SVEUČILIŠTE U ZAGREBU

FILOZOFSKI FAKULTET

ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKEZNANOSTI

Ak. god. 2017./2018.

Ivan Grobenski

Suvremeni pristupi razvojuaplikacija

Diplomski rad

Mentor: doc. dr. sc. Vedran Juričić

Zagreb, srpanj2018.

Sadržaj

1 Uvod 1
2 Aplikacija 2
2.1 Osnovni pojmovi 2
2.2 Tipovi aplikacija 2
2.3 Razvoj aplikacija 4
3 Korisnički zahtjevi i prototipiranje 6
3.1 Korisničke priče 6
3.2 Specifikacije 6
3.2.1 Formalna specifikacija 7
3.2.2 Funkcionalna specifikacija 8
3.2.3 Tehnička specifikacija 9
3.2.4 Specifikacija održavanja aplikacije 11
3.3 Dokumentacija 13
3.3.1 Korisničke upute 13
3.4 Dizajn 13
4 Metodologije razvoja aplikacija 16
4.1 DevOps 17
4.2 SCRUM 18
4.3 IID 19
4.4 FDD 20
5 Tehnički pristup razvoju aplikacija 21
5.1 Principi 21
5.1.1 SOLID 21
5.1.2 DRY 23
5.1.3 KISS 23
5.1.4 Princip paketa 23
5.1.5 GRASP 24
5.2 Oblikovni obrasci 24
5.3 Arhitektura koda 25
5.4 Domain driven design 26
5.5 Test driven development 27
5.6 Acceptance test-driven development 27
5.7 Behaviour driven development 28

5.8 Sustavi za verzioniranje koda 29
6 Testiranje 31
6.1 Unit testovi 31
6.2 Testovi izdržljivosti 32
6.3 Penetracijski testovi 32
6.4 Funkcionalni regresijski UI testovi 32
7 Alati za lakši razvoj u timovima 34
7.1 Trello 34
7.2 TFS 34
7.3 Jira 35
7.4 Slack 36
8 Alfredo : aplikacija za povezivanje IT studenata i poslodavaca u RH 38
8.1 Tehnologija 39
8.2 Arhitektura 39
8.2.1 Projekt infrastrukture 40
8.2.2 Projekt modela 40
8.2.3 Projekt repozitorija 41
8.2.4 Util projekt 42
8.2.5 MVC projekt 42
9 Zaključak 43
10 Literatura 44
11 Prilozi 50

Popis slika

Slika 1 Generalni koraci u procesu oblikovanja aplikacije 5
Slika 2 Primjer dijela formalne specifikacije GKS-a (engl. Graphical Kernel System) 7
Slika 3 Primjer grananja mogućih akcija u aplikaciji kao dio funkcionalne specifikacije 9
Slika 4 Primjer opisa funkcionalnosti kombinacijom slike i opisa koraka 9
Slika 5 Dio obrasca za izradu tehničke specifikacije aplikacije 11
Slika 6 Popis najbitnijih poglavlja specifikacije za održavanje Alfredo aplikacije 12
Slika 7 Primjer izgleda žičanog okvira (wireframea) aplikacije 14
Slika 8 Primjer teme izrađene u photoshopu 15
Slika 9 Koncept slika lika za video igru 15
Slika 10 Dijelovi razvoja koje obuhvaća DevOps. 17
Slika 11 Opći koraci rada po SCRUM metodologiji razvoja aplikacija 19
Slika 12 Model iterativnog razvoja u procesu razvoja aplikacija 20
Slika 13 Universal Inventory System inventarski paket za Unity pogonski sklop igre 24
Slika 14 Given when then test implementiran u programskom jeziku C# 29
Slika 15 Shematski prikaz promjena izvornog koda u Git sustavu 30
Slika 16 Trello i njegov princip rada s karticama. 34
Slika 17 Izgled korisničkog sučelja TFS-a. 35
Slika 18 Ploče sa zadacima na Jiri 36
Slika 19 Izgled korisničkog sučelja Slacka 37
Slika 20 Izgled naslovne strane Alfredo aplikacije 38
Slika 21 Arhitektura Alfreda u Visual Studiju 40
Slika 22 Klase modela spremljene su u Models projektu unutar rješenja 41
Slika 23 Pregled mapa i pripadajućih klasa repozitorijskog projekta aplikacije 42

1 Uvod

Tema ovog diplomskog rada su različiti pristupi razvoju suvremenihaplikacija. S obzirom na složenost samog procesa razvoja jedne aplikacije, opisano je više različitih aspekata razvoja. Radkreće od korisnika, kao entiteta koji ima potrebu za samom aplikacijom,softverom koji će riješiti konkretan problem ili skup problema, a potom se opisuje proces oblikovanja sa strane metodologije razvoja i s tehničke strane, u kojoj je fokus na modernim principima razvoja, oblikovnim obrascima, arhitekturi koda, tehnologijama te razvojnim okvirima.Također,opisuju se i drugiaspekati, uže ili šire, vezani za razvoj aplikacija u suvremenoj IT industriji. Neki od tih aspekata su testiranje, razni alati koji olakšavaju razvoj aplikacije u timu, itd.

Za potrebe ovog rada razvijen je *Alfredo*, web aplikacija čiji je ciljspojiti poslodavce i studente IT usmjerenja u Hrvatskoj. Koristeći navedenu aplikacijuopisane su ideje, metode i koncepti spomenuti u prethodnim poglavljima rada, a kojizaokružujurad u cjelinu.

2 Aplikacija

U ovom poglavlju definirani su osnovni, najbitniji, pojmovi korišteni u radu, navedene su podjele aplikacija i opisane sudimenzije razvoja aplikacija.

2.1 Osnovni pojmovi

Merriam-webster online rječnik definira aplikaciju kao program koji izvršava specifičan zadatak ili skup zadataka (Merriam-webster, 2018).Aplikacije se mogu podijeliti prema nekolikokriterija, a neke od mogućih podjela opisane su u sljedećem potpoglavlju.

Pojam razvoja, u ovome radu, se ne odnosi samo na pisanje koda, već na cijeli proces oblikovanja aplikacije – od prikupljanja korisničkih zahtjeva,preko izrade nacrta, specifikacija i dokumentacija pa sve do pisanja koda kao tehničkog dijela razvoja te testiranja i dostavljanja proizvoda krajnjem klijentu.

Metodologija(Carmel, 1999) se odnosi na organizaciju timova i grupiranje određenih procesa u sustave te na izradu obrazaca ponašanja s ciljem bržeg, efektivnijeg i boljeg rješavanja problema. To jegruba definicija iz Carmelove knjige, a sličnu je imao i Herbert D. Benington (1956.) koji prvi put uopće i spominje ovaj pojam u kontekstu razvoja softvera (Boehm, 1988).Na dijelovima rada, definicija se ponovno spominje radi lakšeg uklapanja u detaljniji kontekst poglavlja.

Značajka (*engl.feature*) je dio aplikacije koji može funkcionirati sam za sebe (Palmer&Felsing, 2002). Na primjer, jedna od značajki aplikacije koja ima funkciju oglasa je sustav slanja poruka među registriranim korisnicima,druga mogućnost registracije i prijave, a treća mogućnost uređivanja postavki profila. Često se spominje u poglavlju koji govori o razvoju aplikacija temeljenom na značajkama.

2.2 Tipovi aplikacija

Prema EDUCBA (2015) aplikacije se mogu podijeliti na :

- Poslovne programe koji rješavaju poslovne probleme neke organizacije. Na primjer interna aplikacija za praćenje potrošenih satnica zaposlenika neke kompanije na temelju koje se onda isplaćuju plaće.
- Programe za pristup sadržaju. Na primjer tražilice i medijski izvođači.

- Suite primjenjivih programa, skup programa povezanih i sličnih funkcija, obilježja i izgledakorisničkog sučelja (Microsoft office paket, Adobe paket, itd.)

- Obrazovne programe, „edukativne aplikacije“ - Učilica, kao primjer edukativne video igre.

- Programe za simulaciju. Na primjer simulacija operacije srce u okolini virtualne stvarnosti (*engl. virtualreallity - VR*)kao metoda treninga budućih kardiokirurga ili simulacija postupka postavljanja kompleksne konstrukcije na građevinsku lokaciju putem tehnologija proširene stvarnosti (*engl. augmentedreality - AR*) za lakšu vizualizaciju procesa.

- Programe za izradu medija (videa, slike, glazbe, animacija, 3D modela i slično). Na primjer Blenderza 3D modeliranje, Photoshopza rastersku obrada slike, Adobe Illustratorza vektorsku grafiku, Sony Vegas pro paket za uređivanje i izrada videa te FL studio za produkciju glazbe.

- Mobilne primjenjive programe, „mobilne aplikacije“, programe koji se izvršavaju na pametnim telefonima. Na primjer Facebook messenger (iOS/Android), Google Chrome (iOS/Android))

- Zabavni softver. Toj kategoriji pripadaju video igre, programi za reprodukciju videa, glazbe ili pregled slika i animacija, čuvari zaslona (*engl.screensavers*) i slično.

- Programe za izradu programa. Uključuju stvaranje potpomognuto računalom i dizajn pomognut računalom. Primjer suVisual studio, Android studio, PyCharm, Unity game engine, Unreal game engine, itd.

Osim te podjele, aplikacije se još mogu podijeliti i prema operacijskom sustavu na kojem se izvršavaju ili pokreću (Linux, Windows, Mac, itd.), tipu platforme (mobilna, računalna,mikrokontroleri, televizori, uređaji posebne namjene i sl) ili s obzirom na mrežnu arhitekturu (aplikacije u oblaku, interne lokalne aplikacije, itd.). Osim toga i sami operacijski sustavi se mogu smatrati aplikacijama(Tanenbaum, 1987). Također, aplikacije nemoraju nužno imati grafičko sučelje kako bi bile aplikacije po definiciji, pa s obzirom na to i API-i (*engl. applicationprogramminginterface*) i programi koju se pozivaju u pozadini putem CLI-a (*engl. command-line interface*) se također smatraju aplikacijama (Moran, 1981). Informacijski sustavi su također aplikacije jer je njihova zadaća dostaviti relevantnu informaciju na zahtjev

korisnika, a što ne bi mogli bez pozadinskog hardvera i softvera temeljenog na nekoj tehnologiji (Pawlak, 1981).

2.3 Razvoj aplikacija

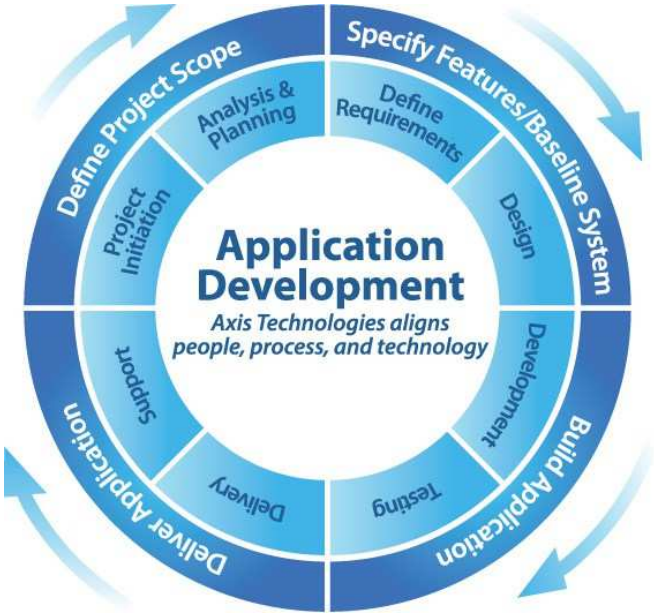
Fokus ovoga rada je na aplikacijama koje se razvijaju u profesionalnim IT okruženjima, etabliranim firmama koje posluju po već isprobanim načelima razvoja aplikacija. S obzirom na to, razvoj aplikacija se u kontekstu ovoga rada odnosi na razvoj aplikacija unutar takvih kompanija.

Naravno da svaka kompanija ima neke svoje metode razvoja no, generalno, industrija se slaže da postoje neka dobra načela kojih se treba pridržavati ukoliko je cilj dostaviti aplikaciju što brže, efikasnije i što veće relevantnosti i iskoristivosti za krajnjeg korisnika (Allen &Chudley, 2012, Carmel, 1999,Hecksel, 2004).

Prvi korak suvremenih tvrtki u tom procesu jest prihvaćanjezahtjeva korisnika u obliku nekog problema koji žele riješiti aplikativno, odnosno softverski (Ein-Dor, Myers, Raman, 2004). Konkretan korak detaljnijeje opisan u narednom poglavlju.

Nakon prihvaćanja i analize zahtjeva, raspisuje se specifikacija i dizajneri rade nacrte rješenja (Ein-Dor, Myers, Raman, 2004), odnosno nacrte i prototipe na temelju kojih će se izraditi konačnorješenje. Krajnje rješenje razvijaju programeri kroz pisanje koda, razvoj i postavljanje baze te analizu serverske arhitekture i mrežnih rješenja skupa s sistemskim inženjerom. Svaka isprogramirana značajka aplikacije mora se i testirati, tako da nakon razvoja značajke kroz pisanje koda, slijedi testiranje iste (Saff& Ernst, 2004). Ako se testiranjem pokazalo kako značajka radi očekivano, ista postaje dio produkcijske verzije aplikacije. S druge strane, ako je testiranjem otkrivena neka nepravilnost, značajka se vraća nazad na doradu i usavršavanje. Proces pisanjakoda i testiranja je iterativan dokle god se značajka u potpunosti ne implementira.

Za bolje razumijevanje procesa razvoja jedne aplikacije, može se pogledati dijagram na slici 1.



Slika 1 Generalni koraci u procesu oblikovanja aplikacije[1]

U ovom radu su redom opisani svi koraci sa slike 1, a nešto veća pažnja je dana tehničkom aspektu razvoja kako bi se s većim razumijevanjem moglo pristupiti poglavlju u kojem se opisuje praktični dio rada, tj web aplikacija razvijena za potrebe istog.

[1]Preuzetosa: http://prevoyancetech.com/services/software-application-development/

3 Korisnički zahtjevi i prototipiranje

Kako bi se što efikasnije moglo procijeniti vrijeme razvoja aplikacije te izračunati troškove provođenja projekta, na početku je potrebno odraditi puno savjetovanja s klijentom, odnosno naručiteljem aplikacije (Albrecht&Gaffney, 1983). Sam klijent ne mora nužno biti i korisnik aplikacije, ali mora moći opisati problem koji želi riješiti, što mu odgovara,a što ne i slično. Na temelju dosadašnjeg radnog iskustva u IT kompanijama u Hrvatskoj, proces komunikacije s klijentom, odnosno naručiteljem aplikacije, jest zapravo iterativan proces. U praksi klijent često mijenja svoje ideje i manevrira od osnovne zamisli dogovorene na inicijalnim sastancima. Takve stvari je često nemoguće izbjeći pa ih treba imati na umu od samog početka, budući da troše vrijeme i iziskuju dodatne napore i energiju. Također, u kompanijama koje rade po SCRUM metodologiji razvoja softvera, u svakoj razvojnoj iteraciji (tzv. *sprintu*) postoji vrijeme određeno samo za dorade značajki na temelju iskomuniciranih potrebnih modifikacija s klijentom. Više o SCRUM metodologiji i takvom načinu rada više se piše u njemu posvećenom potpoglavlju ovog rada.

U kontekstu početnih dogovora s klijentom i prvih koraka u projektu oblikovanja nove aplikacije, treba spomenuti ulogu korisničkih priča, specifikacija i nužnosti dokumentiranja dogovorenih procesa, izmjena i značajki.

3.1 Korisničke priče

Korisničke priče (*engl.userstories*) olakšavaju razumijevanjeznačajki koje treba implementirati u aplikaciji svojim „*ja kao korisnik želim*“ formatom(Cohn, 2004).Dakle radi se o izjavama napisanih na neformalnom, prirodnom, jezikuiz pozicije korisnika koji želi neku značajku u aplikaciji. Obično ih izrađuju voditelji projekata, analitičari, klijenti ili pak sami korisnici (Cohn, 2004). Primjer korisničke priče za aplikaciju koja se bavi oglasima, poput Njuškala, bila bi : „*Kao korisnik aplikacije Njuškalo želim mogućnost slanja poruka drugim korisnicima i upita vezanih za njihove oglase*“.

