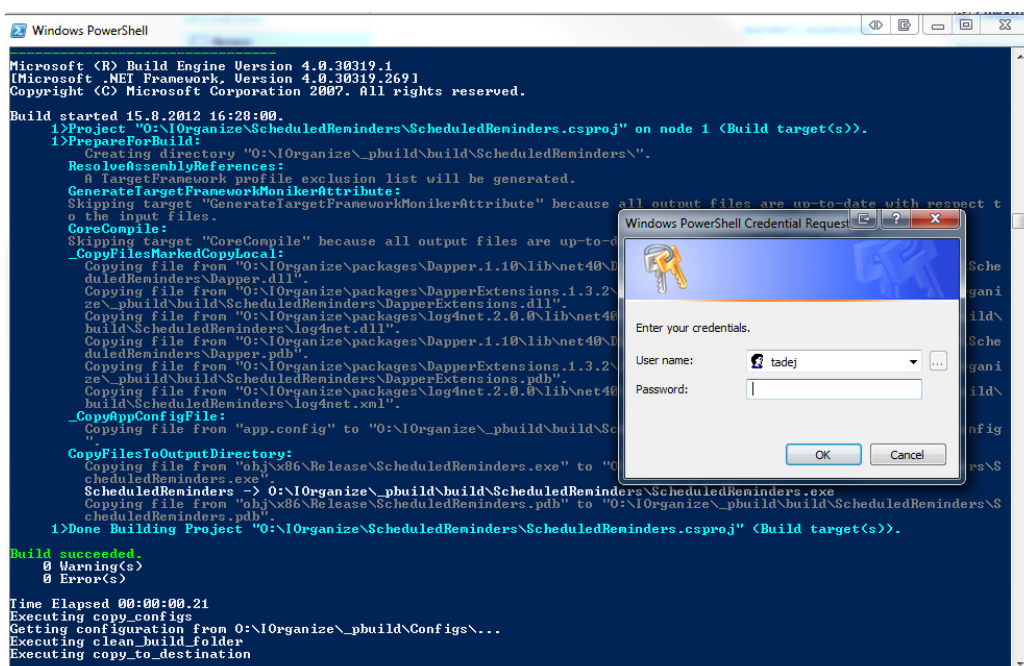
Slika 40 - Koda pogleda za urejanje osebja z Razor sintakso

4.9 Sprotna postavitev na strežniku s pomočjo Powershell in psake

Aplikacijo smo testno postavili na strežniku, da je dostopna tudi izven razvojnega okolja. Želeli smo, da se spremembe v aplikaciji karseda hitro odrazijo tudi na strežniku, da bi lahko potencialni testni uporabniki nove funkcionalnosti sproti preizkusili in nam v najkrajšem možnem času zagotovili povratne informacije. Nadgradnja aplikacije na strežniku zna biti zamudno delo, še posebej ob majhnem številu razvijalcev, ki morajo zaradi tega prekiniti svoj običajni delovni tok, tako da smo poskušali proces čimbolj avtomatizirati. Napisali smo skripto, ki bi z enim klikom to delo opravila namesto nas.

Powershell

Windows Powershell je ogrodje za avtomatizacijo nalog, ki je sestavljeno iz orodne vrstice in z njo povezanih skriptnih jezikov ter integrirano v .NET ogrodje. Vgrajeni ukazi z imenom cmdlets nam omogočajo upravljanje s sistemi iz orodne vrstice [23]. Primer Powershell seje in uspešna izvedba skripta sta prikazani na slikah 41 in 42.



