UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA

ZAKLJUČNA NALOGA

RAZVOJ PROTOTIPA SISTEMA ZA ŠTUDENTSKO MEDSEBOJNO POMOČ

MATEVŽ MAK

MAK

2019

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Razvoj prototipa sistema za študentsko medsebojno pomoč

(Development of a prototype for student mutual assistance)

Ime in priimek: Matevž Mak
Študijski program: računalništvo in informatika
Mentor: prof. dr. Cene Bavec

Koper, september 2019

Ključna dokumentacijska informacija

Ime in PRIIMEK: Matevž MAK

Naslov zaključne naloge: Razvoj prototipa sistema za študentsko medsebojno pomoč

Kraj: Koper

Leto: 2019

Število listov: 47 Število slik: 18 Število tabel: 1

Število referenc: 8

Mentor: prof. dr. Cene Bavec

Ključne besede: Življenjski cikel razvoja aplikacij, aplikacija, uporabniški vmesnik, e-učilnica

Izvleček: Zaključna naloga se osredotoča na predstavitev razvoja prototipa sistema za študentsko medsebojno pomoč. Sistem služi kot orodje, s kateri si lahko študentje delijo zapiske, vprašanja, odgovore in ankete. Sistem je zgrajen na podlagi številnih programskih jezikov, modulov, podatkovne baze in oblačnih storitev. Celoten razvoj je natančneje opisan v zaključnem delu.

Key words documentation

Name and SURNAME: Matevž MAK
Title of the final project paper: Development of a prototype for student mutual assistance
Place: Koper
Year: 2019
Number of pages: 47 Number of figures: 18 Number of tables: 1
Number of references: 8
Mentor: Prof. Cene Bavec, PhD
Keywords: Systems development life cycle, application, user graphical interface, e-classroom
Abstract: The final project paper focuses on presenting the development of a prototype for student mutual assistance system. The system serves as a tool for students to share notes, questions, answers and surveys. The system is build on a variety of programming languages, modules, database and cloud services. The overall development is described in more detail in the final project paper.

ZAHVALA

Rad bi se zahvalil svojemu mentorju doc. dr. Cenetu Bavcu za strokovno pomoč in usmeritve pri izdelavi zaključne naloge.

Zahvaljujem se tudi svoji družini za vsestransko pomoč na moji izobraževalni poti.

Hvala!

KAZALO VSEBINE

KAZALO PREGLEDNIC

KAZALO SLIK IN GRAFIKONOV

SEZNAM KRATIC

SDLC	System development life cycle (življenjski cikel razvoja aplikacij)
EJS	Embedded JavaScript (jezik, ki ima vgrajen JavaScript)
HTML	Hyper Text Markup Language (označevalni jezik HTML)
CSS	Cascading Style Sheets (kaskadne stilske predloge)
CDN	Content delivery network (distributivno omrežje za deljenje vsebine)
API	Application programming interface (aplikacijski programski vmesnik)
JSON	JavaScript Object Notation (objektna notacija v JavaScriptu)
URL	Universal Resource Locator (enolični naslov vira)

1 UVOD

Ljudje se pri uporabljanju interneta srečujemo s številnimi informacijskimi sistemi in se morda tega v tistem trenutku sploh ne zavedamo. Ideje za razvoj informacijskega sistema so neskončne. Uporabnik uporablja te informacijske sisteme s pomočjo grafičnih vmesnikov oz. aplikacij.

V času opravljanja študijskih obveznosti sem ugotovil, da študentje nimamo kvalitetne slovenske socialne platforme, kjer bi lahko študentje sodelovali med sabo. Vsaka fakulteta ima za v ta namen aktivirane elektronske učilnice, vendar se jih študentje ne poslužujejo zaradi problemov kot so anonimnost in prisotnost profesorjev. Študentje se skozi študij srečujejo s številnimi problemi, ki jih je v omejenem času težko rešiti, to pa nas lahko odvrne do študija. V ta namen se je ponudila ideja za razvoj prototipa sistema za študentsko medsebojno pomoč, s katerim si bodo lahko študentje delili znanje in vprašanja.

Namen zaključne naloge je opisati načrtovanje in implementacijo prototipa sistema za študentsko medsebojno pomoč z uporabo številnih metodologij.

V teoretičnem delu zaključne naloge smo predstavili številne modele in njihove razlike. Podatke smo našli s pomočjo strokovnih in internetnih virov. Prav tako je opisan življenjski cikel razvoja spletne aplikacije.

V fazi načrtovanja smo upoštevali izbran model za razvoj prototipa. Pri tem smo upoštevali uporabniške zahteve, programerske jezike in orodja. Na podlagi zahtev smo prikazali številne diagrame, ki prikazujejo potek in uporabo sistema. Definirali smo tudi ogrodje podatkovne baze in njen slovar.