3.2 Specifikacije

Specifikacije predstavljaju opis funkcionalnostikoje je potrebno razviti kako bi aplikacija mogla izvršavati svoje osnovne i dodatne namjene (IEEE, 1984). Specifikacije aplikacije propisuju obično funkcionalne i ne funkcionalne zahtjeve, odnosno zahtjeve koji će činiti srž aplikacije i bez kojih ona ne može obavljati svoju osnovnu funkcionalnost, te „bonus“

značajke, značajke koje nisu neophodne za funkcioniranje aplikacije, ali možda ubrzavaju neki poslovni ili tehnički proces (IEEE, 1984). Postoji više različitih tipova specifikacija kao što su funkcionalne specifikacije, formalne specifikacije, tehničke specifikacije te specifikacije održavanja aplikacije (Spolsky, 2000). Svaka od njih je opisana u veće detalje u narednim potpoglavljima.

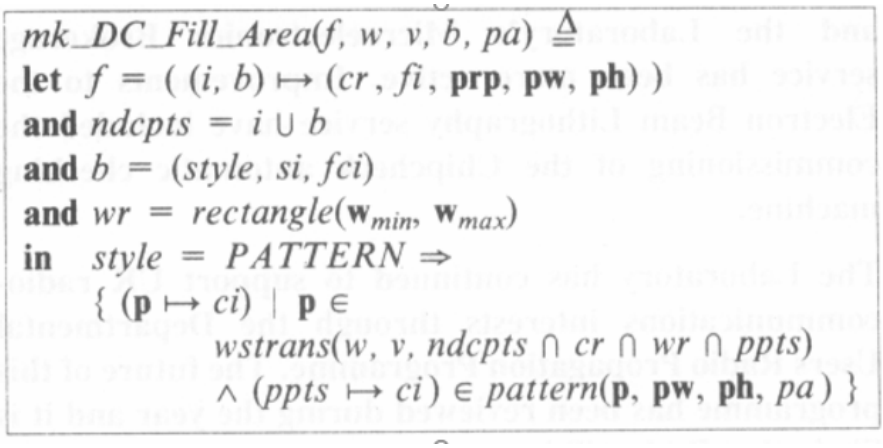
3.2.1 Formalna specifikacija

Formalna specifikacija je skup opisnih svojstava sustava koje za cilj imaju olakšati dizajn i razvoj sustava te analizirati njegove performanse, a formalnima se nazivaju jer imaju svojevrsnu sintaksu koju prate te format (IEEE, 1984).

Ispravno napisana formalna specifikacija se može iskoristiti za izradu implementacija značajki koji će činiti sustav koji opisuje te je stoga jako bitno da je ona efektivna u svojoj analizi, iskoristiva, lako održiva te da efikasno komunicira ciljeve projekta (IEEE, 1984)

Nedostatak formalnih specifikacija je upravo i njihova prednost – formalnost. Naime, velika većina suvremenih tvrtki koristi metode agilnog timskog razvoja, a one iziskuju određenu razinu fleksibilnosti koja se gubi formaliziranjem procesa (Boehm, 1988). Osim toga, skupa je za održavanje i iziskuje određeno poznavanje matematike budući da je uglavnom pišu matematičari ili analitičari (Boehm, 1988). Kao jedno od rješenja predlaže se izrada alata koje će sakriti te matematičke metode u pozadini, kako bi ju mogli izrađivati i dorađivati i drugi članovi tima.

Primjer formalne specifikacije vidljiv je na slici 2.



Slika 2 Primjer dijela formalne specifikacije GKS-a (*engl*. *GraphicalKernel System*)[2]

[2] preuzeto sa: http://www.chilton-computing.org.uk/inf/literature/inf_annual_reports/p003.htm

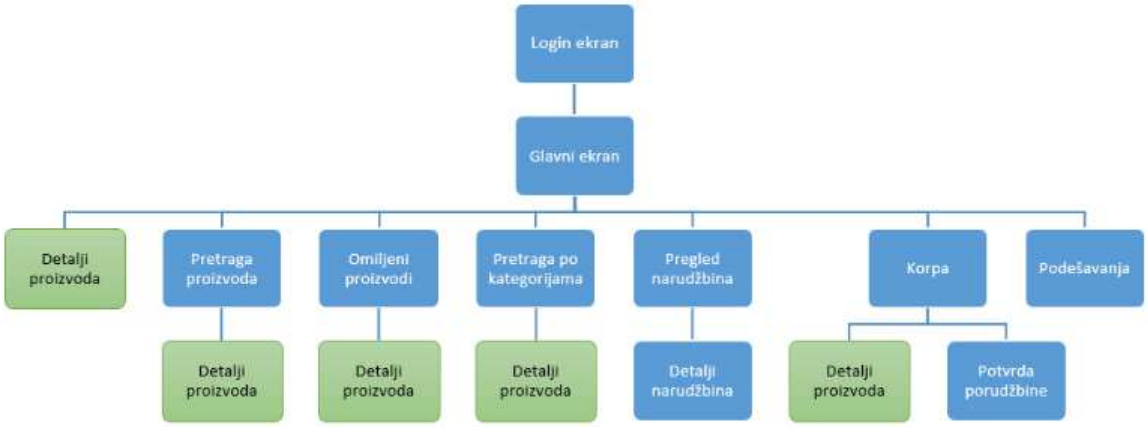
3.2.2 Funkcionalna specifikacija

Funkcionalna specifikacija, ugrubo,predstavlja opis kako softver treba raditi, odnosno što ne bi trebao raditi (Spolsky, 2000). To nije dokument napisan isključivo za tehničke stručnjake, već i za ostale članove tima koji moraju biti upućeni u ono što aplikacija radi (Spolsky, 2000).

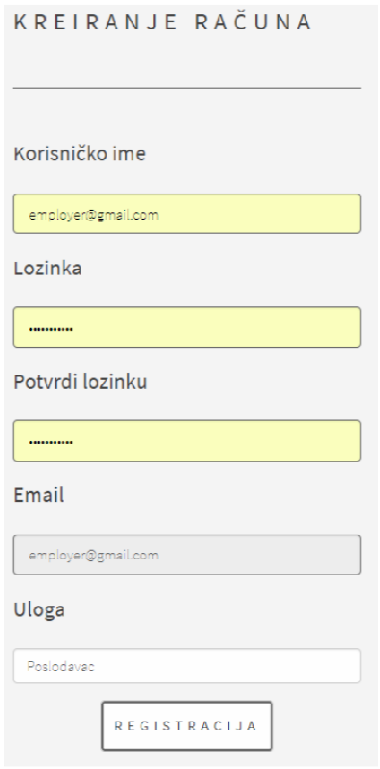
Izrada funkcionalne specifikacije je nužnane samo za one koji razvijaju aplikaciju, već i za klijenta, odnosno naručitelja, osobu koja specificira svoje zahtjeve, kako bi se u bilo kojem trenutku poznavao smjer kretanja razvoja (Beciric, 2015).

Zahvaljujući postojanju funkcionalne specifikacije, mogu se raspisati troškovi projekta, ugrubo predvidjeti vrijeme razvoja pojedinih značajki, procijeniti nastanak eventualnih problemate za programere ono najbitnije, mogućnost uvida usmjerrazvojau bilo kojoj fazi razvoja. Naime, često se u IT kompanijama (pogotovo manjim), zanemari ovaj aspekt razvoja, pa onda novopridošli programeri troše vrijeme analitičara, testera, drugih programera ili pak samih voditelja projekata, kako bi im objasnili ulogu i svrhu postojanja pojedinih dijelova sustava,a to sve samo zato što funkcionalna specifikacija ne odražava trenutno stanje aplikacije (nije ažurirana prolaskom vremena i dolaskom novih značajki) ili je nepostojeća (pogtovou manjim tvrtkama koje ne mogu odvojiti resurse za izradu specifikacija). Izrazito je frustrirajuće raditi na implementaciji značajke za koju programer ne zna kako se uopće uklapa u cjelinu sustava, kako bi se specifično trebala ponašati ili kada se ponašanje značajke ostavi njemu/njoj na interpretaciju. Takve situacije bitrebalo pod svaku mjeru izbjegavati jer uloga programera u timu nije analizirati poslovne procese ili crtati dizajn formejer ga dizajner nije napravio, već kodirati i optimizirati tehnički aspekt aplikacije na temelju već raspisanih zahtjeva i specifikacija.

Primjeridijelova funkcionalne specifikacije vidljivi su na slikama 3 i 4. Na slici 3. se može vidjeti način primjene dijagrama toka u funkcionalnim specifikacijama, dok se na slici 4. može vidjeti primjer opisa funkcionalnosti kombinacijom slike i opisa koraka koji dovode do njenog ostvarenja na primjeru funkcionalnostiregistracijeu Alfredoaplikaciji.



Slika 3 Primjer grananja mogućih akcija u aplikaciji kao dio funkcionalne specifikacije[3]



#	Opis funkcionalnosti
1	Forma za registraciju korisnika na Alfredu izgleda kao na slici
2	Korisnik mora unijeti korisničko ime, lozinku, e-mail te željenu ulogu na aplikaciji (student ili poslodavac)
3	Ako korisnik ne unese neko od obaveznih polja, ispisuje se pogreška na ekranu u obliku teksta koji opisuje zašto se pogreška dogodila
4	Ako se unese korisničko ime koje ne postoji, ispisuje se poruka upozorenja
5	Ako se unese lozinka koja ne odgovara unesenom korisničkom imenu, ispisuje se poruka upozorenja
6	Ako već postoji korisnik s unesenim e-mailom, ispisuje se poruka upozorenja
7	Prilikom uspješne registracije, korisnik se automatski prijavljuje s unesenim podacima te preusmjerava na početnu stranicu

Slika 4 Primjer opisa funkcionalnosti kombinacijom slike i opisa koraka

3.2.3 Tehnička specifikacija

Tehnička specifikacija je dokument namijenjen tehničkom dijelu tima, odnosno programerima, administratorima baza podataka ili čak sistemašima, a njen cilj je opisati tehničke aspekte aplikacije kao što su struktura baze podataka (relacije između tablica),

[3] Preuzeto sa: https://www.majkic.net/novosti/nauka-i-tehnologija/813-funkcionalna-specifikacija-softvera-detaljno

arhitektura sustava, tehnologija (programski jezici) i radni okviri (*engl.Framework*), programi za razvoj, mrežne specifikacije i slično.

Ona je izrazito bitna programerima, a ne toliko ostalim članovima tima koji nisu direktno uključeni u tehnički aspekt oblikovanja aplikacije. Svrha njenog postojanja najviše dolazi do izražaja prilikom početka rada na nekomnovom projektu, kada je potrebno definirati tehnologije i alate koji će se koristiti za razvoj ili kada dođe novi programer u tim, pa ga treba upoznati s kodom u pozadini. Tada ta osoba može u detalje proučiti napisanu tehničku specifikaciju, te ako je ona koherentno i precizno napisana, lako shvatiti ideju iza koda koji čini aplikaciju. Tvrtke obično drže do nekih standarda i konvencija u pisanju koda, a postojanje tehničke specifikacije kao dokumenta koji opisuje tehničku stranu razvoja aplikacije tome još više doprinosi (Ein-Dor, Myers, Raman, 2004).

Primjer obrasca (formata) tehničke specifikacije vidljiv je na slici 5.

TECHNICAL DESIGN DIAGRAM
diagram should illustrate all required new and existing technical resources
COMPONENTS
CONNECTION & LOCATION *designate how new infrastructure will connect within existing infrastructure and will be located*
SERVERS & STORAGE & NETWORK *except in the case that specific tech issues are expected to require physical resources, assume all are virtual*
LOGICAL DATA FLOW

CRITICAL ANCILLARY EQUIPMENT & CONNECTIONS
DIAGRAM ATTACHMENTS & LINKS
list all printed attachment titles, file names, and / or links

TECHNICAL SPECIFICATIONS	
SERVERS	
APPLICATION SERVERS	
operating system	
application memory requirements	
application CPU requirements	
functional characteristics	
expected application transaction volume	
FILE SERVERS	
operating system	
application memory requirements	
application CPU requirements	

Slika 5 Dio obrasca za izradu tehničke specifikacije aplikacije [4]

3.2.4 Specifikacija održavanja aplikacije

Održavanje aplikacije možese gledati s nekoliko različitih aspekata. Aplikacija se može održavati na tehničkoj razini bez izmjene funkcionalnosti, brisanja postojećih ili dodavanja novih funkcionalnosti pa se tada radi o refaktoriranju koda (Gamma, 1995) ili se može održavati na način da se proširuju njene postojeće značajke dodavanjem novih, kroz dogovor s klijentom, ili doradom postojećih (Gamma, 1995). Neovisno o načinu održavanju, proces je

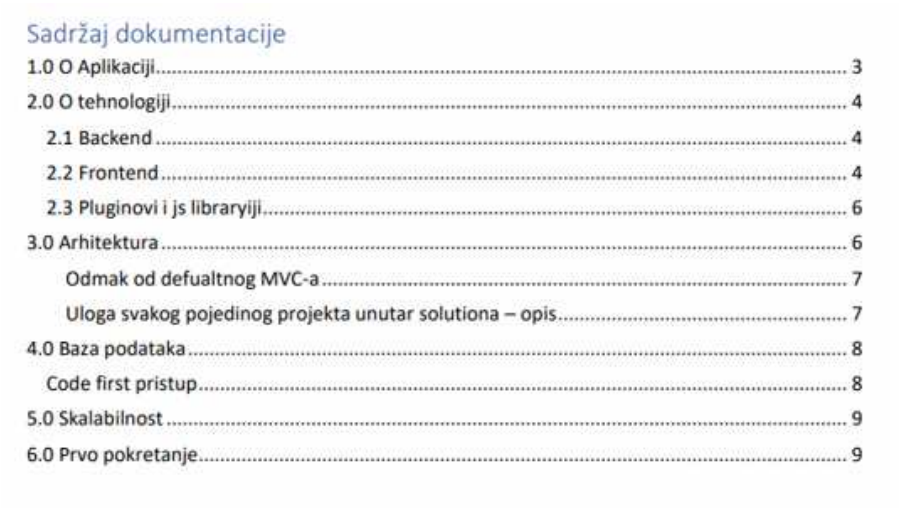
[4] Preuzetog sa: https://d2myx53yhj7u4b.cloudfront.net/sites/default/files/IC-IT-Technical-Specification-Template-PDF.pdf

potrebno dokumentirati. Specifikacije održavanja aplikacije opisuju kako točno održavati aplikaciju (Spolsky, 2000).Na primjer, sa tehničke strane onapo koracima opisuje proces dodavanja nove značajke u aplikaciju, pri tom pazeći da nova implementacija precizno prati i odgovara postojećoj arhitekturi koda.

Sa strane dizajna, pak, važna je dosljednost (Allen &Chudley, 2012). Bitno je da novo dodana značajka prati postojeći dizajn i da od njega ne odstupa(Allen &Chudley, 2012). Primjerice, ako korisnik želi dodati novo polje za unos teksta na formi jer mu se promijenio dio poslovnog procesa, onda će se obično najprije pogledati postojeća tema aplikacije i vidjeti kako ona rješava problem prikaza polja za unos teksta te će se potom isti stil primijeniti na novo dodano polje. Na primjer, ako se radi o web aplikacij, ne će se dodavati novaCSS klasa koja definira element polja za unos teksta kao element fiksne širine 150 piksela i visine 15 piksela te obruba debljine 2 piksela, ako postojeća tema koristi za širinu 80 piksela, visinu 10 piksela te nema obrub.

Dosljednost je bitna i za korisničko iskustvo (*engl. UX – userexperience*), budući da je ono jedno od ključnih parametara koji uvjetuju zadovoljstvo korisnika te njihovu voljnost daljnjeg i ponovnog korištenja aplikacije (Allen &Chudley, 2012).

Primjer izgleda djela specifikacije za održavanje aplikacije vidljiv je na slici 6.



Slika 6 Popis najbitnijih poglavlja specifikacije za održavanje Alfredo aplikacije

3.3 Dokumentacija

Osim izrada specifikacija, u timskom razvoju aplikacija izuzetno je bitno voditi i razne oblike dokumentacija (Gates, Myhrvold, Rinearson, 1995). Svrha dokumentacija je arhiviranje ključnih postojećih procesa, ideja, zamisli, izmjena, dorada ili značajki na projektu, s ciljem eventualno retroaktivne izmjene i primjene. Kako postoje funkcionalne ili tehničke specifikacije, tako postoje i dokumentacije (Gates, Myhrvold, Rinearson, 1995). Njihova svrha je dokumentirati najbitnijeizmjene, kako bi se kasnije lakše snalazili članovi tima koju sudjeluju u njenom razvoju. Kao primjer izrade dokumentacija, naveden je primjer izrade korisničkih uputstava u sljedećem poglavlju.