Slika 41 - Primer Powershell seje, uporabnik je bil vprašan po podatkih za dostop

Build Succeeded!	
Build Time Report	
Name	Duration
clean	00:00:00.5924631
generate_AssemblyInfo	00:00:01.9226937
build_projects	00:00:11.6879393
copy_configs	00:00:00.0107623
clean_build_folder	00:00:03.2990583
copy_to_destination	00:01:03.3680503
Total:	00:01:19.1166156

Slika 42 - Uspešno zaključeno izvajanje skripte s poročilom

Psake

psake je domensko specifičen jezik, napisan v Powershell-u, namenjen kreiranju grajenj (ang. builds) z uporabo vzorca odvisnosti, podobno kot Rake ali MSBuild. Skripta za grajenje je sestavljena iz posameznih nalog (Tasks). Vsaka naloga je funkcija, ki lahko definira odvisnosti od drugih Task funkcij. Primer psake skripte [24]:

Task default -Depends Build

```
Task Build -Depends Init,Clean,Compile{
    "build"
}
```

```
Task Compile -Depends Clean, Init {
    "compile"
}
```

```
Task Clean -Depends Init {
    "clean"
}
```

```
Task Init {
    "init"
}
```

psake izvede naloge in vsili odvisnosti med njimi. Ker je psake napisan v pravem programskem jeziku, in ne v XML-ju, imamo veliko več svobode in fleksibilnosti pri pisanju naših skript. Znotraj skripte lahko uporabimo vse funkcionalnosti PowerShell-a in .NET ogrodja.

Skripta

Za namen te diplomske naloge smo napisali skripto, ki nam avtomatizira postopek postavitve na strežniku in kreiranje rezervnih kopij.

```
task default -Depends clean, build_projects, copy_configs, clean_build_folder, copy_to_destination
```

Slika 43 - Naloge skripte

Skripta po vrsti izvede naslednje naloge: počisti, zgradi projekte, kopiraj nastavitve, počisti direktorij za gradnjo, kopiraj na destinacijo.

Počisti (ang. clean)

Naloga počisti osnovni direktorij, da ne pride do prepisovanja datotek ali kakršnihkoli neskladij z verzijami le-teh. Prikazana je na sliki 44.

```
task clean {
    if (test-path "$build_dir\build") { remove-item "$build_dir\build" -force -recurse }
}
```

Slika 44 - Naloga 'Počisti'

Zgradi projekte (ang. build_projects)

Tu se izvede gradnja projektov v naši rešitvi. Naloga je odvisna od druge naloge z imenom 'Zgradi AssemblyInfo datoteke' (ang. generate_AssemblyInfo), ki najprej zgradi vse AssemblyInfo.cs datoteke za vsak projekt (slika 45).

```
task build_projects -Depends generate_AssemblyInfo {
    $projects | foreach {
        write-host "-----" -f green
        write-host ("Building {0} ..." -f $_.name) -f green
        write-host "-----" -f green
        Exec { & msbuild $_.project_file /maxcpucount:$max_compile_cpus /t:Build /p:"Configuration=$configuration;Platform=$platform;OutputPath=$build_dest"
    }
}

task generate_AssemblyInfo {
    $svn_revision = get_svn_revision
    $version = "$major_version.$minor_version.$build_number.$svn_revision"
    write-host "Generating AssemblyInfo.cs files for version $version..." -f green
    $project_dirs = Get-ChildItem -path ..\ -recurse -include "*.csproj" | split-path -parent
    write-host "Checking for non-existing Properties directories..."
    $property_dirs = $project_dirs | foreach-object {"(0)\Properties" -f $_}
    $non_existing = @()
    foreach($dir in $property_dirs) {
        if ((test-path $dir) -eq $false) {
            $non_existing += $dir
        }
    }
    if ($non_existing.Length -gt 0) {
        throw ("Some Properties directories do not exist.\n{0}" -f ($non_existing -join "\n"))
    }
    Get-ChildItem -path ..\ -recurse -include "*.csproj" | % {
        write-host ("Project {0}..." -f $_.Name)
        createAssemblyInfoForProject $_.FullName $version
    }
}
```

Slika 45 - Nalogi 'Zgradi AssemblyInfo datoteke' in 'Zgradi projekte'

Kopiraj nastavitve (ang. copy_configs)