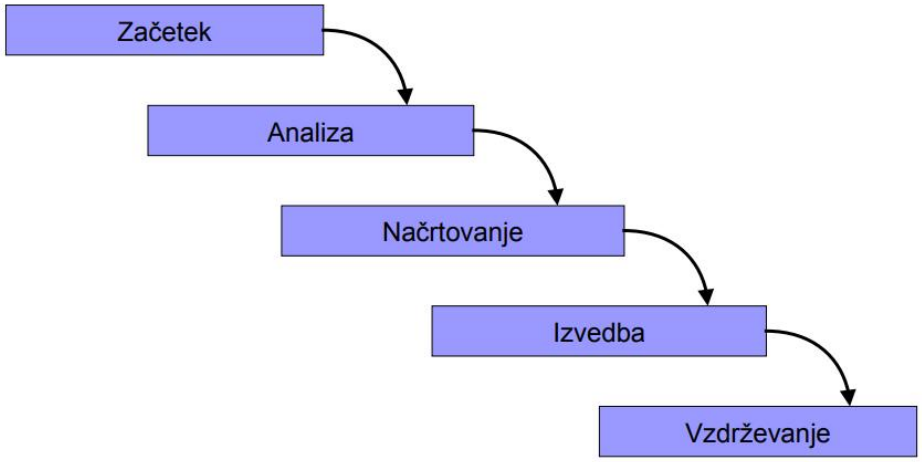
Na podlagi diagramov, načrtovane podatkovne baze in definiranih zahtev sledi faza implementacije. Sistem je zgrajen iz dveh vmesnikov in številnih modulov, ki so zadolženi za opravljanje določene funkcionalnosti. Nekatere rešitve so predstavljene s psevdokodo. Storitve, ki jih uporabnik lahko izvaja so definirane s pomočjo Node.JS. Za izdelavo grafičnih vmesnikov smo uporabili jezike, in sicer CSS, HTML, EJS in okvir Bootstrap.

2 ŽIVLJENJSKI CIKEL RAZVOJA SPLETNE APLIKACIJE

Življenjski cikel razvoja programske opreme vsebuje vse korake od ideje do končnega produkta. Proces se uporablja pri menedžmentu projekta, kjer natančno opišemo vse stopnje razvoja. Ker je proces razdrobljen na več stopenj, lahko lažje ocenimo potrebe oz. zahteve posamezne stopnje. Faze razvoja lahko zamenjajo vrstni red ali pa se celo zgodijo paralelno. Vsaka stopnja je pomembna in nam pomaga pri zagotavljanju kvalitete končnega produkta. V nadaljevanju bomo za opisovanje življenjskega cikla razvoja programske opreme uporabili kratico SDLC[1]. Faze so natančneje opisane v poglavju 2.1. [19]

2.1 Faze

SDLC sloni na globalnem standardu ISO/IEC 12207, ki opisuje vse naloge, potrebne za načrtovanje, gradnjo, preizkušnje in vzdrževanje programskega izdelka. Vsaka faza proizvede končni produkt, ki se nato uporabi v naslednji fazi. [7]



Slika 1: Faze življenjskega cikla razvoja spletne aplikacije

2.1.1 Zasnova sistema

V tej fazi pogledamo sistem v širšem kontekstu na predvideni končni sistem. Potrebno je oceniti ekonomska, zakonodajna, socialna, tehnološka tveganja in koristi. Če projekt ne vsebuje teh tveganj, je treba definirati tehnološke in menedžerske pristope pri razvoju produkta. Najtežji del te faze je definiranje zahtev, ki imajo visoko prioriteto. Te zahteve potem postanejo naši glavni cilji. Po končani fazi moramo imeti definiran menedžerski načrt, projektni plan, ki vsebuje vse zastavljene cilje in časovno obdobje, v katerem mora biti produkt razvit. [20]

2.1.2 Sistemska analiza in zahteve

V tej fazi vzamemo vse cilje iz prejšnje faze in pogledamo, kaj vse bomo potrebovali za razvijanje vsakega od ciljev. Vsak cilj po navadi razdelimo na več podciljev. Za vsak cilj definiramo glavne funkcije in kritična področja, kjer bodo potencialni problemi. Za vsako zahtevo oz. cilj se ustvari dokument, v katerega zapišemo stvari, ki jih potrebujemo za izvedbo. [20]

2.1.3 Načrtovanje sistema

Pri načrtovanju sistema upoštevamo dokumente, ki smo jih zapisali za vsak cilj v prejšnji fazi. Za vsako zahtevo se oblikujejo ena ali več komponent. Vsaka komponenta je plod prototipov, sestankov, delavnic … Komponente vsebujejo natančne opise vseh funkcij produkta, psevdokodo, poslovna pravila, diagrame poteka programske opreme, arhitekturo podatkovne baze in njen slovar. Komponente morajo biti narejene podrobno, da lahko programer opravlja svoje delo s čim manj dodatnih popravkov. [20]

2.1.4 Razvoj sistema

Pri razvoju sistema si pomagamo s komponentami iz prejšnje faze, kjer je podrobno opisana arhitektura sistema. Faza vsebuje tudi UML diagrame. To so diagrami, ki nam pomagajo pri vizualizaciji sistema skupaj z vsemi akterji. Za vsak akter se prikaže njegova vloga in uporaba v sistemu. V tej fazi začnemo z izvedbo produkta, pri čemer nam pomagajo programerji, inženirji za omrežja, inženirji za podatkovne baze. Inženirske ekipe imajo svoje naloge, glavne cilje razdelijo na podcilje. Pri razvoju projekta lahko uporabimo agilne pristope, kot je npr. scrum. Scrum je okvir, ki se uporablja pri ekipnem razvoju, kjer se definirajo cilji in razdelijo naloge med posameznike v ekipi. Vsaka naloga mora iti čez številne faze, preden preide v končni produkt. Izhodni produkt te faze je koda, ki je že pripravljena za testiranje in integracijo. [20]

2.1.5 Integracija in testiranje

V tej fazi poteka sistemska integracija in testiranje, ki ga izvaja strokovnjak za kakovost, ki določi, ali trenutna arhitektura programa izpolnjuje vse cilje, ki jih je stranka zadala. Testiranje se izvaja večkrat, kajti iščemo napake v kodi, poteka tako dolgo, dokler končni uporabnik ni zadovoljen. Po validaciji in verifikaciji končnega uporabnika je produkt uspešno končan. To pomeni, da je kupec potrdil, da produkt izvaja funkcije, za katere so se dogovorili. [21]