3.3.1 Korisničke upute

Korisničke upute ili priručnici su dokumenti namijenjeni krajnjim korisnicima aplikacije, a u kojima stoje upute za njeno korištenje (Smartics, 2018).Na primjer, neka IT tvrtka može razviti informacijski sustav za tvrtku na području energetikeu obliku web aplikacije, no zbog svoje kompleksnosti i velikog broja modula i podmodula, nijeintuitivna za korištenje korisnicima početnicima. U takvim situacijama se onda obično napišu korisničke upute. U uvodu se opiše osnovna namjena aplikacije, zatim njeni dijelovi te koraci za postupanje s pojedinim dijelovima. Na kraju se obično nalaze često postavljenja pitanja (*engl. FAQ-FrequentlyAskedQuestions*) koja korisnicima koji su tek počeli koristiti aplikaciju mogu biti od velike koristi (Smartics, 2018).

Osim korisničkih uputa, praksa je i voditi edukacijske radionice ili po potrebi držati prezentacije, što krajnjim korisnicima također može olakšati kasniju uporabu aplikacije(Smartics, 2018). Naravno, navedeno se više odnosi na poslovne aplikacije i informacijske sustave, nego na neke popularne, društvene, web 2.0 ili mobilne aplikacije, budući da se na njima takvi problemi uglavnom rješavaju izradom informativne sekcije ili mini tutoriala kojeg onda korisnici mogu proći ili pogledatida steknu znanje potrebno za njeno korištenje.

3.4 Dizajn

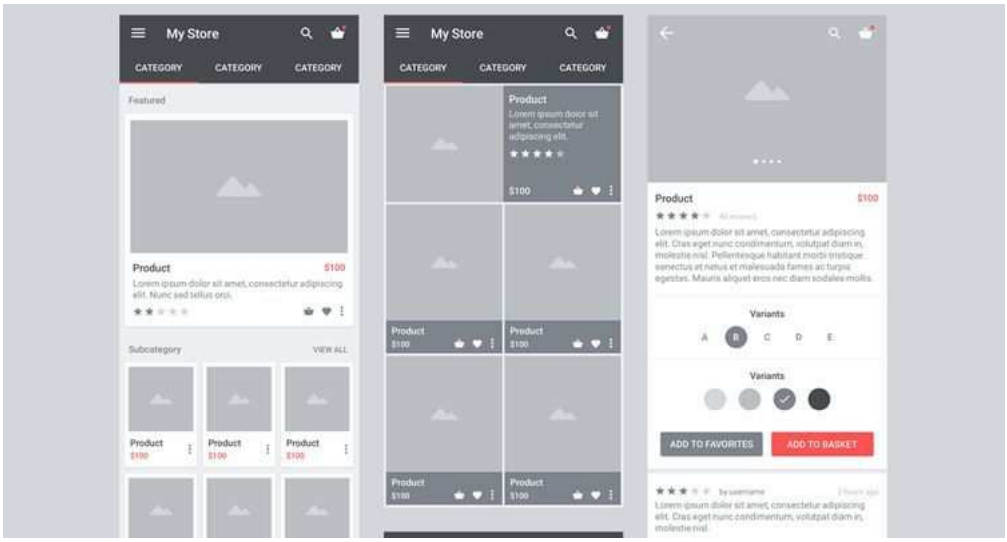
Dizajn se u kontekstu ovoga rada odnosi na dizajn u IT industriji. Govoreći o dizajnu u IT industriji, može se navesti mnoštvo različitih pozicija, od UX dizajnera do 3D modelera koji rade u industriji razvoja video igara. Upravo iz tog razlogaovdje nisu opisane sve moguće dizajnerske uloge u IT branši, već je generalno objašnjena uloga dizajna te nekih radnih pozicija u tom kontekstu.

Nakon što su klijent i voditelj projekta dogovorili detalje i napravili specifikaciju, slijedi dizajnersko prototipiranje i izrada žičanih okvira(*engl. wireframe*) aplikacije, na temelju kojeg će programeri izraditi konačan proizvod (Lin& Lee, 2007). Primjer žičanog okviraizrađenog radi boljeg shvaćanja izgleda zaslona, povezanosti funkcija i interaktivnosti između različitih pogleda mobilne aplikacijemože se vidjeti na slici 7.

Slika 7 Primjer izgleda žičanog okvira (wireframea) aplikacije [5]

U sferi razvoja web aplikacija, ovaj korak izgleda tako da obično UX dizajner napravi prototip aplikacije u nekom od alata za prototipiranje, a onda UI (*engl. userinterface*) dizajner napravi vizualni proizvod u nekom od alata za izradu slika (npr.Photoshop) kojeg će onda kasnije programeri (obično frontenddeveloperi) rezanjem pretvoriti u kod (*engl.slicing*). Primjer gotove teme u Photoshopu može se vidjeti na slici 8. U usporedbi s žičanim okvirom, očito je vidljiva veća razina detalja i izgled koji je više definiran i bliži krajnjem neko što je to na žičanom okviru.

[5]Preuzetosa: https://www.kogimobile.com/blog/5-things-to-consider-before-building-a-mobile-app/



Slika 8 Primjer teme izrađene u photoshopu[6]

U sferi razvoja video igara, ovaj korak izgleda tako da obično *concept artist* napravi koncept scene na temelju uputa dizajnera igre (*engl. game designer*), na temelju kojeg će umjetnik okoline (*engl.enviroment artist*) napraviti grafiku okoline, a umjetnik likova (*engl.character artist*) likove (Bethke, 2003). U slučaju 2D igreradit će se u alatima za izradu vektorske ili rasterske grafike koji su već spomenuti na početku rada, a u slučaju 3D igre u alatima za izradu 3D modela.Na slici 9. se može vidjeti konceptna slika lika za video igru. Na temelju ovog konceptaumjetnik likova će napraviti finalnog lika. Slično se radi i s ostalim artističkim elementima igre (okolinom, predmetima, itd.).

Slika 9 Koncept slika lika za video igru [7]

[6]Preuzetosa: https://medialoot.com
[7] Preuzeto sa: http://calader.blogspot.com

4 Metodologije razvoja aplikacija

Aplikaciju, osim dizajna, čini i programski kod. No, ne strateško kodiranje i forsiranje što bržeg zatvaranja projekta, i njegovog privođenja kraju, može rezultirati lošom kvalitetom aplikacije koju je kasnije potencijalno i teško za održavati (Hecksel, 2004). Pitanje koje se postavlja je kako organizirati razvojni tim tako da oni što efikasnije i bezbolnije razviju aplikaciju, sa što manje utrošenih resursa (vremena, novca, ljudstva) i što većom krajnjom kvalitetom proizvoda. Upravo tako je nastala potreba za uvođenjem metodologija razvoja u industriju razvoja aplikacija.Termin metodologija u kontekstu razvoja softvera prvi koristi Herbert D. Benington 1956.godine u svojoj knjizi o vodopadnom modelu *(engl.waterfall model*) razvoju softvera (Boehm, 1988).Beningtonov model vodopada na proces razvoja aplikacije gleda kao na niz koraka, gdje je ostvarenje svakog koraka u nizu nužan preduvjet za mogućnost ostvarenja sljedećeg (Boehm, 1988). Iz tog razloga je model vodopada problematičan i teško primjenjiv u modernim IT tvrtkama koje cijene brzinu i fleksibilnost u razvoju. Teško je očekivati da će moderna IT tvrtka čekati, na primjer, završenjekoraka izrade prototipa i dizajna, kako bi mogla krenuti na korak pisanja koda aplikacije jer se takvi koraci često mogu izvoditi paralelno.

Odabir odgovarajuće metodologije razvoja aplikacije ovisi o više stvari, a neke od njih su tip projekta (aplikacije) na kojoj se radi, kultura firme, preference članova tima i voditelja projekata i poslovna fleksibilnost (Burns&Dennis, 1985). U narednim potpoglavljima opisat će se neke od najkorištenijih metodologija u IT kompanijama današnjice. Prema Heckselu (2004), to su:

- ASD (engl.*Adaptive software development*)
- MSF (*engl.Microsoft solutionframework*)
- Kanban
- Lean
- DevOps
- DAD (*engl. Disciplinedagiledelivery*)
- DSDM(*engl.Dynamicsystems development method*)
- FDD (*engl. Feature-driven development*)

- IID (*engl. Iterativeandincremental development*)
- SCRUM
- XP(*engl.Extremeprogramming*)
- UP (*engl.UnifiedProcess*) te mnogi drugi.

U narednim poglavljimaopisani suDevOps, SCRUM, IID i FDD.

4.1 DevOps

DevOpsje praksa koja želi unificirati razvoj softvera i softverske operacije, tako je nastao i naziv, kombinacijom dijelova izraza tih dvaju pojmova na engleskom (Hecksel, 2004). Njegov cilj je automatizacija i nadzor svih koraka u procesu izrade softvera,od integracije, testiranja i postavljanja (*engl. deployment*) na produkcijsku okolinu (Hecksel, 2004). Cilj su mu češćapostavljanja na produkcijsku okolinu i upravljanje infrastrukturom. Kao što se može vidjeti i na dijagramu na slici 10., DevOps povezuje operacije, nadzor, planiranje,razvoj, testiranje, postavljanje na produkcijsku okolinu i osiguranje kvalitete u jednu cjelinu.

Slika 10 Dijelovi razvoja koje obuhvaća DevOps.[8]

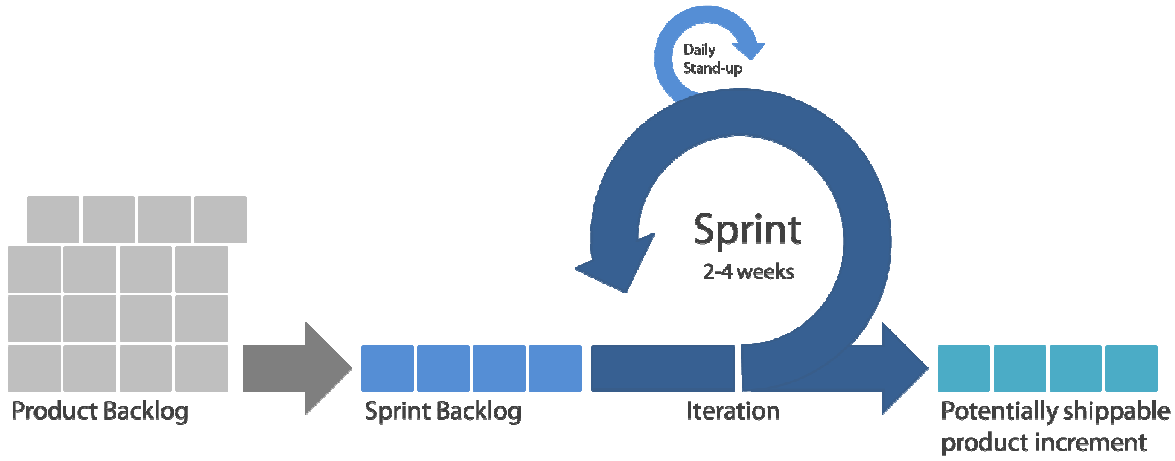
[8]Preuzetosa:https://www.agiletechnologist.com/2017/05/10/devops-introduction-executive-summary/

4.2 SCRUM

SCRUM je jedna od agilnih metoda razvojakojem je fokus na samom razvoju softvera (SCRUM, 2018).Radni dan tima koji radi po SCRUM metodologiji izgleda tako da se na početku radnog vremena, kada se skupi cijeli tim, odradi dnevni scrum (*engl. dailyscrum*), odnosno sastanak na kojem svaki član tima priča što je radio „jučer“, što će raditi „danas“ te ima li nekih problema ili poteškoća (Schwaber&Beedle, 2002). Sam sastanak vodi tzv. *scrummaster*, osoba koja uglavnom ima ulogu psihologa u timu, nakon čega se tim vraća na posao, svaki član za svoje računalo. Zadaci se rješavaju pomoću nekog SCRUM alata koji omogućava pregled zadataka koji su u određenim stanjima. Stanja mogu biti „obaveznonapraviti“ (*engl. commited*), „u tijeku“ (*engl. inprogress*), „testiranje“(*engl. testing*) ili „završeno“ (*engl. done*) (Schwaber&Beedle, 2002).Svrha postojanjastanjaje da se u bilo kojem trenutku može vidjeti na čemu točno neki član tima radi, kako bi se zadaci mogli lakše raspodijeliti. Zadaci se, naime, čitaju iz rezerve(*engl.backlog)*, odnosno iz ukupnog skupa zadataka koje treba riješiti, a koji su tamo dospjeli bilo radi korisničkih potreba, potreba za implementacijom novih značajki, ili radi pronađene greške (*engl. bug*) (Schwaber&Beedle, 2002).

Ono što je glavna značajka SCRUM-a su tzv. *sprintovi*, odnosno periodi od obično dva tjedna u kojima tim redom rješava zadatke iz rezerve kako su se dogovorili na planiranju (*engl. planing*) (SCRUM, 2018). Planiranjeje sastanak koji se održava na početku novog sprinta, obično prvi dan novog sprinta, a na kojem članovi tima skupa prolaze kroz korisničke zahtjeve i procjenjuju vremena potrebna za njihovu implementaciju, ako se radi o proizvodnoj jedinici iz rezerve (*engl. PBI-productbacklogitem*), ili sređivanje, ako se radi o bugu. Proizvodna jedinica iz rezerve je naziv za značajku koju treba implementirati, a na sebi ima opis značajke, odnosno što treba implementirati, poveznicu na specifikaciju, procjenu sati te eventualne napomene i konkretno ime i prezime člana tima koji ga mora riješiti prije završetka sprinta za koji je važeći (Schwaber&Beedle, 2002). Tim tako, tijekom sprinta, rješava proizvodne jedinice iz rezerve i bugove te po potrebi radi dorade dogovorene na sastancima koji se zovu*grooming* sastanci, a na kraju sprinta (ili prvi dan sljedećeg) se radi retrospektiva. Retrospektiva je dakle sastanak koji se održava svakih dva tjedna kao i planiranje, a za zadatak ima potaknuti članove tima na osvrt na završeni sprint i što su sve napravili (Schwaber&Beedle, 2002).

Prema možda kompleksan za opisati pismeno, SCRUM je itekako efikasan u praksi, pogotovo kada se tim međusobno upozna i nauči raditi zajedno. Najbitniji koraci su i vizualno ilustrirani na slici 11.



Slika 11 Opći koraci rada po SCRUM metodologiji razvoja aplikacija[9]

4.3 IID

IID je kombinacija iterativnog dizajna ili metoda te inkrementalnog modela razvoja softvera (Mitchell &Seaman, 2009). Nastao je kao odgovor na neefikasni i problematični model vodopada.

Osnovna ideja iza IID-a je razvoj sustava kroz iteracije i u manjim vremenskim razmacima (inkrementima), ostavljajući razvojnim programerima i stručnjacima dovoljno vremena za uvid u prošle greške, ali i dobre prakse, prakse koje će nastaviti raditi u sljedećim iteracijama (Wirts, Gary&Lazarus, 2006). Zapravo, može se reći kako je IID i dio prethodno opisanog SCRUMA, gdje su iteracije i kratki vremenski periodi zapravo sprintovi, a u svakoj iteraciji (sprintu) se dodaje nova značajka u aplikaciju te se tako ona konstantno unaprjeđuje.

Vizualni prikaz iterativnog modela vidljiv je na slici 12. Prvo se planira i analizira vrijeme potrebno za rješavanje nekog korisničkog zahtjeva, pokušava se doći do implementacije na temelju postojećeg dizajna, potom slijedi razvoj pa postavljanje na produkcijsku okolinu (*deploy*), ako je sve u redu.