V tem koraku se na ciljno destinacijo kopirajo nastavitve naše aplikacija, kot so povezava na bazo (ang. connection string), vrsta gradnje, ime aplikacije ipd. To je še posebej uporabno v primeru, ko aplikacijo postavljamo na večih strežnikih, saj imamo še vedno lahko samo eno verzijo skripte za postavitve, a še vseeno za vsako postavitev svoje nastavitve. Koda te naloge je vidna na sliki 46.

```
task copy_configs {
    $configs_dir = "$configs_dir\"
    write-host "Getting configuration from $configs_dir..."
    if ((test-path $configs_dir) -eq $false) {
        write-host "Configs folder does not exists. Development defaults will be used!" -f yellow
        return
    }
    copy-item -path "$configs_dir\*" -destination "$website_dir" -recurse -force
}
```

Slika 46 - Naloga 'Kopiraj nastavitve'

Počisti direktorij za gradnjo (ang. clean_build_folder)

Naloga nam počisti direktorij za gradnjo, tako da imamo po njeni izvedbi v tem direktoriju samo datoteke, ki se bodo uporabile za postavitve aplikacije na strežniku (slika 47).

```
task clean_build_folder {
    copy-item -path "$build_dest\ScheduledReminders" -destination "$website_dir\ScheduledReminders" -recurse -force
    if (test-path "$build_dir\build") { remove-item "$build_dir\build\*" -force -recurse -exclude *_PublishedWebsites* }
    copy-item -path "$website_dir\*" -destination "$build_dest" -recurse -force
    remove-item "$build_dir\build\_PublishedWebsites" -force -recurse
}
```

Slika 47 - Naloga 'Počisti direktorij za gradnjo'

Kopiraj na destinacijo (ang. copy_to_destination)

Naloga najprej pridobi podatke o trenutnem uporabniku sistema, s katerimi se poskusi povezati na destinacijo. Če avtentikacija ne uspe, bo uporabnik vprašan po njegovih prijavnih podatkih. Nato izbriše zadnjo varnostno kopijo in ustvari novo ter nazadnje še skopira vse potrebne datoteke na ciljno destinacijo. Naloga je prikazana na sliki 48. Za arhiviranje varnostnih kopij smo uporabili odprtokodno programsko rešitev 7-Zip, ki omogoča arhiviranje v večino znanih zapisov (7z, GZIP, TAR, ZIP in drugi) [25].

```
task copy_to_destination {
    $credentials = Get-Credential -credential $env:username
    write-host "$credentials.Username - $credentials.Password"
    $networkCred = $credentials.GetNetworkCredential()
    $password = $networkCred.Password
    $user = $networkCred.Username
    net use $destination_folder $password /USER:$user

    write-host "Delete last backup..."
    if (test-path "$destination_folder") { remove-item "$destination_folder\*" -force -recurse -include *backup* }

    write-host "Compress the backup..."
    $backup_archive = "$destination_folder\backup_$current_time.zip"
    write-host $backup_archive
    pushd "$destination_folder"
    & $7z_bin a -r $backup_archive "$destination_folder\*"
    popd

    $exclude = @('aspnet_client','*backup*')
    write-host "Copy the new build to the destination..."
    if (test-path "$destination_folder") { remove-item "$destination_folder\*" -force -recurse -exclude $exclude }
    copy "$build_dest\*" "\\$destination_folder" -recurse
    net use * /delete
}
```

Slika 48 - Naloga 'Kopiraj na destinacijo'

4.10 SQL Dapper – Preslikava iz podatkovne baze v objekte

SQL Dapper je preslikovalnik med objekti v objektno orientiranem programskem jeziku in relacijskim modelom. Ta preslikovalnik sicer nameroma ne ponuja nabora funkcionalnosti, kot kateri drugi ORM-ji, na primer LINQ-to-SQL ali Entity Framework. Ponuja nam minimalen vmesnik med našo podatkovno bazo in našo aplikacijo. Ne glede na to pa še vedno vsebuje celoten nabor operacij nad podatkovno bazo - kreiraj (ang. create), beri (ang. read), posodobi (ang. upgrade) in druge [28]. Njegova preprostost in usmerjenost ga uvršča med najhitrejšje ORM-je ta trenutek. Slika 49 prikazuje hitrost preslikave pri več kot 500 iteracijah pri različnih preslikovalnikih, slika 50 pa konkretno uporabo orodja v naši aplikaciji.