2.1.6 Vzdrževanje

Zadnja faza vsebuje vzdrževanje oz. posodabljanje sistema. Moramo zagotoviti, da je arhitektura projekta bila zgrajena modularno, da implementacija (uvajanje) novih komponent ni velik problem. V tej fazi sprejemamo zahteve oz. želje, npr. za hitrejšo izvedbo določenih operacij (optimizacija sistema) in dodajanje novih funkcij sistema. [21]

2.2 Modeli

Za razvoj programske opreme uporabljamo različne modele za različne tipe programske opreme in aplikacije. Ti modeli so sestavni del SDLC. Pomembna faza razvoja programske opreme je izbira najustreznejšega modela, ki ustreza pogojem kot so čas, denar, izkušenost inženirjev… Prednosti in slabosti različnih modelov so zapisane v naslednjih poglavjih. [12]

2.2.1 Splošni model – linearno sekvenčni model

Splošni model je definiran kot splošni življenjski cikel, ki smo ga definirali v poglavju 2.1. Vsebuje niz faz, ki si sledijo zaporedno, torej praviloma ne preidemo v drugo fazo, dokler trenutna ni dokončana. Tak model nima povratnih zank, torej se ne vračamo na prejšnje faze. [6] Linearno sekvenčni model je priljubljen med inženirji, saj je pregleden, vsaka faza pa ima postavljene svoje roke. Vse faze so natančno definirane in to nam omogoča enostavno dodeljevanje nalog. Model je primeren za manjše projekte, kjer je vsaka zahteva razumljena. Primer linearno sekvenčnega modela je slapovni model. [14]

2.2.1.1 Slapovni model

Slapovni model je linearno sekvenčni model. Faze si sledijo sekvenčno, v naslednjo fazo pa ne moremo preiti, dokler trenutna ni dokončana. Po vsaki končani fazi sledi pregled, da lahko sledimo, ali gre projekt v pravo smer. Lastnost tega modela je, da se začne testiranje produkta šele ob koncu razvoja. Slapovni model se priporoča pri projektih, kjer denar in čas nimata primarne vloge. Model se uporablja pri razvoju bančnih sistemov, zdravstvenih sistemov, jedrskih sistemov, torej tam, kjer je pomembno, da izvajanje poteka brezhibno. [23]

Prednosti slapovnega modela [23]:

- Uporaba je enostavna in preprosta.
- Model nam omogoča enostavno upravljanje zaradi razčlenitve, vsaka faza ima definirane zahteve in način pregleda končane faze.

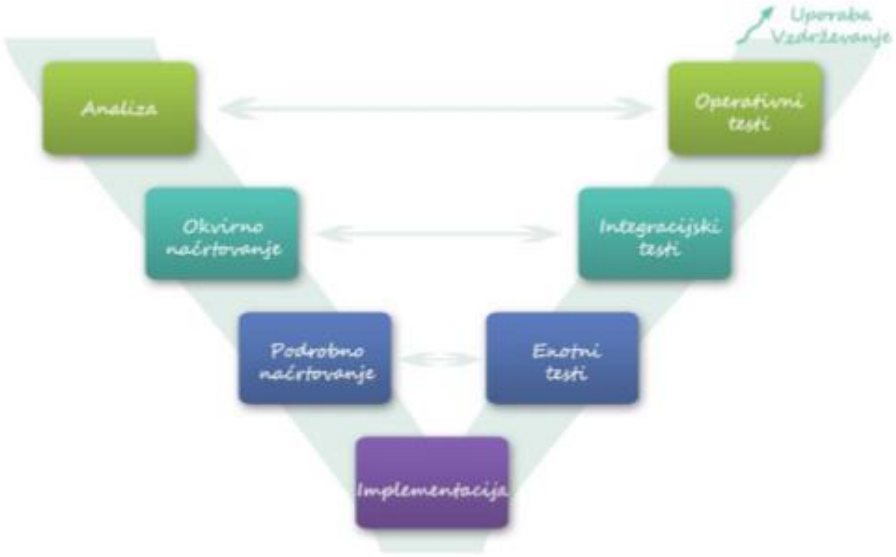
- Faze modela se ne prekrivajo, končujejo se zaporedno ena za drugo.
- Odličen model za manjše projekte, kjer so vse zahteve jasno definirane in razumljive.

Slabosti slapovnega modela[23]:

- Če v fazi testiranje ugotovimo, da je prišlo do napak, je težko popraviti programsko opremo, če ni bila dobro zasnovana v prejšnjih fazah.
- Delujoča programska oprema je prisotna šele proti koncu razvojnega življenjskega cikla.
- Tveganje in negotovost pri dolgotrajnih projektih.
- Model ni primeren za kompleksne objektno orientirane projekte in projekte, ki so v trenutnem izvajanju.
- Tveganje pri projektih, kjer se lahko zahteve velikokrat spremenijo.