[9]Preuzetosa:https://slidemodel.com/templates/software-diagrams-powerpoint/scrum-agile-methodology-high-level-diagram/

Slika 12 Model iterativnog razvoja u procesu razvoja aplikacija[10]

4.4 FDD

FDD (*engl. Feature-driven development*) je iterativan i inkrementalan proces razvoja softvera, odnosno metodologija, te je, baš kao i SCRUM, primjer agilne metode razvoja aplikacija (Palmer&Felsing, 2002). Temelji se na značajkama, zbog čega i je agilan te olakšava rapidan razvoj eliminirajući svaki poslovni proces koji ne doprinosi razvoju iste (Palmer&Felsing, 2002).

Za ovu metodologiju su karakteristični još i tzv. *milestonovi* odnosno točke razvoja u kojima je ostvaren neki kritični skup izmjena ili je napravljen dovoljan broj zamišljenih implementacija značajki (Palmer&Felsing, 2002).

[10] Preuzeto sa: http://www.topsinfosolutions.com/methodology/

5 Tehnički pristup razvoju aplikacija

U ovom poglavlju bit će riječi o najvažnijim principima u razvoju softvera, oblikovnim obrascima, sustavima za verzioniranje koda te raznim arhitekturama koje se koriste na stvarnim, modernim informacijskim sustavima današnjice.Pojmovi, principi i načela opisana u ovom potpoglavlju nužan su preduvjet za razumijevanje tehničkog opisa Alfredoaplikacije u 8. poglavlju rada.

5.1 Principi

Kako programeri ne bi pisali loš kod koji treba iznova nanovo pisati i u kojem se nemogu uopće snaći, postoje tzv. principi kojih se mogu držati, a koje je smislila zajednica programera kroz dugogodišnjupraksu od 80-ih godina do danas (MacLennan, 1987). Ti principi omogućavaju bezbolnije programiranje i programiranje u kojem programeri mogu biti više usmjereni na implementaciju značajke, nego njenu tehničku pozadinu (MacLennan, 1987).

Najčešće korišteni principi u razvoju aplikacija navedeni su i opisani u prilogu. Neki od tih principa korišteni su i prilikom izrade Alfredo aplikacije.

5.1.1 SOLID

SOLID je akronim u kojem svatko slovo predstavlja početno slovo naziva principa kojeg bi se trebalo držati prilikom razvoja bilo koje aplikacije (MacLennan, 1987).

Značenje pojedinih slova, prema MacLennan(1987):

S-*Single responsibilityprinciple*

Klasa bi trebala raditi jednu i samu jednu stvar. Na primjer,klasa koja se zove *EnemyMovement.cs* treba se baviti isključivo logikom kretnje neprijatelja,a ne i instanciranjem objekata po mapi igre.

O - *Open/closedprinciple*

Ovaj dio SOLID principa kaže kako bi entiteti trebali biti otvoreni za ekstenziju (proširenje),ali zatvoreni za modifikaciju. To znači da se sustav može lako proširiti jer je izgrađen slijedeći ovaj princip, ali i da onemogućuje pretjerane modifikacije postojeće implementacije.

L-*Liskovsubstitutionprinciple*

Liskovo pravilo zamjene kaže kako objekti u programu moraju biti zamjenjivi s instancama njihovih pod tipova bez mijenjanja ispravnosti programa. Klasa kontrola (*UIkontrole.cs*) za korisničko sučelje može se učiniti javnom apstraktnom tako da ju svaka pojedinačna klasa neke specifične kontrole (gumba, labele, itd) može nasljediti. Budući da su pojedinačne kontrole također klase za sebe (na primjer klasa *Gumb.cs* za gumb kontrolu na stranici ili klasa *Labela.cs* za labele na stranici), onda one sve mogu nasljediti tu generičkuapstraktnu klasu kontrole (*UIkontrole.cs*). Na taj način, objekt tipa *UIkontrole*se može dobiti i instanciranjem objekta klase *Gumb,* pozivom konstruktora *Gumb* klase ili instanciranjem objekta klase *Labela,* pozivom konstruktora *Labela* klase.

I- *Interfacesegregationprinciple*

Ovaj princip kaže kako je više manjih, specifičnih sučelja (*engl. interface*) bolje nego jedno generičko veliko.Na primjer, u Alfredo aplikaciji postoji sučelje za generički repozitorij,kojeg implementira klasa generičkog repozitorija, a onda svi ostali specifični repozitoriji u projektu,na primjer repozitorij za dohvat poslodavaca, poslova, poruka i slični, nasljeđuju taj generički repozitorij, ali i za njih specifičan repozitorij. Na primjer, klasa repozitorija poslovaimplementira,osim generičkog repozitorija, i *IJobRepository*sučelje koje je za tu klasu specifično jer sadrži za nju specifične metode dohvata podataka, dok klasa repozitorija poruka, osim generičkog repozitorija, implementira i specifično*IMessageRepository* sučelje.

D - *Dependencyinversionprinciple*

Ovaj princip kaže kako bi se klase trebale oslanjati više na apstrakcije nego na konkretne implementacije, tj. princip omogućava tzv. *decoupling*, gdje moduli s višeg sloja ne ovise o modulima nižeg sloja, što olakšava razvoj i povećava modularnost i fleksibilnost projekta. U slučaju Alfredo aplikacije, rješenje (*engl. solution*) je podijeljeno po projektima, kako bi se pratilo ovo načelo te dobila što manju međuovisnost u kodu. Primjerice, jedan od načina na koji je ostvarena manja međuovisnost između klasa jest uporabom tzv.ubrzigavanja ovisnosti (*engl. dependencyinjection)*u web projektu rješenja.Ubrizgavanje ovisnosti forsira i moderni.NETCore budući da nas liši repetitivnog pisanja poziva konstruktora za instanciranje objekta nekog konkretnog tipa svugdje po kodu. To je moguće jerinstancu konkretnog tipa prosljeđuje konstruktoru klase koja ga treba prilikom pokretanja aplikacija (na *startupu*), putem primjene sučelja.

5.1.2 DRY

DRY (engl. *Don'trepeatyourself*) se odnosi na pisanje koda koji je ponovno upotrebljiv i to ne samo na nekom drugom mjestu u postojećem projektu, već primjenjivi na drugim projektima (Stroustrup, 2008). Na primjer korisno je sačuvati kod za ajax poziv metode kontrolera tako da ga drugi put možemo samo kopirati i prilagoditi parametre,a ne ponovno pisati ispočetka ili prisjećati se sintakse.

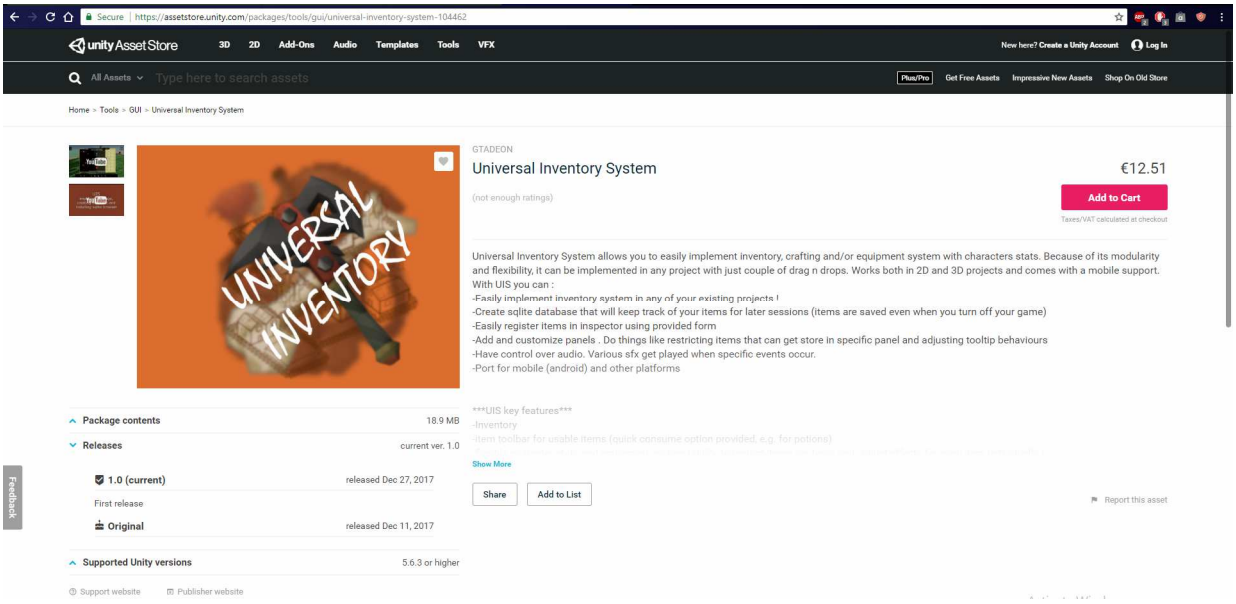
5.1.3 KISS

KISS(engl. *Keepitsimpleandstupid*) je princip koji predlaže što jednostavnije implementacije rješenja umjesto onih kompleksnih. Katkadprogramerima, prilikom kodiranja, na um počnu padati i neka moguća, hipotetska, buduća, proširenja sustava pa počnu kodirati postojeće stvari tako da budu otvorena za proširenja za te buduće izmjene koje možda nikad neće doći. Drugi primjer jest komplicirano rješenje za neki banalni problem. Primjer apsurda vidljiv je na kodnome bloku 1 u prilogu.U tom primjeru, mogla se maknuti linija koja poziva funkciju koja vraća *bolean* i ostaviti samo *ifelse* provjera koja onda vraća *bolean* rezultat (vrijednost varijable *true* ili *false*)

5.1.4 Princip paketa

Princip paketa kaže kako se klase koje doprinose rješavanju problema unutar istog konteksta trebaju organizirati u skupine – pakete, pa se onda kao takve mogu koristiti i drugdje (Stroustrup, 2008).

Na primjeru inventara igre, moguće je objediniti sve skripte pod jedan krovni inventarski namespace te ih kao takve izvesti u obliku paketa koji se onda može koristiti i u drugim projektima razvoja video igara, na primjer u Unity pogonskom sklopu igre (*engl. game engine*). Primjer takvog paketa vidljiv je na slici 13.



Slika 13 UniversalInventory System inventarski paket za Unity pogonski sklop igre

5.1.5 GRASP

Ovaj princip govori o dodjeli odgovornosti klasama unutar projekta. Nekoliko je oblikovnih obrazaca povezano s njim, primjerice *controller* oblikovni obrazac koji preporuča zasebnu klasu za nošenje s pozivima koji nemaju veze s korisničkim sučeljem (*engl. userinterface*). Primjer arhitekture koja slijedi ovaj princip je MVC (*engl. Model ViewController*) arhitektura, gdje se točno zna koji sloj ima koju funkciju, tj. *Model*je obična klasa koja sadrži entitete, pogled (engl. *views*) je frontend dio,dok je *kontroler* dio na backendu koji predstavlja most između modela i pogleda. U Alfredo aplikaciji, standardna MVC arhitektura nije bila dovoljna za potrebe razvoja projekta pa je ona znatno proširena i dorađena novom.

5.2 Oblikovni obrasci

Imajući na umu prethodno spomenute principe, mnogi programeri su tijekom godina otkrili razne obrasce koji se ponavljaju po raznovrsnim projektima, odnosno shvatili su kako se neki problemi iznova i iznova javljaju (Gamma, 1995). Prolaskom godina, rješenja za neke od tih čestih problema su postala dijelom ustaljene programerske prakse. Ugrubo, govori se o tzv. oblikovnim obrascima (*engl. design patterns*).

Oblikovni obrasci omogućuju brzo i sigurno rješenje nekog problemauporabom koda koji je već dobro poznat zajednici, a za koji se zna da efikasno rješava konkretan problem (Gamma,1995).

Oblikovnih obrazaca ima mnogopa su u ovome radu opisani samo neki. Jedna od mogućih podjela može se vidjeti u tablici 2 (Gamma, Helm, Johnson, Vlissides, 1994).

Vrsta oblikovnog obrasca	**Primjer obrasca**
Obrasci stvaranja	Apstraktna tvornica (engl. *abstractfactory*) Builder Dependencyinjection Lijena inicijalizacija (*engl*. *lazyinitialization*) Bazen objekata (*engl*. *objectpool*) Singleton
Strukturalni	Adapter Most (engl. *bridge*) Dekorater (*engl*. *decorator*) Kompozit (*engl*. *composite*) Proxy
Bihevioralni	Command Iterator Promatrač (*engl*. *observer*)

Tablica 2. Vrste i primjeri oblikovnih obrazaca.

Primjerice, bazen objekata se može koristiti u području razvoja igara za instanciranje municije igrača na samom početku igre, kada se ona tek učitava, kako bi spriječio pad performansi njenom naknadnom odgodom. Promatrač semože koristitiza lakše lovljenje događaja po kodu, dok se apstraktna tvornica koristi za kreiranje objekata jer smanjuje međuovisnost u kodu naspram tradicionalnog načina kreiranja objekata putem običnog poziva konstruktora klase konkretnog tipa bilo gdje u kodu (Gamma,1995).

5.3 Arhitektura koda

Arhitektura koda je pojam koji se odnosi na način na koji je kod unutar unutarrješenja (*engl. solution*) uređen, na slojeve i njihove odnose, klase i njihove odnose i odgovornosti te na načine dohvata podataka i njihovog protoka (Shaw &Garlanm, 1996).

Dobro postavljena arhitektura nužan jeuvjet za uspješan razvoj aplikacije, budući da ona omogućava lako dodavanje novih funkcionalnosti, odnosno omogućuje laku proširivost sustava i njegovu nadogradnju.

Arhitektura koda projekta također ovisi i o tipu projekta koji se razvija. Za neke projekte je prikladnije implementirati kompleksne arhitekture kao što je arhitektura temeljena na oblikovanju vođenom domenom (*engl. DDD-Domaindriven design*), dok je za druge dovoljna MVC arhitektura, na primjer u slučaju izrade jednostavnijih web aplikacija(Shaw &Garlanm, 1996).

U narednim potpoglavljima opisane su arhitekture koje se koriste na stvarnim aplikacijama s ciljem njihova boljeg razumijevanja.

5.4 Domaindriven design

Domenom vođeno oblikovanje (*DDD)* je arhitektura koju je prvi osmislio i svijetu predstavio Eric Evans u svojoj istoimenoj knjizi 2004. godine.

Ideja koju je Evans imao je ta da bi u centru sustava trebala biti domenska logika i modeli te da bi sav dizajn sustava trebao bazirati upravo na njima(Evans, 2004).

Oblikovanje vođeno domenom najviše se isplati primijeniti u velikim informacijskim sustavima, sustavima koji imaju veliku poslovnu logiku u pozadini te za koje je predviđeno aktivno održavanje narednih godina(Danenas, Garsva, 2012). Preciznom i odgovarajućom primjenom DDD arhitekture na takvim projektima, programeri su rješenibarem djela muke prilikom implementacije novih značajki.

DDD arhitektura obuhvaća nekoliko bitnih koncepata, a to su kontekst, domena, model i tzv. „sveprisutni“ jezik (*engl. ubiquitouslanguage*) (Evans, 2004).

Kontekst je okolina u kojoj neka riječ ili izjava nastaje, a koja određuje njeno značenje (Evans 2004). Govoreći o DDD arhitekturi i aplikacijama, kontekst ovisi o njenoj domenskoj namjeni (Evans, 2004). Primjerice kontekst može biti zdravstvo, ali također može biti i državni sustav vodoopskrbe.

Domena je sfera ili niša znanja sustava, odnosno područje unutar kojeg se rješava konkretan problem(Evans, 2004). Na primjer, u slučaju aplikacije koja prati pacijente javne zdravstvene skrbi neke države, a koja se temelji na DDD arhitekturi, domena može bitizdravstveni sustav te državei sve administrativne ili zdravstveno zakonodavne poddomene koji on obuhvaća.

Model je apstrakcija domene (Evans, 2004), a obično se koristi za rješavanje problema povezanih s domenom,primjerice u slučaju s javnim zdravstvom, jedan od modela je model bolnica, drugi je ambulante, treći hitna pomoć, a četvrti privatne klinike.

Zadnji pojam, sveprisutni jezik, se odnosi na jezik strukturiran oko domenskog modela,a koji koriste svi članovi tima kako bi uspješno povezali sve aktivnosti tima sa samim softverom (Evans, 2004). Često se u praksi događa da klijenti, odnosno stručnjaci iz konkretne domene, neki entitet zovu jednim službenim nazivom, dok se u kodu taj isti entitet zove nekako drugačije. Na timu je da odredi nazivlje koje će se koristiti u kodu te da ono ima jasnu referencu po specifikacijama i dokumentacijama, odnosno da ono reflektira neko stvarno stanje unutar domene za koju se radi aplikacija jer se time smanjuje dvosmislenost, odnosno ubrzavai omogućavaefikasan razvoj.