Method	Duration
Hand coded (using a SqlDataReader)	47ms
Dapper ExecuteMapperQuery<Post>	49ms
ServiceStack.OrmLite (QueryById)	50ms
PetaPoco	52ms
BLToolkit	80ms
SubSonic CodingHorror	107ms
NHibernate SQL	104ms
Linq 2 SQL ExecuteQuery	181ms
Entity framework ExecuteStoreQuery	631ms

Slika 49 - Hitrost različnih preslikovalnikov [27]

```

public EventViewModel GetEventForAddEdit(int? eventId)
{
    EventViewModel eventViewModel;

    if (eventId == null)
        return new EventViewModel();

    using (var connection = GetOpenConnection)
    {
        var query = new StringBuilder();
        query.AppendLine("SELECT * FROM Event");
        query.AppendLine("WHERE Id = @eventId");

        var eventEntity = connection.Query<Event>(query.ToString(), new { eventId }).ToList().FirstOrDefault();
        eventViewModel = EventMapper.MapEventToEventViewModel(eventEntity);
    }

    return eventViewModel;
}

public void Save(EventViewModel model)
{
    using (var connection = GetOpenConnection)
    {
        Event eventModel;
        if (model.Id > 0)
        {
            eventModel = connection.Get<Event>(model.Id);
            EventMapper.Update(model, eventModel);
            connection.Update(eventModel);
        }
        else
        {
            eventModel = EventMapper.Create(model);
            connection.Insert(eventModel);

            var user2Event = new User2Event(CurrentUserId, eventModel.Id);
            connection.Insert(user2Event);
        }
    }
}

```

Slika 50 - Uporaba SQL Dapper-ja v naši aplikaciji za pridobivanje entitete iz podatkovne baze in za shranjevanje le-te

4.11 Večjezičnost

Lokalizacija je običajna težava, s katero se srečujemo pri razvoju spletnih aplikacij. Bistvo pri razvijanju večjezične aplikacije je ločevanje vsebine strani in poslovne logike. To pomeni, da vsebine nikoli ne prikazujemo direktno na strani, ampak ji dodelimo ključ, ki ga nato lahko povežemo z dejansko vsebino za pravo jezično nastavitve [29].

ASP.NET.MVC je bil zgrajen na podlagi ASP.NET, kar pomeni, da lahko uporabimo vse njegove funkcionalnosti. ASP.NET spletni obrazci uporabljajo datoteke z viri (eng. resource files) za shranjevanje vsebine strani v različnih jezikih. Z uporabo razreda ResourceManager, ki ga orodje Visual Studio ustvari avtomatsko, lahko te vsebine pridobimo in uporabimo na naši strani. Zato smo v naši aplikaciji ustvarili direktorij App_GlobalResources, ki vsebuje vse datoteke z viri za različne jezike. V našem primeru sta to dve datoteki - ena za privzeti jezik, ki je angleščina, in druga za slovenski jezik (slika 51). Prvo smo poimenovali Resources.resx, drugo pa Resources.sl.resx, kjer je 'sl' kultura tega jezika.