2.2.1.2 V Model

Model V je SDLC model, kjer se procesi zgodijo sekvenčno v V obliki (slika 2). Model je znan tudi kot model verifikacije in validacije ter je nadgradnja slapovnega modela. Vsaka razvojna faza vsebuje testno obdobje. Ob zaključku testiranja model preide v naslednjo fazo. Obdobje testiranja je planirano paralelno, imamo verifikacijski del na eni strani in veljavnostni del na drugi strani. V času razvoja oz. programiranja se obe strani združita in dobimo arhitekturo V modela. V fazi analize projekta zbiramo zahteve stranke in jih poskušamo razumeti v tehnološkem razvoju. Ko poznamo vse zahteve, se lotimo sistemskega oblikovanja, kjer podrobno definiramo strojno, programsko opremo, ki jo bomo potrebovali za razvoj končnega produkta. Sistemsko testiranje je zasnovano na planiranju oblikovanja sistema. Naslednja faza je načrtovanje arhitekture sistema, tukaj definiramo vse tehnološke pristope, ki jih bomo uporabili. Načrt je razčlenjen na več modulov za boljšo preglednost in lažje testiranje vsakega modula v kasnejših fazah. Prisotni so tudi diagrami, iz katerih točno razumemo, kako bosta potekala komunikacija med moduli in prenos podatkov do posameznega modula. V tej fazi tudi definiramo integracijske teste, ki so dokumentirani za naslednje faze. Nato sledi modularna arhitektura, kjer preverimo, če so moduli kompatibilni med sabo in zunanjimi sistemi. V tej fazi uvajamo enotne teste, ki nam pomagajo pri identificiranju in eliminiranju potencialnih problemov. V fazi uvajanja sistema se izbere programerski jezik, ki je kompatibilen s planirano arhitekturo sistema. Programska oprema, gre skozi številne preglede in optimizacije za doseganje najboljše izvedbe sistema. Po končanem razvoju sledijo številni testi, npr. enotski, integracijski, sistemski, sprejemni testi. To so veljavnostni testi, pri katerih so nam v pomoč verifikacijski testi, ki smo jih zapisali v predhodnih fazah. [13]



Slika 2: V model [18]

Prednosti V modela:

- Zahteve so dobro definirane in razčiščene.
- Definicija sistema je jasna.
- Tehnologija, ki jo uporabimo pri razvoju, ni dinamična in je dobro razumljena v razvojni ekipi.
- Ne vsebuje ne definiranih in ambicioznih zahtev.
- Primeren za krajše projekte.
- Enostaven za upravljanje, zaradi svoje togosti in tega, da vsaka faza vsebuje pregled.

Slabosti V modela:

- Tveganje in negotovost.
- Ni primeren za kompleksne objektno orientirane projekte.
- Slab model za projekte, ki so v izvajanju.
- Ni primeren za projekte, kjer se lahko zahteve pogosto spreminjajo.
- Ko je projekt v fazi testiranja, je težko spreminjati komponente, če so bile slabo definirane.
- Delujoč produkt je prisoten šele v zadnjih fazah modela.

2.2.2 Postopno se razvijajoči modeli

V postopoma razvijajočih se modelih se pojavljajo mnogi življenjski razvojni cikli. Takšnim modelom pravimo tudi multi-slapovni modeli. Cikli so razčlenjeni na manjše iteracije, da je bolj pregledno in da jih lažje upravljamo. Vsaka iteracija vsebuje analizo

zahtev, načrtovanje, uvajanje in testno fazo. V vsaki iteraciji dobimo kot produkt delujočo programsko opremo. [15]

Prednosti postopno se razvijajočih modelov [15]:

- Hiter razvoj programske opreme, posledično lahko hitreje dobimo povratne informacije od stranke.
- Je bolj dinamičen, lažje spreminjamo in dodajamo zahteve.
- Lažje je testiranje v manjših iteracijah razvoja.
- Bolj učinkovito predvidimo in rešimo tveganja znotraj iteracije.
- Vsaka iteracija vsebuje svoj mejnik.

Slabosti postopno se razvijajočih modelov [15]:

- Problemi pri načrtovanju sistema, kajti vse informacije niso podane na začetku načrtovanja, dodatne zahteve pa lahko porušijo in zaustavijo razvoj sistema.
- Dolgotrajen proces razvoja.

2.2.2.1 Prototipni model in hiter razvoj aplikacij

Prototipni modeli omogočajo hiter razvoj aplikacij. V osnovni fazi se zberejo osnovne informacije o sistemu in nato se razvije prototip. Na podlagi prototipa stranka poda dodatne zahteve, pripombe, s katerimi se razvijejo dodatni prototipi z dodatnimi funkcijami in posodobitvami. Za prvotni model ni časovne omejitve, namen modela je, da se inženirji lotijo najzahtevnejših zahtev in jih potem pokažejo stranki, ki poda povratne informacije. Ta postopek se rekurzivno ponavlja, dokler stranka ni zadovoljna, tako se razvoj sistema zaključi.

Prednosti prototipnega modela [22]:

- Razvoj sistema je tesno povezan s sodelovanjem s stranko, tako ima stranka dober vpogled v razvoj.
- Sistem se razvija v iteracijah, tako se lahko končni uporabnik lažje nauči njegove uporabe.
- Večino napak v kodi ali procesih lahko opazimo že v zgodnjih fazah, ker razvoj poteka v iteracijah.
- Zaradi konstantnih povratnih informacij stranke, lahko dosežemo boljše rešitve in končni produkt.
- Ker razvoj poteka v iteracijah, stranka bolj razume koncept sistema. Tako nam lahko poda dodatne zahteve, ki jih lažje implementiramo. Problemi pri drugih modelih so, da si stranka ne predstavlja vse funkcionalnosti sistema, kasnejše posodobitve tako škodijo načrtovani arhitekturi sistema.