Alfredo web aplikacija se također temelji na DDD arhitekturi.U narednim poglavljima se može vidjeti kako izgleda njena konkretna implementacija na projektu koji sezasniva na C# programskomjeziku i .NET core radnom okviru.

5.5 Test drivendevelopment

TDD praksa zasniva se na pisanju testova koje napisane metode moraju proći (Beck, 2003). Naime, ideja je daprogamer na umu prvo ima funkcionalnost koju mora ostvariti, potom smisli metode koje ju ostvaraju, pa testove za njih.Jednom kad je smislio i testove i metode, prvo treba krenuti pisati testove za metode, a onda tek konkretne implementacije metoda koje te testove moraju proći.

Ovakav razvoj aplikacija može oduzeti puno vremena, zbog čega ga ne primijenjuje punorazvojnih timova. Prednostje ta što tim pristupom postoji velika sigurnost da aplikacija zbilja radi očekivano i predviđeno jer se testovi mogu lako pokrenuti u bilo kojem trenutku.

5.6 Acceptance test-driven development

Slično kao i TDD, ATTD se zasniva na testovima, ali onim koji su dogovoreni na temelju komunikacije između klijenta, testera i programera (Riley&Goucher, 2009). Često je takav kod oblika „*givenwhenthen*“ .

Na primjer :

- *Given*: „Aplikacija bez prijavljenog korisnika i prikazom početne stranice“
- And :„Korisnik se nalazi na stranici, ali nije prijavljen „
- *When :*„Korisnik unese svoje korisničko ime i lozinku te klikne na gumb prijava“

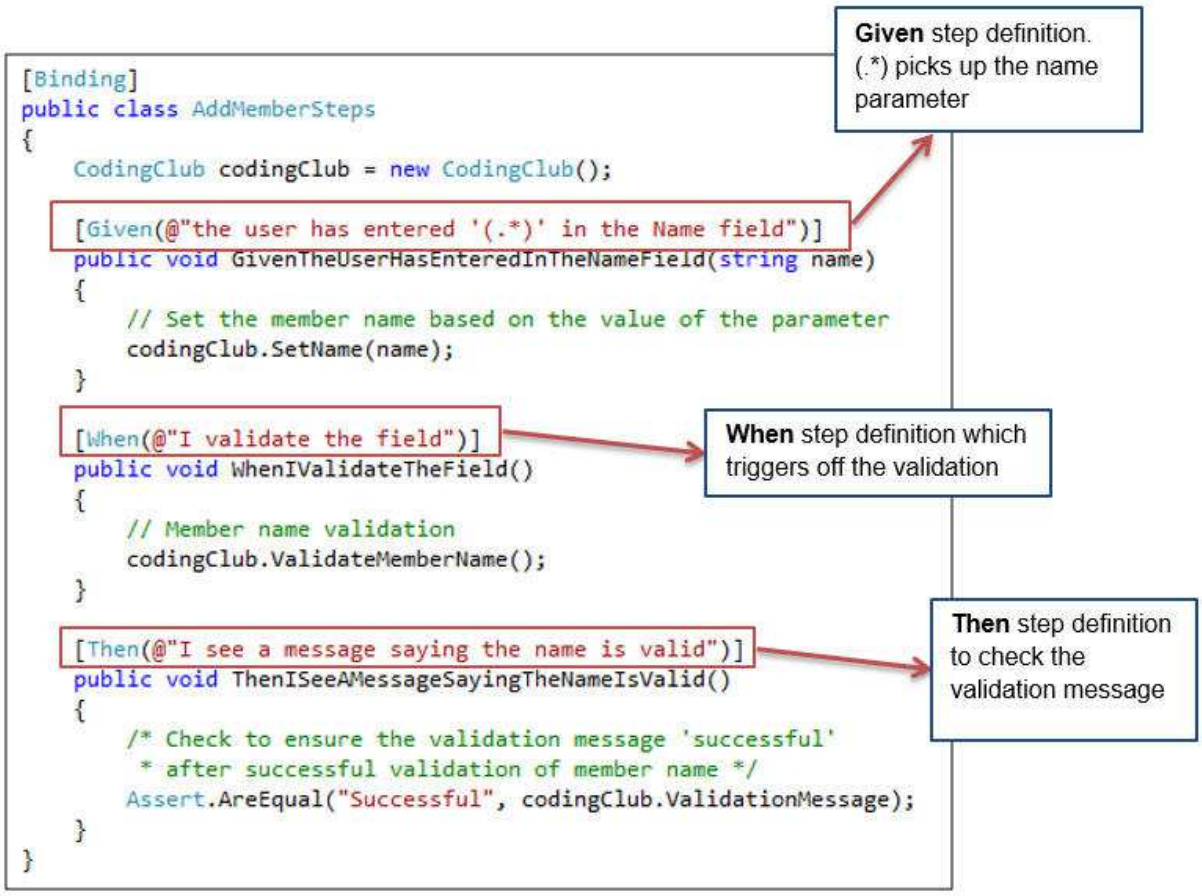
- *Then*: „Korisnik će biti prijavljen u aplikaciju i preusmjeren na stranicu svog profila, ako je uneseno ispravno korisničko ime i lozinka, inače će se ispisati poruka o neuspjeloj prijavi i ostat će na stranici na kojoj je i bio“

5.7 Behaviourdriven development

BDD je nastao iz oblikovanja vođenog testiranjem (TDD), tako da kombinira ideje iz TDD-a i dio ideja iz DDD-a (Wynne, Hellesoy, Tooke 2017). Ideja je definirati gdje krenuti u procesu razvoja, što testirati, što ne testirati, kako nazvati test i kako razumjeti zašto je neki test pao (Saff& Ernst, 2004). U središtu BDD-a su unit testovi i *acceptance* testovi koji su nastali zbog potreba testiranja funkcionalnosti značajke koja rješava neki domenski problem(Wynne, Hellesoy, Tooke 2017).

Kod BDD-arazvojni stručnjak zapravo mora prvo definirati set testova za neku jedinicu pa napraviti pad testa, potom implementirati jedinicu te finalno potvrditi da implementacija jedinice prolazi napisani test (Wynne, Hellesoy, Tooke 2017).

Primjer konkretne implementacije u C# programskom jeziku putem uporabe tzv. „*givenwhenthen*“ testova vidljiv je naslici 14. U tom primjeru sadržaj metode će se izvršiti kada program pročita scenarij zapisan u obliku atributa iznad nje, jednom kada se test pokrene.



Slika 14 Givenwhenthen test implementiran u programskom jeziku C# [11]

5.8 Sustavi za verzioniranje koda

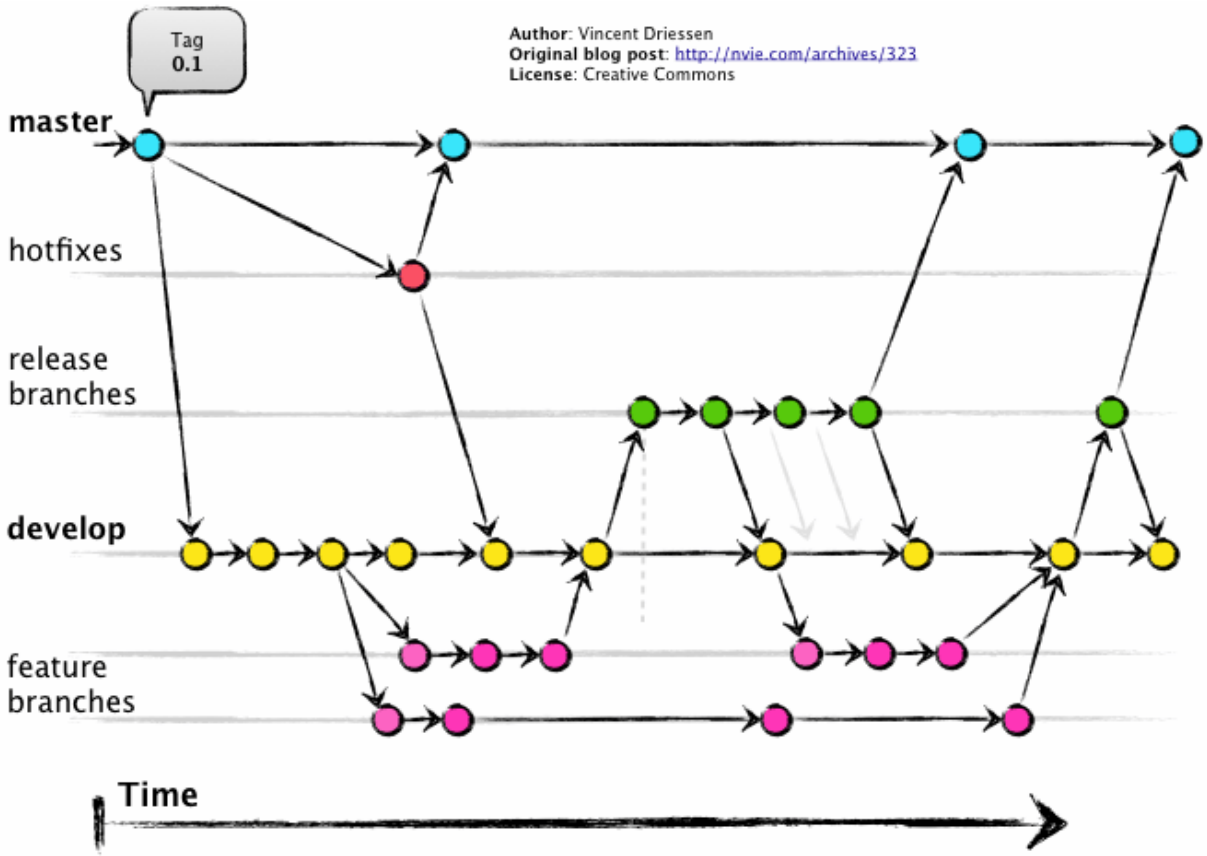
Jednom kada je sav kod potreban za neku značajku napisan, te eventualno prošao *codereview* ostalih članova tima, šalje se na neki zajednički repozitorij koda. To centralno mjesto čuvanja koda temelji se na sustavu za verzioniranje koda.Verzioniranje izvornog koda je postupak u kojem se izvorni kod pohranjuje u središnju bazu, pri čemu se bilježi verzija izmjene kako bi se one mogle pratiti kroz povijest (Sussman,Fitzpatrick, Pilato2008).

Kvalitetan timski razvoj aplikacije bez ovog tehničkog aspekta je gotovo nemoguć. Naime, zahvaljujući verzioniranju koda, može se u bilo kojem trenutku vidjeti tko je napravio koju izmjenu na nekoj klasi,vratiti na prethodnu verzija, ukoliko trenutna ne radi, tj. napraviti tzv. *rollback*, imati kontrolu nad trenutnom verzijom aplikacije te odlučivati što ulazu u finalnu verziju aplikacije putem metode *pullrequestova* i *codereviewova*. Namjena codereviewa je ocijeniti kvalitetu koda koji se želi poslati na sustav za verzioniranje. Praksa je obično da

[11]Preuzetosa: https://www.red-gate.com/simple-talk/dotnet/net-development/behaviour-driven-development-part-2-implementing-scenarios-and-step-definitions-in-specflow/

seniori i *mid*developeri ocjenjuju kod *junior*developera ostavljanjem komentara na linije koda.

Cijeli postupak verzioniranja može se zamisliti kaograna (glavna verzija), koja može i nemora imati manje podgrane (podverzije),a čvorovi na njoj predstavljaju verzije koje su učitane na server (slika 15.)



Slika 15 Shematski prikaz promjena izvornog koda u Git sustavu[12]

[12] Preuzeto sa:https://numergent.com/2016-12/Branching-workflow-git-flow-and-github-flow.html

6 Testiranje

Testiranje je korak koji nikako ne smije biti zanemaren u softverskoj industriji jer jednako kao i tehnički razvoj ili prototipiranje i dizajn, direktno utječe na kvalitetu proizvoda. Taj korak osigurava da softver ili dio softvera radi ono što mora raditi (Saff& Ernst, 2004). Njimese pronalaze eventualne greške u kodu s ciljem njihova ispravljanja prije nego što postanu dio produkcijske verzije aplikacije (Saff& Ernst, 2004).

Testiranje se provodi po više različitih okolina na kojima je aplikacija namještena. Na primjer, u kontekstu razvoja web aplikacija može se testirati lokalno (npr. unutar VisualStudia, prilikom razvoja), na QA (*engl. qualityassurance*) okolini, UAT (*useracceptence test*) okolini i na produkcijskoj verziji (testovi koje rade krajnji korisnici na aplikaciji).

Također, neovisno o tipu aplikacije koja se razvija, postoji više različitih vrsta testova, poput unit testova, testovaizdržljivosti, penetracijskih testova ili funkcionalnih regresijskih UI testova kao primjera automatskih testova (Riley&Goucher, 2009).

6.1 Unit testovi

Unit testove pišu programeri i njihova primarna namjena je osigurati ispravnu funkcionalnost i rad neke najmanje jedinice sustava, poput metode (Saff& Ernst, 2004). Unit testove je, gledano sa strane arhitekture koda i čitljivosti, dobro odvojitiu novi,zasebni, projekt, kako bi imali lakši pregled i kontrolu nad njima.

Jedan od unit testova u aplikaciji namjenjenoj za knjiženja je na primjer provjera uspješnog dohvata knjiženja iz baze prema nekom parametru. Primjer unit testa u pucačkoj igri jest test koji osigurava da se neprijatelj stvara na nekom mjestu u 3D prostoru kada je uvjet ispunjen.

Primjer unit testa napisanog u C# programskom jeziku u VisualStudiu vidljiv je u kodnom bloku 2. u prilogu.U tom kodnom bloku se provjerava jeli prvi parametar („10“) jednak rezultate funkcije *sum* (drugi parametar) koja prima dva cijela broja.U slučaju u kodnom bloku testće proći jer je prvi parametar *AreEqualAssert* metode (*„10“*) uistinu jednak drugom (*„5+5“*). Da je *sum* funkcija vratila neki drugi rezultat, unit test ne bi prošao.

6.2 Testovi izdržljivosti

Testovi izdržljivosti uglavnom se provode naserverima, kada je potrebno provjeriti da server može podnijeti istovremenu aktivnost velikog broja korisnika (Saff& Ernst, 2004). Primjer bi bio Studomat i njegov rad pod „pritiskom“ kada svi studenti odjednom krenu upisivati neke kolegije na početku semestra. Cilj testa izdržljivosti je simulirati velik broj istovremenih prijava korisnika te njihovih akcija na aplikaciji, kako bi se vidjele performanse aplikacije i ustvrdile eventualne mane sustava.

6.3 Penetracijski testovi

Penetracijski testovi se provode iz sigurnosnih razloga, kako bi pronašli eventualne sigurnosne nedostatke u sustavu, s jedne strane, te kako bi bili sigurni da je sigurnost sustava na željenoj razini, s druge (Arkin, Stender, McGraw, 2005).

Uobičajeni penetracijski testovi uključuju (Infigo, 2018):

- Umetanje SQL koda (*engl. SQL injection*), odnosno postupak u kojem se umeće dio koda napisan SQL sintaksom najčešće u polje neke forme, poljekoje inače predviđa neku drugu vrstu unosa, npr. ime i prezime, a sve s ciljem dobivanja nekih drugih podataka iz baze ili pak uništavanja baze (Halfond, Viegas, Orso, 2006).

- XSS ranjivosti (*engl. cross site scripting*) odnosi se na ubacivanje neke skripte ili dijela koda koji onda zlonamjerno djeluje na stranicu, na primjertako da prikuplja podatke dostupne u pregledniku (Acunetix, 2018)

- Cross Site RequestForgery je oblik napada koji prisiljava krajnjeg korisnika na izvršavanja neželjenih akcija na web aplikaciji na kojoj je on trenutno autenticiran. Na primjer, ponovni unos email adrese (Barth, Jackson, Mitchell, 2008)

- Rukovanje pogreškama

6.4 Funkcionalni regresijski UI testovi

Zadatak funkcionalnih regresijskih UI testova je osigurati da dio aplikacije, obično neka značajka, radi ono što je propisano u specifikaciji, odnosno što korisnik očekuje da radi (evolva, 2018). Zadatak automatskih testova nije otkriti nove greške, nego služiti kao osiguranje koje garantira rad ključnih funkcionalnosti koje su trenutno implementirane u aplikaciji.