Name	Value
AccommodationExpenses	Accommodation expenses
AccommodationRequirements	Accommodation requirements
AddAdvertisement	Add advertisement
AddArtist	Add artist
AddDocument	Add document
AddEdit	Add/Edit
AddEvent	Add event

Name	Value
AccommodationExpenses	Stroški nastanitve
AccommodationRequirements	Zahteve nastavitve
AddAdvertisement	Dodaj oglas
AddArtist	Dodaj nastopajoče
AddDocument	Dodaj dokument
AddEdit	Dodaj/Uredi
AddEvent	Dodaj dogodek

Slika 51 - Datoteki z viri v angleškem in slovenskem jeziku

Ne glede na to, da imamo sedaj vse izraze, ki se bodo pojavili na uporabniškem vmesniku, v obeh jezikih, pa naša aplikacija še vedno ne podpira lokalizacije, saj nimamo načina, kako definirati jezično nastavitvev. Zavoljo preprostosti smo to storili preko povezave, kar pomeni, da bo <http://localhost/en-US/Home/Index> povezava do angleške domače strani, <http://localhost/sl-SI/Home/Index> pa povezava do slovenske. Uporabnik lahko jezik spremeni kjerkoli na strani, prav tako pa lahko poda povezavo do določene strani skupaj z nastavitvijo jezika. Da smo to dosegli, smo spremenili usmerjanje v naši aplikaciji (slika 52).

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.MapRoute("Localization",
        "{lang}/{controller}/{action}/{id}",
        new { controller = "Event", action = "Index", id = UrlParameter.Optional });

    routes.MapRoute("Default",
        "{controller}/{action}/{id}",
        new { controller = "Event", action = "Index", id = UrlParameter.Optional });
}
```

Slika 52 - Nova pot za lokalizacijo v naših nastavitvah usmerjanja

Napisali smo še atribut z imenom `LocalizationAttribute`, ki deduje od `ActionFilterAttribute` atributa in prepiše njegovo `OnActionExecuting` metodo. Naša nova metoda preveri podatke o usmeritvi in glede na njih določi jezik. Če podatkov ni, se uporabi privzeta nastavitvev. Atribut smo nato uporabili na naših kontrolerjih (slika 53), kar je povzročilo, da so vse akcije uporabljale logiko za lokalizacijo.

```
[Localization]
[Authorize]
public class EventController : BaseController
{
    private readonly IEventService _eventService = new EventService();
    private readonly IAdvertisementService _advertisementService = new AdvertisementService();
}
```

Slika 53 - Uporaba 'Localization' atributa na kontrolerju

Da lahko uporabnik na enostaven način zamenja jezik, smo na vrh strani dodali povezave, ki to počnejo. Napisali smo razred, ki hrani informacije o povezavah za jezike in te povezave tudi kreira. Da so povezave vidne ves čas, smo jih dodali v glavno stran. Težava se je pojavila še pri strani za prijavo v aplikacijo. To se je zgodila zaradi naših poti. Naša pot za lokalizacijo ima štiri parametre - jezik, kontroler, akcija in identifikator, privzeta pot pa 3 - kontroler, akcija in identifikator. Če imamo povezavo *http://localhost/en-US/Home/Index*, se le-ta ujema z našo potjo za lokalizacijo (jezik=en-Us, kontroler=Home, akcije=Index, identifikator ni obvezen). V primeru povezave *http://localhost/* pa se le-ta ujema s privzeto potjo, pri kateri so lahko parametri tudi nični. Če pa pogledamo povezavo za prijavo v aplikacijo (*http://localhost/Account/Login*), se bo le-ta zopet ujemala z našo potjo za lokalizacijo, saj sta parametra akcija in identifikator opcijska. Končni rezultat bo, da bo aplikacija mislila, da je 'Account' jezik, 'Login' kontroler, 'Index' pa se bo uporabila kot privzeta akcija. Kontroler z imenom 'Login' pa v našem primeru ne obstaja, kar nam vrne napako. Ker je povezava za prijavo avtomatsko preusmerjena s strani ASP.NET sistema za avtorizacijo, ne moremo vanjo dodati podatkov o jeziku, tako da smo morali za ta poseben primer definirati novo pot, prikazano na sliki 54.