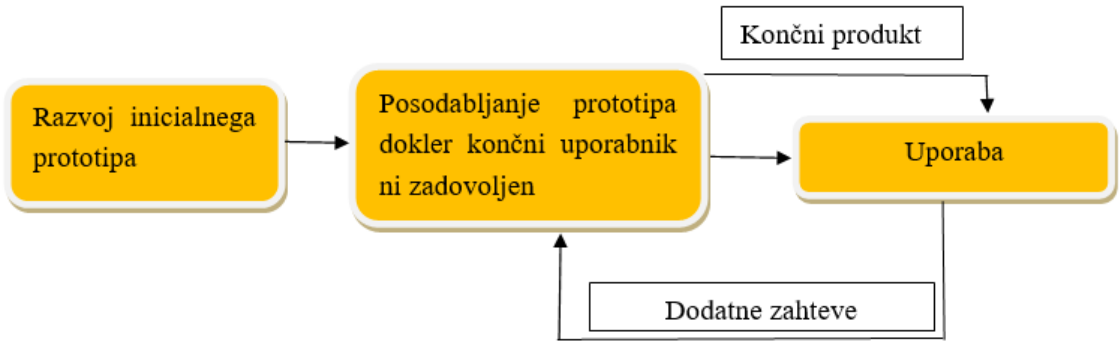
- Hitro delujoča programska oprema.

Slabosti prototipnega modela [22]:

- Področje uporabe sistema se lahko razširi in lahko dobimo večji sistem, kot smo načrtovali.
- Ker je sistem zgrajen iz prototipov, lahko pride do problemov za vizijo končne verzije sistema med člani programerske ekipe. Posledično lahko dobimo slabe rešitve in izdelan končni produkt, ki ga je problem vzdrževati.
- Najslabši izid tega modela je, da stranke mislijo, da je prototip že zaključen sistem. Uporabniki lahko tudi vzljubijo nekatere funkcije, ki niso del končnega produkta.
- Prototipi morajo biti zgrajeni hitro, v nasprotnem primeru lahko razvoj preide v časovne in ekonomske težave.

2.2.3 Model sprotne gradnje in prilagajanja

Modelu sprotne gradnje in prilagajanja rečemo tudi ad hoc model in je uporabljen za razvoj aplikacij brez vnaprej definiranih zahtev in oblikovanja. Namen tega modela je, da se začetni inicialni produkt konstantno posodablja, dokler uporabnik ni zadovoljen. V prvi fazi se ustvari temeljna ideja končnega produkta in njegove funkcionalnosti. Tako se razvije začetni prototip. Uporabnik nato pregleda prototip in poda povratne informacije o dodatnih zahtevah. V primeru, da uporabnik ni zadovoljen oz. je podal dodatne zahteve, se ponovno vrnemo v razvojno fazo, kjer posodobimo programsko opremo. Ta postopek se rekurzivno ponavlja, dokler končni uporabnik ni zadovoljen. [3]



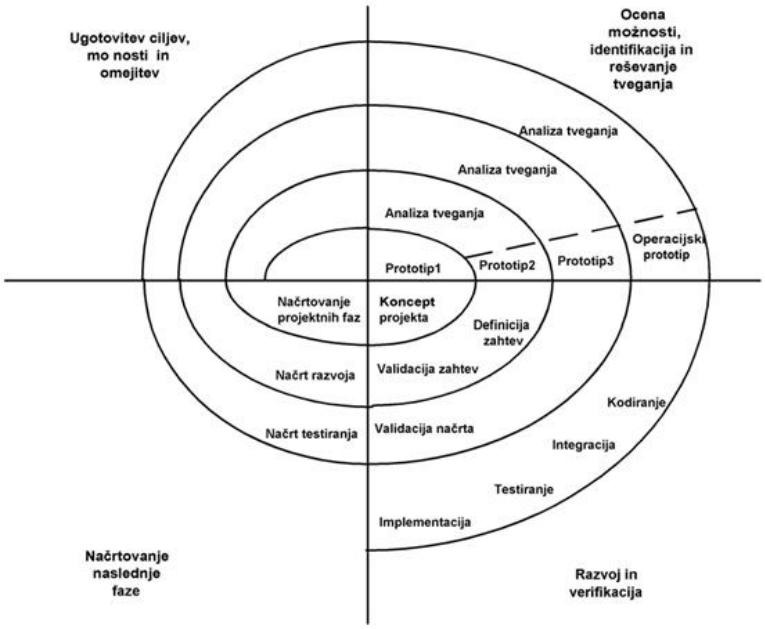
Slika 3: Potek ad hoc modela

Prednosti ad hoc modela:
Za uporabo tega modela ne potrebujemo veliko izkušenj za izvedbo in menedžment projekta. Primeren je za manjše projekte, pri katerih ne potrebujemo veliko načrtovanja. Model uporabimo, ko nimamo veliko časa za razvoj in nam ni treba pisati podrobne dokumentacije. [3]

Slabosti ad hoc modela:
Slabosti tega modela so, da težko sledimo napredku projekta, ker ne uporabljamo načrta. Lahko se tudi zgodi, da bodo stroški razvoja na koncu večji kot pri drugih modelih, zaradi konstantnega popravljanja kode, dodajanja funkcionalnosti. Problemi nastajajo tudi pri vzdrževanju produkta, saj dokumentacija po navadi ni bila narejena. [3]

2.2.4 Spiralni model

Spiralni model je eden izmed najpomembnejših modelov pri razvoju programske opreme. Njegova uporaba nastopi, ko se razvija projekt z visokim tveganjem. Prednost modela je, da mu je dodana tudi analiza tveganja. Zanimiva je struktura modela, ker je v obliki spirale s številnimi zankami. Število teh zank ni točno določeno in je odvisno od projekta do projekta. Vsaka zanka je poimenovana kot faza razvoja programske opreme. Projektni vodja določa, koliko faz vsebuje projekt. Dolžina preseka spirale prikazuje trenutno ceno razvoja. Vsaka faza spiralnega modela je razdeljena v štiri kvadrate. Vsak kvadrat ima svojo funkcionalnost. Namen vsakega kvadrata se ponovi v vsaki zanki razvoja.