U Java, .NET (C#) i Python projektima moguć je koristiti *Selenium* radni okvir za izradu takvih testova (SeleniumHQ, 2018). Selenium omogućava pisanje automatskih testova koji će se onda izvršiti u nekom pregledniku, primjerice Google Chromeu (SeleniumHQ, 2018).Kodira se ponašanje koje bot mora izvesti kao i očekivani rezultat testa. Nakon pokretanja testa i njegovog izvršavanja u pregledniku, test ili prolazi ili pada, ovisno o tome jel dobiveni rezultat odgovara očekivanom ili ne.

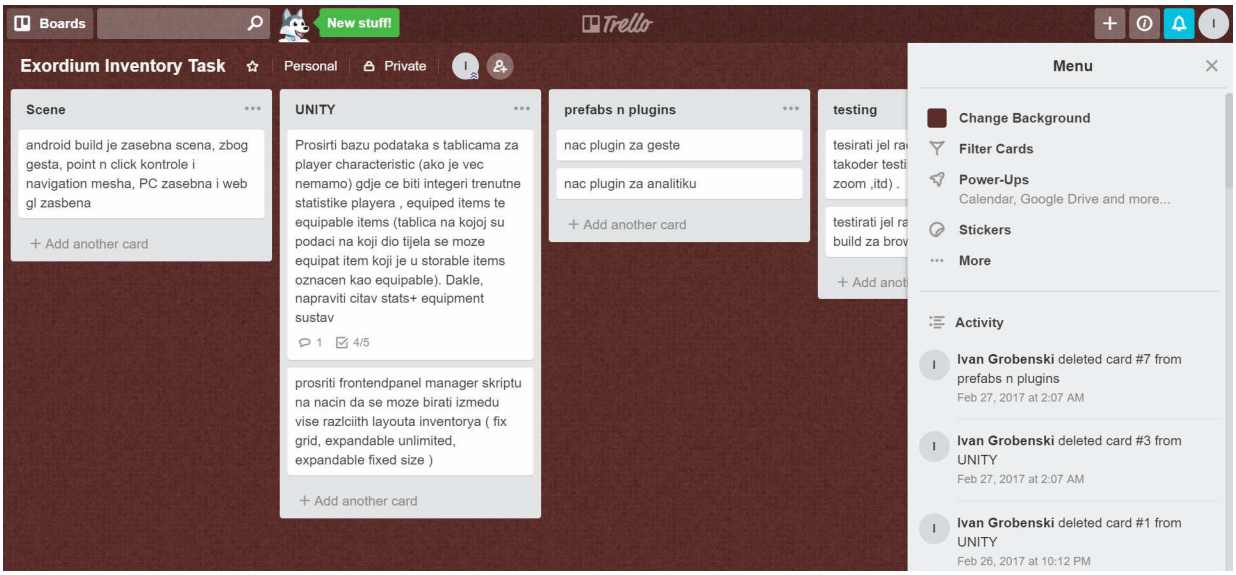
7 Alati za lakši razvoj u timovima

U ovom poglavlju navedeni su neki od alata koji su se pokazali korisnim za timski razvoj aplikacije.

7.1 Trello

Trello je web aplikacija s besplatnom i plaćenom verzijom, a omogućavaotvaranje i uređivanje karticakoje su onda pogodne za praćenje zadataka razvojnih timova (Trello, 2018).

Primjer izgleda zadataka napisanih na karticama za potrebe razvoja inventara vidljiv je na slici 16.



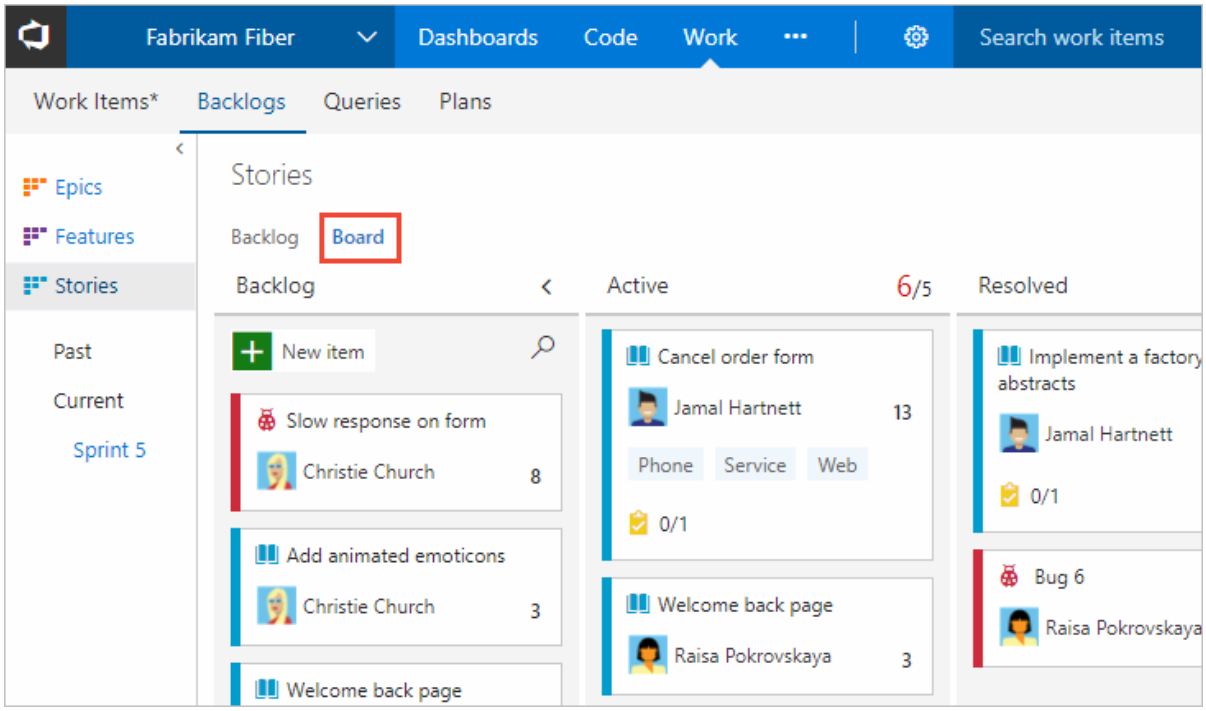
Slika 16 Trello i njegov princip rada s karticama.

7.2 TFS

TFS *(engl. Team Foundation Server)*je Microsoftov proizvod za rad u timovima, a koji omogućava upravljanje izvornim kodom, izradu izvještaja, upravljanje projektom, automatske buildove verzija, testiranje, jednostavno upravljanje verzijama aplikacije koje trebaju postati dio produkcijske, upravljanje životnim vijekom aplikacije, pisanje zadataka po nekoj od najkorištenijih metodologija razvoja softvera poput SCRUM-ai još mnogo toga (VisualStudio, 2018).

U firmi u kojoj radim za vrijeme pisanja ovog rada koristimo TFS pa iz osobnog iskustva mogu reći da uvelike doprinosi i olakšava implementaciju temeljnih ideja i paradigmi SCRUM-a kao metodologije. TFS omogućava brzu izradu proizvodnih jedinica iz rezerve (tzv. PBI-eva) i njihovo spremanje u rezervu, kontrolu nad svim potrebnim kolonama (*napraviti, u tijeku, testiranje, završeno, pronađene greške* i slične), uvid u trenutno stanje sprinta i postotak riješenih i ne riješenih zadataka, broj utrošenih sati, zadatke na kojima radi bilo koji član tima te brzo raspisivanje pronađenihgrešakapo potrebi.

Izgled sučelja TFS-a vidljiv je na slici 17 u prilogu. TFS se može otvoriti u bilo kojem pregledniku, a pristupa mu se pomoću korisničkog imena i lozinke.

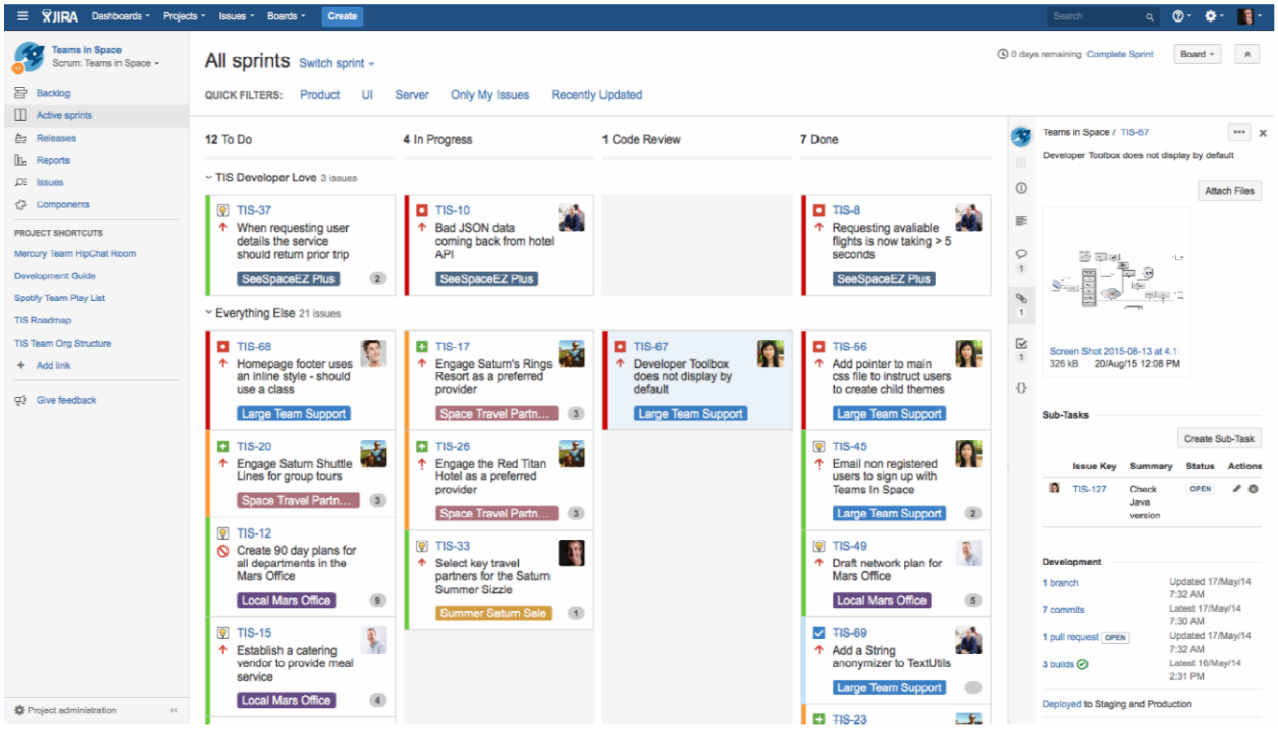


Slika 17 Izgled korisničkog sučelja TFS-a. [13]

7.3 Jira

Jira je alat koji omogućava lakši timski rad.Jira ima ploču (*engl. board*) za planiranje i praćenje izmjena značajki, mogućnost puštanja aplikacije u produkciju (*engl. release)*te izrade izvještaja, kao i mogućnost praćenja prilagođenog toka izvršavanja zadataka po našim osobnim i specifičnim potrebama (Jira, 2018).Izgled Jire vidljiv je na slici 18. Zadatke definira i rješava tim, a oni mogu biti u više različitih stanja (kolona), slično kao i na TFS-u.

[13]Preuzetosa: https://blogs.msdn.microsoft.com/bharry/2017/07/24/tfs-2017-update-2-rtm/



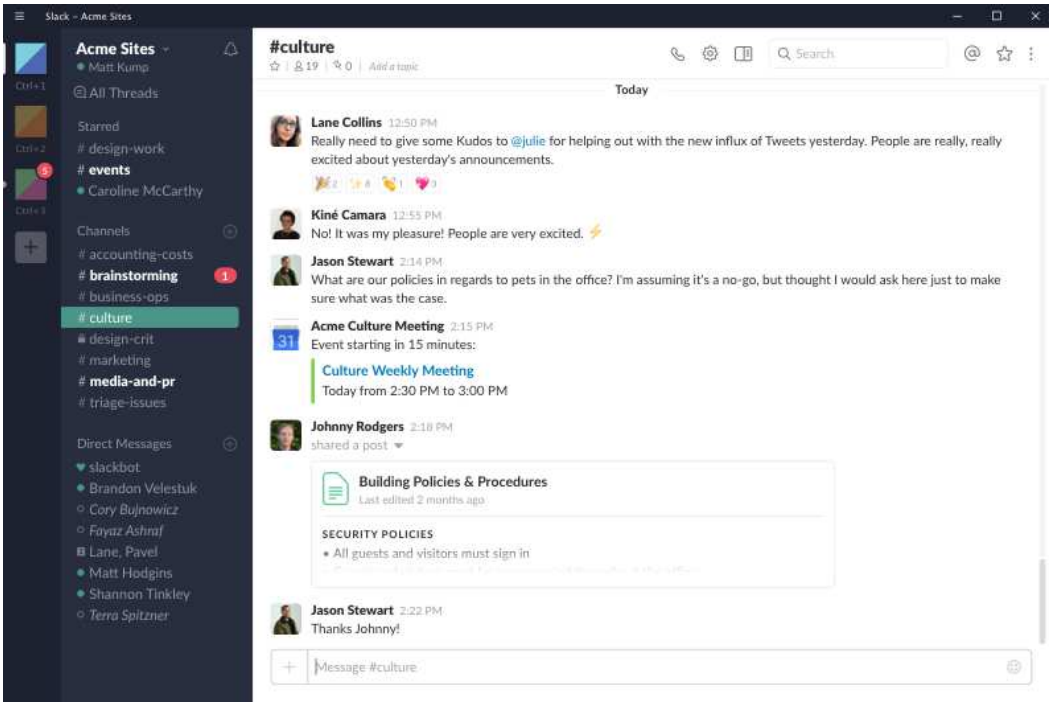
Slika 18 Ploče sa zadacima na Jiri[14]

7.4 Slack

Slack je alat za komunikaciju izrazito korišten u modernim IT tvrtkama (Slack, 2018). Omogućava slanje tekstualnih i medijskih poruka, izradu zasebnih kanalaovisno o ograncima firme i slično. Postoji desktop i mobilna verzija Slacka (Slack, 2018).

Na primjer, može se otvoriti jedan kanal za frontend razvojne stručnjake, drugi za administratore baza podataka, treći za android programere i jedan generalni kanal, u kojem su svi članovi tima ili tvrtke. S tako organiziranim komunikacijskim kanalima rješava se problem „*spamanja*“ poruka. Izgled korisničkog sučelja Slacka vidljiv je na slici 19.

[14]Preuzetosa: https://medium.com/halting-problem/pm-plans-out-entire-life-on-jira-c17fce5500b9



Slika 19 Izgled korisničkog sučelja Slacka[15]

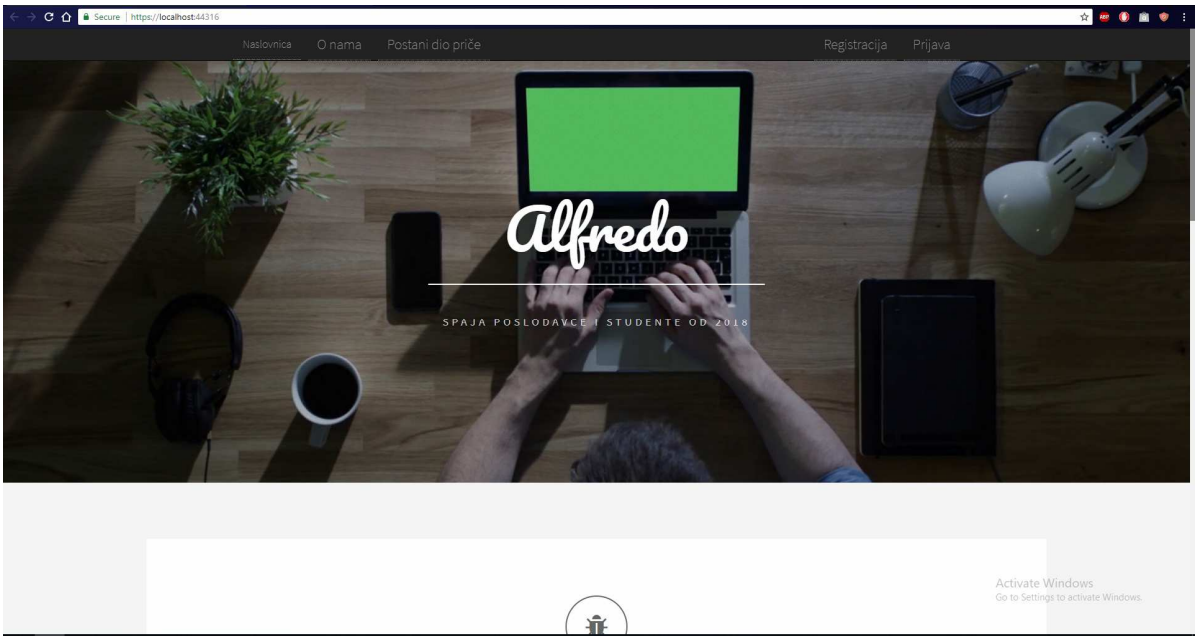
[15]Preuzetosa:https://www.lifewire.com/slack-tips-4137670

8 Alfredo : aplikacija za povezivanje IT studenata i poslodavaca u RH

Alfredo (slika 25) je naziv za web aplikaciju razvijenu za praktične potrebe ovog diplomskog rada- Napisana je u C# programskom jeziku i unutar modernog .NET core radnog okvira na backendu te pomoću HTML5, CSS-a i parJavaScript radnih okvira na frontendu.