```
routes.MapRoute("Login",
    "Account/{action}",
    new { controller = "Account", action = "Login" });
```

Slika 54 - Posebna pot za prijavo

4.12 Validacija

V aplikaciji smo uporabili ASP.NET MVC validacijo podatkov. V naših modelih za prikaz (ang. view models) smo na vsako lastnost dodali anotacijo (slika 56), ki pove, kakšna validacija se mora nad to lastnostjo izvesti. Za preverjanje pravilnosti elektronskega naslova smo uporabili regularne izraze (ang. regular expressions), za tekstovna polja privzet atribut za dolžino teksta (ang. string length), za datume atribut tipa podatka (ang. data type), za vsa obvezna polja pa atribut obvezno (ang. required). Vsa sporočila o morebitnih napakah smo lokalizirali, da so prikazana v trenutno izbranem jeziku. Opozorila o napačnih vnosih se prikažejo desno od problematičnega polja, kot je prikazano na sliki 55.

Accommodation expenses	80 error	The field Accommodation expenses must be a number.
Accommodation requirements	4*	
OtherExpenses	0	
Country	Italy	
Email	pirupa@error	Please enter a valid e-mail address

Slika 55 - Opozarjanje na napačne vnose


```

private static void SendEmail(string emailAddress, string emailSubject, string emailBody)
{
    var mail = new MailMessage();
    var smtpServer = new SmtpClient("smtp.gmail.com");

    mail.From = new MailAddress("iorganizeevents@gmail.com");
    mail.To.Add(emailAddress);

    mail.Subject = "IOrganize reminder " + emailSubject + " - " + DateTime.Now;
    mail.Body = emailBody;

    smtpServer.Port = 587;
    smtpServer.Credentials = new NetworkCredential("iorganizeevents@gmail.com", "password");
    smtpServer.EnableSsl = true;