Slika 4: Spiralni model [17]

V prvi fazi zanke, poskušamo zbrati vse zahteve in cilje končnega uporabnika. Razumevanje uporabnika nam pomaga pri prepoznavanju problemov in alternativnih rešitev. Prav tako naredimo študijo izvedljivosti, in sicer, ali projekt vsebuje ekonomska, časovna, socialna in zakonodajna tveganja. Druga faza je analiza teh tveganj in kako najti najbolj ugodno alternativno rešitev, ki ustreza časovnemu mejniku in proračunu. Operativni in tehnološki problemi so definirani v tej fazi. Evalvacija vseh teh faktorjev vpliva na nadaljnji razvoj. Tretja faza je implementacija sistema, tukaj se upoštevajo načrt in vse alternativne rešitve, ki smo jih definirali v prejšnji fazi. Pri implementaciji lahko uporabimo tudi druge modele, npr. slapovni ali iterativni model. Preden zaključimo fazo se, opravi tudi testiranje programske opreme. Zadnja faza določa načrt za naslednji cikel spirale. Projektni vodja naredi pregled nad napredkom razvoja in pri tem upošteva vse omejitve oz. faktorje za nadaljnji razvoj. Vsi problemi, ki jih identificirajo uporabnik ali programerska ekipa, so definirani v tej fazi, ravno tako so izvršeni potrebni ukrepi. Po končani fazi preidemo v pod-zanko spirale in postopek se ponovi. Model nam omogoča, da hitro prepoznamo tveganje in tako lahko projekt opustimo v pravem času. [16]

Prednosti spiralnega modela [16]:

- Eden izmed najbolj prilagodljivih modelov. Število faz projekta je odvisno od težavnosti projekta.
- Nadzor projekta je pregleden in enostaven, vsaka zanka vsebuje pregled narejene programske opreme.
- Ocena tveganja je vgrajena v vsako zanko.
- Dodatne zahteve so lahko implementirane v kasnejših zankah.
- Cena razvoja skozi projekt postaja vse bolj realna.
- Primeren za projekte, ki vsebujejo visoka tveganja.

Slabosti spiralnega modela[16]:

- Razvoj projekta je drag.
- Potrebno je imeti izkušene inženirje, ki bodo znali prepoznati tveganja in podati alternativne rešitve.
- Ni primeren za projekte, ki imajo manjša tveganja.
- Težko je oceniti, kakšen proračun bo zahteval projekt in koliko časa bomo potrebovali za razvoj.
- Pri projektih, kjer je veliko dokumentacije, lahko pride do težav pri menedžmentu projekta.

3 NAČRTOVANJE SPLETNE APLIKACIJE

V naslednjih podpoglavjih bomo uporabljali metodologijo, ki smo jo opisovali v prejšnjih poglavjih. V prvi fazi bomo govorili o analizi zahtev in sistema. Pri analizi zahtev definiramo vse zahteve uporabnika in metode oz. tehnologije, ki nam bodo pomagale pri razvoju teh zahtev. Glavne zahteve lahko razdelimo na več pod ciljev tako, da bomo imeli lažji pregled nad razvojem. V drugi fazi sledi načrtovanje, kjer upoštevamo vse zahteve in podrobno definiramo funkcije, psevdokodo, podatkovno bazo in njen slovar. V tej fazi so tudi primerni razni diagrami uporabe in poteka programske opreme.

3.1 Uporabniške zahteve

- Celoten sistem je namenjen študentom. Potrebujemo vmesnik, kjer lahko uporabnik ustvari račun in se prijavi v sistem. Vsak uporabnik lahko unikatno generira svoje socialno okolje na spletni aplikaciji. Potrebujemo vmesnik, ki omogoča kreiranje spletne učilnice za posamičen predmet ali celo skupino predmetov.
- Ob kreiranju spletne učilnice za posamičen predmet mora vmesnik omogočati dodajanje novih članov, dodajanje administrativnih pravic, spremembo imena skupine ali predmeta.
- Vmesnik spletne učilnice mora biti zgrajen iz več modulov, kot so: omogočanje objavljanja zapiskov, kreiranje anket, postavljanje vprašanj, odgovarjanje na vprašanja, objavljanje pomembnih rokov in klepetalnica.
- Vsak modul mora prikazati podatke, ki so povezani na specifičen predmet. Z vsemi podatki, ki krožijo po specifičnem razredu ali skupini, opravljajo ti moduli, posledično morajo biti podatki prikazani dinamično.
- Sistem mora omogočati sistem glasovanja. Sistem glasovanja se upošteva pri modulih, kot so: nalaganje zapiskov, postavljanje vprašanj, kreiranje anket in odgovarjanje na vprašanja. Uporabnik lahko glasuje, ali so mu zapiski, vprašanja ali odgovori koristili. Tukaj moramo upoštevati, da lahko uporabnik za specifično vsebino glasuje le enkrat.
- Vsi moduli morajo omogočati shranjevanje vsebine spletne učilnice v podatkovno bazo. Pri klepetalnici se morajo uporabljati socketi, ker gre za pogovor med uporabniki v realnem času.
- Ko uporabnik objavi vprašanje ali zapiske, morajo pri tem biti obveščeni vsi uporabniki.