Namjena aplikacije je spojiti IT studente i poslodavce u Republici Hrvatskoj. Uočeno je kako IT poslodavci u RH imaju problema pronalskakvalitetne radne snage. Ljude koje zaposle često trebaju doškolovati jer znanje steknuto tijekom studija na fakultetima često nije dovoljno za rad na stvarnim projektima modernih tvrtki. Ako student tijekom studija nije odvojio dio slobodnog vremena na proučavanje i vježbanje programiranja van nastavnog programa fakulteta, teško da će imati potrebno znanje na modernom tržištu rada. Slično je i s dizajnom, kojeg također treba vježbati kroz rad na konkretnim projektima.

Alfredomože poslužiti kao posrednik između poslodavaca i studenata, posrednik koji studentima omogućava pokazivanje znanja i vještina te pronalazak posla,a poslodavcima pronalazak kvalitetne radne snage putem principa objave poslova i zadataka za dotično radno mjesto.



Slika 20 Izgled naslovne strane Alfredo aplikacije

8.1 Tehnologija

U ovom dijelu objašnjene su tehnologijekorištene na izradi Alfreda.

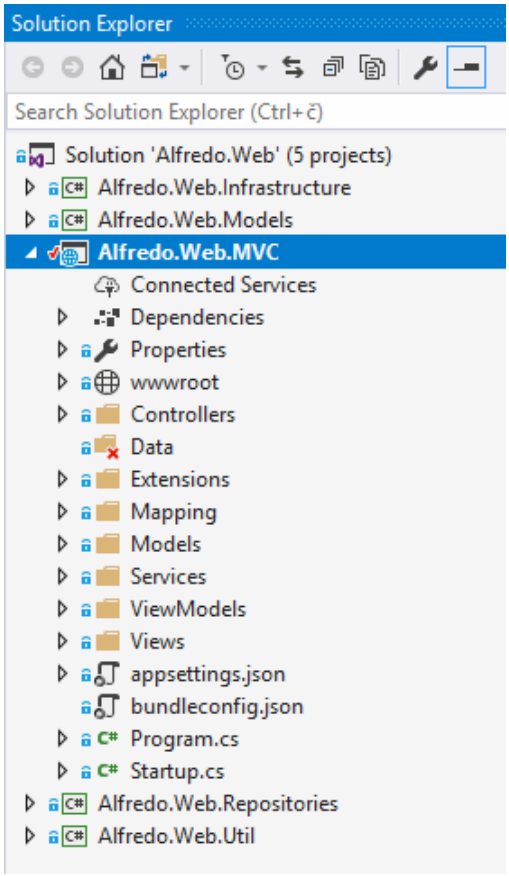
Čitav projekt je napravljen pomoću .NET core radnog okvira. .NET core radni okvir je moderni okvir kojeg Microsoft zadnjih dvije godine svakih nekoliko mjeseci ažurira,a predstavlja laganu verziju klasičnog .NET okvira (Microsoft, 2018).

.NET core sadrži slične nazive bibliotekakao što je imao klasični .NET, samo što su one znatno optimizirane. Primjerice, kao *ORM*(*engl. objectrelationalmapping*) radni okvir se sada koristi tzv. *Entityframeworkcore*, dok se u .NET-u koristio *Entityframework*. Za razliku od klasičnog, on je puno optimiziraniji, što rezultira boljim i bržim upitima prema bazi, budući da je zadaća ORM-ova mapirati upite iz koda u SQL upite i poslat ih na bazu (Shaw &Garlanm, 1996).

Na frontenduje mrežu elemenata (engl. *grid*) složena pomoću bootstrap CSS radnog okvira, budući da omogućava brzo slaganje mreže, a i čini stranicu responzivnom, dok se za ostvarenje animacija (na primjer animacija prijelaza) koristilo par različitih, besplatnih,JS biblioteka otvorena koda.

8.2 Arhitektura

Arhitektura Alfreda je DDDbudući da je Alfredo aplikacija koja se potencijalno može izuzetnoskalirati i proširiti s raznim funkcionalnostima s obzirom na svoju domenu (svijet potražnje i ponude poslova). Na slici 21 može se vidjeti snimak zaslona arhitekture Alfreda učinjen unutar VisualStudia. Kao što je vidljivo i na slici, rješenje web aplikacije Alfredo ima 5 zasebnih projekata.



Slika 21 Arhitektura Alfreda uVisual Studiju

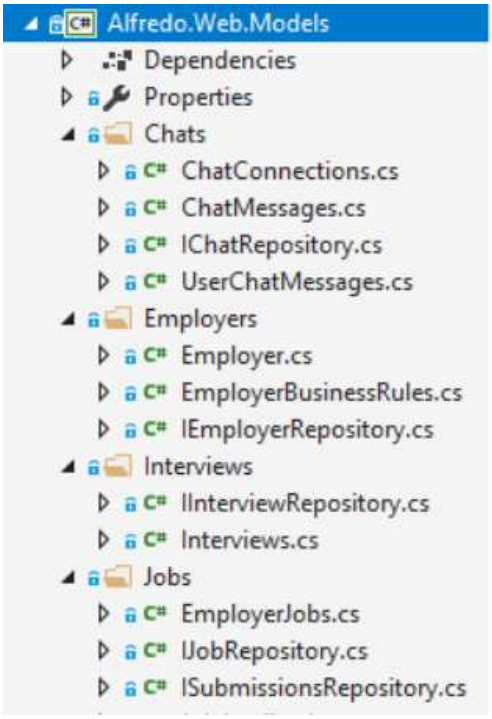
8.2.1 Projekt infrastrukture

Projekt infrastrukture je projekt o kojem ovise svi niži slojevi(projekti),a koji sadrži neke fundamentalne stvari iskoristive i u drugim projektimapoput sučelja korijenskog agregata (*IAggregateRoot*), generičkog repozitorija, klase potrebne za loger na razini baze koji će zapisivati ulovljene pogreške na stranici u bazu, *entity base* koji čini osnovu svake tabliceu ovome projektu napravljene *codefirst* pristupom i slično.

8.2.2 Projekt modela

Projekt modela (*Models*) je projekt koji čini srž DDD arhitekture. Models projekt je poslovna bitprojekta. Kad god je potrebno proširiti bazu novom tablicom, u taj projekt se dodaje istoimena klasu u za to predviđenu mapu. Klase ovdje, dakle, reflektirajutablice u bazi. Na primjer, klasa *Employer.cs* odgovara tablici zaposlenika (*Employers*) u bazi, klasa *Jobs.cs* tablici poslova (*Jobs*) i slično.

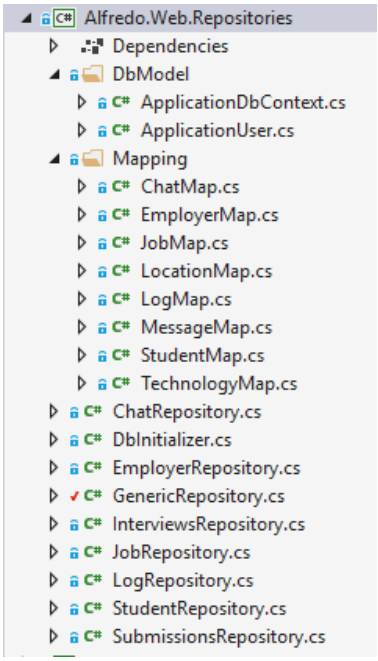
Tu se također nalaze i sučeljaspecifičnih repozitorija,sučelja koja implementiraju klase repozitorijskog projekta.U projektu modela postoje klase poruka, studenata, poslodavaca, adresa i druge (slika 22), a na temelju njih se generiraju istoimene tablice u bazi.



Slika 22 Klase modela spremljene su u Models projektu unutar rješenja

8.2.3 Projekt repozitorija

Projekt repozitorija (*repositories*) je zapravo implementacija (slika 23)repozitorijskog oblikovnog obrasca. Zbognjega MVC sloj ne ovisi direktno ni o kojoj konkretnoj bazi,što smanjuje međuovisnost u kodu i omogućava fleksibilniji razvoj. Uloga ovog sloja, općenito, je primati i slati podatke bilo prema bazi ili prema API-u. U slučaju Alfredo aplikacije, ovaj sloj komunicira s bazom. Kod bazne klasegeneričkog repozitorija može se vidjeti na kodnome bloku 3. u prilogu.



Slika 23 Pregled mapa i pripadajućih klasa repozitorijskog projekta aplikacije

8.2.4 Util projekt

U zadnjem sloju se nalazi*util* projekt, kao mini projekt u koji se mogu dodavati *linq*i slične ekstenzije. Kod klase ekstenzija ovog projekta može se vidjeti u kodnom bloku 4. u prilogu.Klasa ekstenzije predstavlja linq ekstenziju za stringove po čitavom projektu. Metodevidljive u kodnom bloku4.mogu se pozvati nad bilo kojim objektom tipa string u projektu. Na primjer, metoda *WordCount* se može pozvati nad bilo kojim objektom tipa string i vratit će broj pronađenih riječi u stringu nad kojim je pozvana (npr. za string „idem doma sretan“ vratit će tri)

8.2.5 MVC projekt

MVC projekt jezadani projekt prilikom izrade novog web rješenja u VisualStudiui premda je i u Alfredu zadržaosvoju osnovnu funkciju „startup projekta“,koristi se samo za frontend stvari te kao posrednikizmeđu frontenda i backenda. Ideja je bilaputem DDD arhitekturerasteretiti kontrolere na način da oni služe za dohvat podatka iz odgovarajućeg repozitorija (npr. chat poruka nekog korisnika iz *chat*repozitorija ili podatka o profilu zaposlenika iz *employer* repozitorija) i otpremu u dotičnipogledili kao odgovor na asinkroni ajax poziv iz JavaScripta.U MVC projektu nalaze se i viewmodelikoji se mapiraju na modele (i obrnuto) putemautomapera koji je ubačen u projekt kao nuget paket (nuget, 2018). To je dobra praksa jer na pogled (*view*)korisniku nikako ne bi trebalo slati čitave objekte s nepotrebnim podacimanego samo ona svojstva (i njihove vrijednosti) koja trebaju tom specifičnom pogledu.

9 Zaključak

Izrada aplikacija u modernim IT tvrtkama zahtjeva pomnu poslovnu, financijsku, tržišnu i projektnu analizu. Napraviti web aplikaciju uz sav dostupan edukativni materijal na internetu nije teško. Napraviti kvalitetnu i održivu aplikaciju, jest. Jednako je bitno pažljivo biranje prikladne metodologije razvoja kaoi prikladne arhitekture sustava. Obje stavke igraju važnu ulogu u cjelokupnom procesu timskog razvoja suvremene aplikacije.Odabir metodologije koja nije u skladu s kulturom tvrtke ili preferencama članova tima može u jednakoj mjeriti naškoditi vremenu isporuke kao što to može i pisanje nekvalitetnog koda ili loše postavljena arhitektura sustava. U oba slučaja važno se držati određenih principa i konvencija.

Kod odabira metodologija se preporuča isprobati primjenu neke od postojećih metodologija razvoja kao što je SCRUM, baš kao što je i u procesu pisanja koda, i razvoja tehničkog dijela aplikacije, preporučljivo primjenjivati konvencije kodiranja koje propisuje zajednica, ali i sama tvrtka.Smislenije je implementirati opće poznato, učinkovitoi isprobanorješenje za neki problem, nego trošiti vrijeme na osmišljavanje novogkojemožda na kraju ne će biti ni tako efikasno. Upravo iz tog razloga treba prilikom svakog tehničkog razvoja na umu imati najbitnije principe programiranja kao što su SOLID, DRY ili KISS, neovisno o tome radi li se o razvoju video igre ili programiranju modula rakete. Ti principi su nastali s razlogom, kao rezultat višegodišnje programerske prakse, a zahvaljujući njima razvijeni su i razni oblikovni obrasci.

Na temelje proučene literature i dosadašnjeg iskustva razvoja aplikacija, mogu zaključiti kako je puno bitniji način razmišljanja programera, negonjegovo poznavanje nekog konkretnog programskog jezika, okruženja za razvoj ili radnog okvira.Naravno da je bitno pratiti tehnologije te nastanak i razvoj modernih radnih okvira jer su oni temeljni alat za razvoj, no za samog programera puno je bitnijinačin razmišljanja i sposobnost sagledavanja problema.Upravo je poznavanje principa i mogućnost implementacije oblikovnog obrasca po potrebi ono što tvrtke traže kod programera jer se zahvaljujući takvim zaposlenicima nastavlja pisati kvalitetan i održiv kod, odnosno kod koji čini arhitekturu prilagođenu poslovnim potrebama i domeni, a ne obrnuto.