    try
    {
        smtpServer.Send(mail);
    }
    catch (Exception ex)
    {
        Log.Error(String.Format("Error sending email! \n {0}", ex.Message));
    }
}

```

Slika 57 - Funkcija za pošiljanje elektronske pošte

5 Zaključek

V diplomskem delu smo obravnavali področje organizacije posebnih dogodkov. V okviru tega smo razvili spletno aplikacijo, katere namen je olajšanje posameznih vidikov organizacije in zbiranje vseh potrebnih informacij na enem mestu. Za osnovo smo uporabili ogrodje Microsoft .NET. Uporabili smo tudi ogrodje ASP.NET MVC 4, ki je v času pisanja tega dokumenta še v fazi kandidat za izdajo, vendar nam je zagotovil funkcionalnosti, ki so nam olajšala razvoj in nam zagotovile, da bo aplikacija tehnično aktualna še dalj časa. Da pa bo aplikacija vzdrževana in da tudi pri nadaljnjem razvoju ali popravkih ne bomo imeli večjih težav, pa nam je zagotavljal princip testno vodenega razvoja. Za vse dele aplikacije smo napisali teste, kar nam je zagotovilo bolj čisto in namensko kodo že v času razvoja ter nam bo v prihodnosti olajšalo izboljšavo le-te.

Področje organizacije dogodkov se sicer v zadnjem času zelo hitro razvija, vseeno pa je veliko stvari še odprtih in nedorečenih. Čeprav obstajajo zakonska določila za prijavo dogodka in potrebna dovoljenja za njegovo izvedbo, je še vedno potrebno veliko pogajanj s pristojnimi oblastmi. Organizacija dogodkov je ta trenutek še vedno bolj umetnost kot znanost. To je predvsem zaradi različne narave dogodkov, ki nam onemogočajo načrtovanje pravil pri njihovi organizaciji. Vseeno se lahko opiramo na priporočila, navedena v tem diplomskem delu, ki so bila napisana z namenom, da bi ob istočasni uporabi razvite aplikacije potencialnim organizatorjem poenostavila vsaj nekatere vidike organizacije.

6 Literatura in viri

- [1] Anton Shone, Bryn Parr , *Successful Event Management: A Practical Handbook*, London: Thomson Learning, str. 3 – 4, 2004.
- [2] Harris Goldberg, Adam Tobey, Mike Russo, Adam Taylor, Dave Stevens, Amanda Campbell, *Concert Ideas Event Planning Guide*, Woodstock, NY: Manning Concert Ideas, Inc., str. 1- 12, 2010.
- [3] Event management. Dostopno na: http://en.wikipedia.org/wiki/Event_management
- [4] Kako organizirati prireditve. PDF dokument, dostopno na: <http://leader.lto-blegos.si/apl/doc/37FDE17F.pdf>
- [5] (2007) Organizacija dogodkov od A do Ž. PTP dokument, dostopno na: http://www.nevladnik.info/media/objave/dokumenti/2011/05_Organizacija_dogodkov.ppt
- [6] (2007) Informatika v medijih. PDF dokument, dostopno na: http://mafalda.informatika.uni-mb.si/2007-08/ivm/predavanja/10-primeri_uporabe.pdf
- [7] (2012) SOLID (object-oriented design). Dostopno na: [http://en.wikipedia.org/wiki/SOLID_\(object-oriented_design\)](http://en.wikipedia.org/wiki/SOLID_(object-oriented_design))
- [8] (2012) Single responsibility principle. Dostopno na: http://en.wikipedia.org/wiki/Single_responsibility_principle
- [9] (2012) Open/closed principle. Dostopno na: http://en.wikipedia.org/wiki/Open/closed_principle
- [10] (2012) Liskov's Substitution Principle(LSP). Dostopno na: <http://www.oodeesign.com/liskov-s-substitution-principle.html>
- [11] (2012) Interface segregation principle. Dostopno na: http://en.wikipedia.org/wiki/Interface_segregation_principle
- [12] (2012) Dependency inversion principle. Dostopno na: http://en.wikipedia.org/wiki/Dependency_inversion_principle
- [13] (2007) .NET Framework diagram. Dostopno na: <http://en.wikipedia.org/wiki/File:DotNet.svg>
- [14] (2009) NerdDinner predstavitevni projekt, napisan v MVC arhitekturi. Dostopno na: <http://nerddinner.codeplex.com/>
- [15] (2012) ASP.NET MVC 4. Dostopno na: <http://www.asp.net/whitepapers/mvc4-release-notes>
- [16] (2012) NUnit. Dostopno na: <http://www.nunit.org/index.php?p=documentation>

[17] Roy Osherove, *The art of unit testing with examples in .NET*, Greenwich: Manning Publications Co., pogl. 1, 2009.

[18] (2012) .NET Framework. Dostopno na: http://en.wikipedia.org/wiki/.NET_Framework

[19] (2012) Test driven development. Dostopno na: http://en.wikipedia.org/wiki/Test-driven_development

[20] (2009) Test-driven development. Dostopno na: http://en.wikipedia.org/wiki/File:Test-driven_development.PNG

[21] (2012) JetBrains ReSharper. Dostopno na: <http://www.jetbrains.com/resharper/>

[22] Jess Chadwick, *Programming Razor*, Sebastopol: O'Reilly Medica, Inc., pogl. 1, 2011.

[23] (2008) Windows PowerShell. Dostopno na: http://en.wikipedia.org/wiki/Windows_PowerShell

[24] (2012) Psake. Dostopno na: <http://en.wikipedia.org/wiki/Psake>

[25] (2012) 7-Zip. Dostopno na: <http://www.7-zip.org/>

[26] (2011) Revision control. Dostopno na: http://en.wikipedia.org/wiki/Revision_control

[27] (2012) SQL Dapper. Dostopno na: <http://stackoverflow.com/tags/dapper/info>

[28] (2012) dappet-dot-net. Dostopno na: <http://code.google.com/p/dapper-dot-net/>

[29] (2012) Localization in ASP.NET MVC – 3 Days Investigation, 1 Day Job, Dostopno na: <http://geekswithblogs.net/shaunxu/archive/2010/05/06/localization-in-asp.net-mvc-ndash-3-days-investigation-1-day.aspx>