3.2 Načrtovanje sistema

Po končani analizi uporabniških zahtev preidemo v fazo načrtovanja. Tukaj definiramo metode, tehnologije, diagrame, ki nam bodo pomagali pri razvoju sistema.
Za implementacijo sistema so nam na voljo naslednja orodja in programska okolja:

- Express
- MongoDB

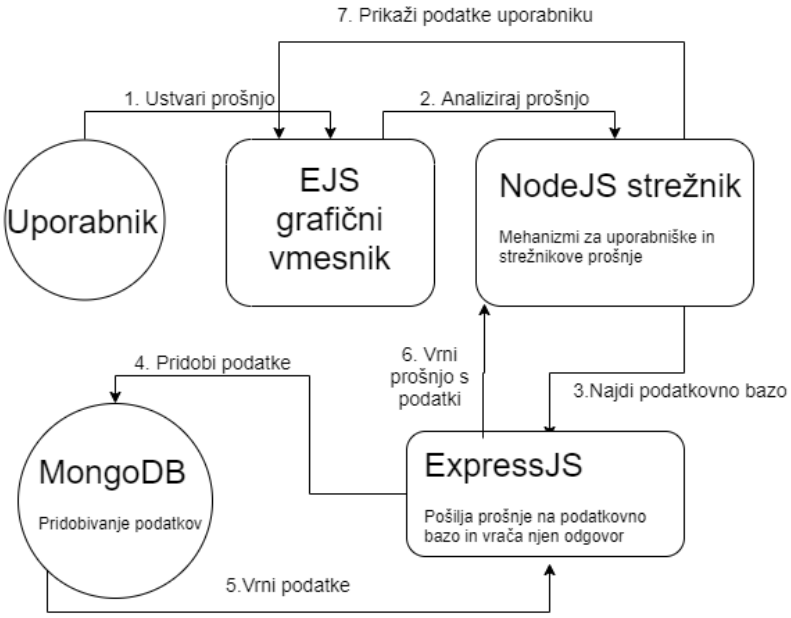
- NodeJS
- Visual studio code
- HTML[3]
- MongoAtlas
- EJS[2]
- Heroku
- CSS[4]
- Bootstrap

Temeljni sklepi:

- Spletna aplikacija je zgrajena iz dveh vmesnikov, in sicer domače strani, kjer se lahko uporabnik registrira in prijavi v sistem. Drugi vmesnik pa predstavlja pregled aktivnosti vseh učilnic, skupin. Vmesnik za pregled aktivnosti je zgrajen iz več modulov, ki nadzorujejo pretok podatkov in jih zapisujejo v podatkovno bazo.
- Ker podatkov ni potrebno procesirati in je projekt objektno orientiran, bomo uporabili objektno orientirano bazo MongoDB. Za komunikacijo med serverjem in podatkovno bazo bomo uporabili NodeJS.
- Način razvoja bo sprva potekal lokalno, ob uspešnem lokalnem testiranju bomo programsko opremo naložili na razvojno okolje Heroku, kjer bodo potekala še dodatna testiranja.
- Uporabnik lahko ustvari predmet ali pa številne predmete v skupini. Moduli, ki prikazujejo podatke specifičnega razreda, morajo upoštevati, ali se uporabnik nahaja v določeni skupini ali razredu, in mu tako prikazovati dinamične podatke glede na specifičen razred, ki se nahaja v skupini ali ne.
- Za dinamično predstavitev bomo uporabili EJS, ki nam omogoča uporabo if stavkov in for zank v HTML dokumentu.

3.2.1 Tok podatkov

Uporabnik naredi prošnjo na grafičnem vmesniku. Ta prošnja se analizira v Node.JS strežniku. Knjižnica *expressJS* izvede prošnjo na podatkovno bazo in pričakuje odgovor. MongoDB vrne podatke, ki jih zahteva prošnja. Express sporoči Node.JS strežniku, ali je prišlo do napake ali ne. To obvestilo nato prikažemo uporabniku v grafičnem vmesniku.



Slika 5: Tok podatkov v sistemu

3.3 Programske definicije

Sistem bo definiran s pomočjo naslednjih programskih jezikov:

- Javascript
- HTML
- CSS
- Mongoose
- Node.js
- Heroku
- EJS
- Bootstrap

HTML, CSS nam omogočata kreiranje statičnih spletnih aplikacij, vendar bo naš sistem dinamičen, to nam omogočata EJS in NodeJS. NodeJS nam omogoča, da enostavno kreiramo server lokalno na specifičnem portu našega računalnika s pomočjo paketa Express. EJS prikazuje spremenljivke, ki jih pošilja Node.JS server.

3.3.1 HTML

HTML je označevalni jezik za izdelavo spletnih strani. Komponente HTML, imenovane značke, predstavljajo ogrodje spletnega dokumenta. Posebnost tega jezika je, da se ga lahko izdela v vsakem urejevalniku besedila. Osnovni primer urejevalnika besedila je npr.

beležnica. Osnovni gradniki html jezika so značke, ki imajo obliko <*tag*> </*tag*>. Prvi znački rečemo začetna, drugi pa končna značka. Znotraj teh dveh značk lahko spletni oblikovalec doda besedilo in tudi druge značke. [5]

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

Slika 6: Primer ogrodja HTML dokumenta

Slika prikazuje glavne značke, ki sestavljajo ogrodje html dokumenta. V head znački določimo naslov dokumenta, nastavitve pogleda aplikacije. Ta del je namenjen uvažanju CSS dokumentov, fontov ali CDN-jev[5]. Body značke vsebujejo vsebino html dokumenta, to so tabele, paragrafi, seznami, slike itd. V tem delu po navadi tudi uvažamo skriptne datoteke ali njihove CDN-je.