10 Literatura

1. Albrecht,A.J., Gaffney, J.E., Software Function, Source Lines ofCode, and Development EffortPrediction: A Software Science Validation, 1983. [5.6.2018.], dostupno na :https://ieeexplore.ieee.org/abstract/document/1703110/

2. Allen,J.J., Chudley. J.J., Smashing UX design: Foundations for designing online userexperiences [e-book], 2012. [1.6.2018.], dostupno na :https://www.google.com/books?hl=hr&lr=&id=ANX-_0Yn8-wC&oi=fnd&pg=PR9&dq=ux+design&ots=dPHMq6vLgk&sig=SIQWc7xHpW5a1jTiYczp0y8lC78

3. Arkin, B., Stender, S., McGraw, G., Software PenetrationTesting, IEEE [online], 2005. [2.6.2018.], dostupno na: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1392709

4. AutoMapper, nuget [online], 2018. [25.6.2018], dostupno na: https://www.nuget.org/packages/automapper/

5. Automatsko testiranje web aplikacija, Evolva [online], 2018. [3.6.2018.], dostupno na: http://www.evolva.hr/hr/automatsko_testiranje_web_aplikacija.html

6. Barth, A., Jackson, C., Mitchell, J.,C., RobustDefenses for Cross-Site RequestForgery, Proceedingsofthe 15th ACM Conference on Computer andCommunicationSecurity (CCS) [online], 2008. [26.6.2018.] dostupno na:https://classes.soe.ucsc.edu/cmps223/Spring09/Barth%2008.pdf

7. Beciric, D. Funkcionalna specifikacija softvera [online]; 2015. [28.5.2018.], dostupno na: https://www.linkedin.com/pulse/funkcionalna-specifikacija-softvera-uvod-dejan-beciric/

8. Beck, K., Test-driven development: byexample [e-book], 2003. [3.6.2018.], dostupno na: https://books.google.hr/books?hl=hr&lr=&id=CUlsAQAAQBAJ&oi=fnd&pg=PR7&dq=test+driven+development&ots=QBd0Z6_RNZ&sig=rv5xTx-f7fb0_vt_-

XOYLMnQ0NM&redir_esc=y#v=onepage&q=test%20driven%20development&f=false

9. Bethke, E., Game development andproduction [e-book], 2003, [28.5.2018.], dostupno na: https://books.google.hr/books?hl=hr&lr=&id=m5exIODbtqkC&oi=fnd&pg=PR3&dq=game+development+roles&ots=owAtUzcmMP&sig=9Qt9dCCwxnotBkfSRj9lq6vu4dQ&redir_esc=y#v=onepage&q=game%20development%20roles&f=false

10. Boehm, B.W., A spiral model of software development andenhancement, Computer [online], 1988. [3.6.2018.], dostupno na:http://www.dimap.ufrn.br/~jair/ES/artigos/SpiralModelBoehm.pdf

11. Burns,R.N., Dennis, A.R., Selectingtheappropriateapplication development methodology , SIGMIS [online], 1985. [31.5.2018], dostupno na : https://dl.acm.org/citation.cfm?id=1040696

12. Carmel, E., Global Software Teams: ColloboratingAcrossBordersand Time Zones [e-book], 1999. [1.6.2018.], dostupno na : https://qckwg0tgb03.storage.googleapis.com/MDEzOTI0MjE4WA==03.pdf

13. Cohn, M., Userstoriesapplied: For agile software development [e-book],2004., [1.6.2018.], dostupno na: https://www.google.com/books?hl=hr&lr=&id=SvIwuX4SVigC&oi=fnd&pg=PR15&dq=user+stories&ots=VqTff9pYMN&sig=Q-X0E_XY8u4UjzjiCRRbls9eIJo

14. Danenas, P., Garsva, G., DomainDriven Development andFeatureDriven Development for Development ofDecisionSupport Systems, International Conference on Informationand Software Technologies, 2012. [29.6.2018.], dostupno na:https://link.springer.com/chapter/10.1007/978-3-642-33308-8_16

15. Definitionofapplication, Merriam-Webster; 2018. [31.5.2018.], dostupno na: https://www.merriam-webster.com/dictionary/application

16. EDUCBA, WhatisApplication Software &ItsTypes, 2015. [1.6.2018.], dostupno na: https://www.educba.com/what-is-application-software-its-types/

17. Ein-Dor,P., Myers, M., Raman, K.S., IT industry development andtheknowledgeeconomy: A fourcountrystudy, Journal of Global Information [online], 2004. [2.6.2018.], dostupno na: http://www.umsl.edu/~lacitym/fourcountry.pdf

18. Evans, E., Domaindriven design [book], 2004

19. Gamma, E., Examples to Accompany: Design PatternsElementsofReusableObject-Oriented Software [e-book], 1995. [2.6.2018.], dostupno na: http://asi.insa-rouen.fr/enseignement/siteUV/genie_logiciel/supports/ressources/exemples_de_la_vie_reelle_pour_illustrer_pattern.pdf

20. Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns: ElementsofReusableObject-Oriented Software [e-book] 1994. [27.6.2018.] dostupno na :https://sophia.javeriana.edu.co/~cbustaca/docencia/DSBP-2018-01/recursos/Erich%20Gamma,%20Richard%20Helm,%20Ralph%20Johnson,%20John%20M.%20Vlissides-Design%20Patterns_%20Elements%20of%20Reusable%20Object-Oriented%20Software%20%20-Addison-Wesley%20Professional%20(1994).pdf

21. Gates, B., Myhrvold, N., Rinearson, P., TheRoadAhead, [e-book],1995. [26.6.2018.] dostupno na:http://book.ilkaddimlar.com/d_pdf_book_komputerler_23543.do

22. Halfond, W.G.J, Viegas, J., Orso, A., A Classificationof SQL InjectionAttacksandCountermeasures, Proceedingsofthe IEEE and ACM International Conference on Automated Software Engineering (ASE 2006) [online], 2006. [25.6.2018], dostupno na:https://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf

23. Hecksel, D., System andmethod for software methodologyevaluationandselection, 2004. [5.6.2018.], dostupno na :https://patents.google.com/patent/US20040243968A1/en

24. How to document a Software Development Project, Smartics [online]; 2018. [28.5.2018.], dostupno na: https://www.smartics.eu/confluence/display/PDAC1/How+to+document+a+Software+Development+Project

25. IEEE, 830-1984 — IEEE Guide to Software RequirementsSpecifications. 1984. [31.5.2018.], dostupno na: https://ieeexplore.ieee.org/document/278253/

26. Jira software, Jira [online], 2018. [12.5.2018.], dostupno na: https://www.atlassian.com/software/jira

27. Lin, L.C., Lee, W.O, UI toolkit for non-designersintheenterpriseapplicationsindustry, CHI'07 ExtendedAbstracts on Human FactorsinComputing Systems [online], 2007. [2.6.2018.], dostupno na:https://dl.acm.org/citation.cfm?id=1240902

28. MacLennan, B.J., Principlesofprogramminglanguages: design, evaluation, andimplementation [e-book], 1987. [14.5.2018.], dostupno na: https://gtnn2ie01.storage.googleapis.com/MDE5NTExMzA2Mw==01.pdf

29. Metodologije razvoja softvera [slides], 2015. [31.5.2018.], dostupno na : http://predmet.singidunum.ac.rs/pluginfile.php/14195/mod_folder/content/0/2%20-%20Metodologije%20razvoja%20softvera%202017.pptx?forcedownload=1

30. Mitchell, S.,M., Seaman, C.B., A Comparisonof Software Cost, Duration, andQuality for Waterfall vs. IterativeandIncremental Development: A SystematicReview, IEEE [online], 2009. [26.6.2018.] dostupno na: http://www.cin.ufpe.br/~in1037/AllFinal/SE52%20Mitchell%202009.pdf

31. Moran,T.P., TheCommandLanguageGrammar: a representation for theuserinterfaceofinteractivecomputersystems, International Journal of Man-MachineStudies 15(1) [online], 1981. [24.5.2018.], dostupno na: https://www.sciencedirect.com/science/article/pii/S0020737381800223

32. Palmer, S.R., Felsing, J.M.. A PracticalGuide to Feature-Driven Development [e-book], 2002, [6.6.2018.], dostupno na :https://dl.acm.org/citation.cfm?id=600044

33. Pawlak, Z., Informartionsystemstheoreticalfoundations, Information Systems 6(3) [online], 1981. [1.6.2018.], dostupno na : https://www.sciencedirect.com/science/article/pii/0306437981900235

34. Penetracijski test web aplikacija, Infigo [online], 2018. [3.6.2018.], dostupno na: http://www.infigo.hr/penetracijski-test-web-aplikacija-s78?t=1

35. Riley, T., Goucher, A., BeautifulTesting: LeadingProfessionalsReveal How TheyImprove Software [e-book], 2009. [26.6.2018.] dostupno na: http://yourbookchoice.com/downloads/beautiful-testing-leading-professionals-reveal-how-they-improve-software-theory-in-practice.pdf#

36. Saff, D., Ernst, M.D., Anexperimentalevaluationofcontinuoustestingduring development, ACM SIGSOFT Software Engineering Notes [online], 2004. [2.6.2018.], dostupno na:https://dl.acm.org/citation.cfm?id=1007523

37. Schwaber, K., Beedle, M., Agile software development withscrum [e-book] 2002. [15.5.2018.], dostupno na : http://sutlib2.sut.ac.th/sut_contents/H129174.pdf

38. Shaw, M., Garlanm, D., Software Architecture: Perspectives on anEmerging Discipline [e-book] 1996. [15.5.2018.], dostupno na: http://acme.able.cs.cmu.edu/pubs/show.php?id=192&type=both

39. Slack naslovna stranica, Slack [online], 2018. [4.6.2018.], dostupno na: https://slack.com

40. Spolsky, J., PainlessFunctionalSpecifications – Part 1: WhyBother?, Joel On Software [online]; 2000. [28.5.2018.], dostupno na: https://www.joelonsoftware.com/2000/10/02/painless-functional-specifications-part-1-why-bother/

41. Stroustrup, B., Programming: PrinciplesandPracticeUsing C++ [e-book], 2008. [26.6.2018.] dostupno na:https://ptgmedia.pearsoncmg.com/images/9780321992789/samplepages/9780321992789.pdf

42. Sussman,B.,C., Fitzpatrick, B.W., Pilato, C.M., VersionControlwithSubversion [e-book], 2008. [26.6.2018.] dostupno na:https://svn.di.unipmn.it/svn-book-1.7.pdf

43. Što je to XSS?, Acunetix [online], 2018. [2.6.2018.], dostupno na: http://aquilonis.hr/acunetix/cross_site_scripting.html

44. Tanenbaum, A.S., Operatingsystems: design andimplementation [e-book], 1987. [1.6.2018.], dostupno na : http://csci.csusb.edu/ykarant/courses/w2005/csci599/syllabus.ps

45. Team Foundation Server, VisualStudio [online], 2018. [26.6.2018.] dostupno na: https://visualstudio.microsoft.com/tfs/

46. Trello, 2018. [2.6.2018.], dostupno na: https://trello.com

47. Whatis SCRUM?, SCRUM.org [online], 2018. [26.6.2018.] dostupno na:https://www.scrum.org/resources/what-is-scrum

48. WhatisSelenium?, SeleniumHQ, 2018. [16.6.2018.], dostupno na: https://www.seleniumhq.org

49. Wirts, S., Gary,A., Lazarus, E., Software application development methodsandframework, 2006. [28.5.2018.], dostupno na: https://patents.google.com/patent/US7076766B2/en

50. Wynne, M., Hellesoy, A., Tooke S., Thecucumberbook: behaviour-driven development for testersanddevelopers [e-book] 2017. [2.6.2018.], dostupno na:https://books.google.hr/books?hl=hr&lr=&id=fA9QDwAAQBAJ&oi=fnd&pg=PT14&dq=behaviour+driven+development&ots=_mVhWZ-0pf&sig=YjjyVTwF0uys5TCDgmUaxGiWAEg&redir_esc=y#v=onepage&q=behaviour%20driven%20development&f=false

11 Prilozi

```
Function MyBooleanFunction() As Boolean

    Dim someBooleanValue As Boolean = SomeBooleanFunction()
    Dim anotherBooleanValue As Boolean

    If someBooleanValue = True Then
        anotherBooleanValue = True
    Else
        anotherBooleanValue = False
    End If

    Return anotherBooleanValue

End Function
```

Kodni blok 1 Primjer nepoštivanja KISS principa u kodu.

```
1   using System;
2   using Microsoft.VisualStudio.TestTools.UnitTesting;
3
4   namespace IDGUnitTests
5   {
6       [TestClass]
7       public class Primjer
8       {
9           public void SampleTest()
10          {
11              Assert.AreEqual(10, Sum(5, 5);
12          }
13
14          private int Sum(int x, int y)
15          {
16              return x + y;
17          }
18      }
19  }
```

Kodni blok 2. Primjer unit testa u Visual Studiju.

```
1.  using Alfredo.Web.Data;
2.  using kd.Infrastructure.Domain;
3.  using Microsoft.EntityFrameworkCore;
4.  using Microsoft.Extensions.Logging;
5.  using System;
6.  using System.Collections.Generic;
7.  using System.Linq;
8.  using System.Linq.Expressions;
9.  using System.Threading.Tasks;
10.
11. namespace Alfredo.Web.Repositories
12. {
13.     public abstract class GenericRepository<T> : IGenericRepository<T>
14.       where T : class
15.     {
16.         private readonly ApplicationDbContext _context;
17.         protected DbSet<T> Dbset;
18.
19.         protected GenericRepository(ApplicationDbContext context)
20.         {
21.             _context = context;
22.             Dbset = _context.Set<T>();
23.         }
24.
25.         /// <summary>
26.         /// adds passed entity to database and returns its id
27.         /// </summary>
28.         public virtual void Add(T entity)
29.         {
30.             try
31.             {
32.                 Dbset.Add(entity);
33.             }
34.             catch (Exception)
35.             {
36.                 //log
37.             }
38.             Save();
39.         }
40.
41.
42.         public virtual void Delete(T entity)
43.         {
44.             Dbset.Remove(entity);
45.             Save();
46.         }
47.
48.
49.         public void Update(T entity)
50.         {
51.             try
52.             {
53.                 _context.Entry(entity).State = EntityState.Modified;
54.             }
55.             catch (Exception ex)
56.             {
57.                 //log
58.             }
59.             Save();
60.         }
61.
62.         public async Task<List<T>> FindByAsync(Expression<Func<T, bool>> predicate)
63.         {
```

```
64.             try
65.             {
66.                 return await Dbset.Where(predicate).ToListAsync();
67.             }
68.             catch (Exception)
69.             {
70.                 //log
71.                 return null;
72.             }
73.         }
74.
75.         /// <summary>
76.         /// fetch T by its id
77.         /// </summary>
78.         public T GetById(int id)
79.         {
80.             try
81.             {
82.                 return Dbset.Find(id);
83.             }
84.             catch (Exception)
85.             {
86.                 //log
87.                 return null;
88.             }
89.         }
90.
91.
92.         public virtual IEnumerable<T> GetAll()
93.         {
94.             try
95.             {
96.                 return Dbset.AsEnumerable<T>();
97.             }
98.             catch (Exception)
99.             {
100.                //log
101.                return null;
102.            }
103.        }
104.
105.
106.        private void Save()
107.        {
108.            try
109.            {
110.                _context.SaveChanges();
111.            }
112.            catch (Exception ex)
113.            {
114.
115.                //log
116.            }
117.        }
118.
119.
120.    }
121. }
```

Kodni blok 3. Klasa generičkog repozitorija - bazna klasa svakog specifičnog repozitorija u projektu.

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Text;
4.
5.
6.  namespace ExtensionMethods
7.  {
8.      /// <summary>
9.      /// various useful linq extensions
10.     /// </summary>
11.     public static class Extensions
12.     {
13.
14.         /// <summary>
15.         /// counts words
16.         /// </summary>
17.         public static int WordCount(this String str)
18.         {
19.             return str.Split(new char[] { ' ', '.', '?' },
20.                              StringSplitOptions.RemoveEmptyEntries).Length;
21.         }
22.
23.
24.         /// <summary>
25.         /// trims spaces on both ends and all excess white spaces in between.
26.         /// e.g. for passed string " project    1" will return "project 1"
27.         /// </summary>
28.         public static String RemoveExcessWhiteSpaces(this String str)
29.         {
30.             StringBuilder sb = new StringBuilder(str);
31.             RemoveExcessWhiteSpaces(sb);
32.             return sb.ToString();
33.         }
34.
35.         public static void RemoveExcessWhiteSpaces(StringBuilder sb)
36.         {
37.             if (sb.Length == 0)
38.                 return;
39.
40.             // set [start] to first not-whitespace char or to sb.Length
41.
42.             int start = 0;
43.
44.             while (start < sb.Length)
45.             {
46.                 if (Char.IsWhiteSpace(sb[start]))
47.                     start++;
48.                 else
49.                     break;
50.             }
51.
52.             // if [sb] has only whitespaces, then return empty string
53.
54.             if (start == sb.Length)
55.             {
56.                 sb.Length = 0;
57.                 return;
58.             }
```

```
59.
60.             // set [end] to last not-whitespace char
61.
62.             int end = sb.Length - 1;
63.
64.             while (end >= 0)
65.             {
66.                 if (Char.IsWhiteSpace(sb[end]))
67.                     end--;
68.                 else
69.                     break;
70.             }
71.
72.             // compact string
73.
74.             int dest = 0;
75.             bool previousIsWhitespace = false;
76.
77.             for (int i = start; i <= end; i++)
78.             {
79.                 if (Char.IsWhiteSpace(sb[i]))
80.                 {
81.                     if (!previousIsWhitespace)
82.                     {
83.                         previousIsWhitespace = true;
84.                         sb[dest] = ' ';
85.                         dest++;
86.                     }
87.                 }
88.                 else
89.                 {
90.                     previousIsWhitespace = false;
91.                     sb[dest] = sb[i];
92.                     dest++;
93.                 }
94.             }
95.
96.             sb.Length = dest;
97.         }
98.
99.         /// <summary>
100.        /// converts e.g. "2,3,4" to [2,3,4]
101.        /// </summary>
102.        public static IEnumerable<int> StringToIntList(this String str)
103.        {
104.            if (String.IsNullOrEmpty(str))
105.                yield break;
106.
107.            foreach (var s in str.Split(','))
108.            {
109.                int num;
110.                if (int.TryParse(s, out num))
111.                    yield return num;
112.            }
113.        }
114.
115.    }
116. }
```

Kodni blok 4. Linq ekstenzije za stringove koje se nalaze u util projektu