3.3.2 CSS

CSS je jezik, s katerim oblikujemo značke html dokumenta. S pomočjo CSS jezika lahko določimo barvo, font, pozicijo, odmike, velikost, obrobe in druge atribute. HTML jezik nam omogoča dodajanje razredov ali identifikatorja posamezni znački, ki jih nato uporabimo v CSS podlogi, kjer definiramo oblikovna pravila za posamezno značko, razred ali identifikator. HTML po navadi razdelimo v sekcije in jih unikatno poimenujemo z identifikatorjem. Elementom znotraj sekcije po navadi dodamo razrede. CSS lahko povežemo s HTML dokumentom na tri načine, zaradi preglednosti pa se najbolj uporablja zunanji način povezave. [2]

```
.right-landing {
    position: absolute;
    width: 50%;
    height: 100%;
    top:0px;
    left: 50%;
    background: #00A6E0;
}
```

Slika 7: Primer oblikovanja elementa

3.3.3 Node.JS

Node.js je platforma, zgrajena na osnovi JavaScripta za enostaven razvoj hitrih, prilagodljivih aplikacij. Platforma je odprtokodna in jo lahko poganjamo na operacijskih sistemih OS X, Windows in Linux. Močna stran Node.js-a je, da omogoča široko izbiro knjižnic, ki nam olajša razvoj aplikacije. Ena izmed najbolj uporabljenih knjižnic je Express, s katero postavimo lokalni server. Node.js je tudi asinhron, če pogledamo primer API-jev[6], server nikoli ne čaka podatkov, ki jih ponuja API ampak gre na naslednjega. Platforma prikazuje podatke v kosih in jih ne shranjuje, prav tako uporablja eno nit in je visoko prilagodljiva tako za male kot za večje projekte. [10]

Slika 8: Primer vzpostavitve lokalnega serverja

Zgornja slika prikazuje vzpostavitev serverja s pomočjo knjižnice *express*. Tako lahko le v nekaj vrsticah naredimo svoj lokalni server, kjer bo »živela« naša aplikacija.

3.3.3.1 Node.JS knjižnice

Ena glavnih prednosti Node.JS so njegove knjižnice, ki so jih razvili inženirji po vsem svetu. To je lahko dvorezen meč, kajti včasih ni pametno razvijati projekta na osnovi knjižnic, ker postanemo odvisni od njih in v naslednjih posodobitvah porušijo naš projekt ali pa niso kompatibilne z drugimi knjižnicami. Zato se nekatera podjetja z resnimi projekti izogibajo uporabi knjižnic, ampak lete sprogramirajo sami. V primeru lažjih projektov se lahko poslužimo teh knjižnic, ki nam omogočajo hiter in enostaven razvoj. Tako lahko privarčujemo s časom in z denarjem.

3.3.4 MongoDB in Mongoose

MongoDB je objektno orientirana baza, kajti v njej imamo shranjene dokumente, ki so v obliki JSON[7]. V projektni arhitekturi načrtamo modele, ki so podlaga za objekte, ki jih potem shranjujemo v podatkovni bazi. MongoDB je imenovana tudi kot noSql podatkovna

baza in je zelo fleksibilna. Za komunikacijo med Node.js serverjem in MongoDB podatkovno bazo potrebujemo knjižnico z imenom *mongoose*. Gre za precej enostaven jezik, pri katerem lahko pišemo poizvedbe v podatkovno bazo. Pri tej podatkovno bazi se ne uporablja primarnih in tujih ključev, ampak indekse, ki se generirajo ob tvorbi dokumenta v zbirki.

```
const mongoose = require("mongoose");

const answersSchema = new mongoose.Schema({
    votes: {
        type: Number,
        required: true
    },
    voters: [
        { type: mongoose.Schema.Types.ObjectId, ref: 'Voter' }
    ],
    answerDescription: {
        type: String,
        required: true
    },
    author: {
        type: String,
        required: true
    },
    date: {
        type: Date,
        default: Date.now
    },
});

const Answer = mongoose.model("Answer", answersSchema);
module.exports = Answer;
```

Slika 9: Primer modela

Zgornja slika prikazuje temeljne atribute modela, ki ga uporabljamo skozi projekt in potem shranimo v podatkovno bazo. Gre za objekt, ki je sestavljen iz več atributov.

3.3.5 EJS in Bootstrap

EJS nam pomaga pri generiranju HTML elementov. Lahko uporabljamo tudi funkcije, ki so že vgrajene v JavaScriptu. Predvsem je pomemben za generiranje dinamične vsebine, ker lahko uporabljamo if stavke in for zanke. Prednosti tega jezika so, da je zgrajen na podlagi JavaScripta, kar pomeni, da gre za hiter jezik, ki je enostaven za odpravljanje napak. [4]

Druga zadeva, ki nam prav tako pomaga z oblikovanjem HTML elementov, je Bootstrap. Bootstrap velja za enega izmed najbolj priljubljenih HTML, CSS okvirjev. Bootstrap je definiran kot zbirka razredov, ki imajo definiran svoj stil. Te razrede nato povežemo na HTML element in tako dobimo preoblikovan element, ne da bi sami napisali CSS

oblikovanje za specifičen element. S pomočjo Bootstrapa lahko enostavno postavimo mrežo, na kateri bo ležala vsebina, in tudi, kako se bo prilagajala posamezni širini zaslona.

3.3.6 Heroku in MongoAtlas

Heroku je oblačna platforma, ki omogoča podjetjem, da zgradijo, nadzorujejo in prilagajajo strojno opremo aplikacije glede na promet aplikacije. Ni nam treba skrbeti za strojno opremo, ampak vzdržujemo samo programsko opremo. Glavni cilj oblačne platforme je, da se inženirji fokusirajo na razvoj, tako Heroku sam poskrbi za potencialne varnostne probleme. Primeren je za uporabo, kjer razvijamo projekte v ekipah, kajti Heroku omogoča, da se projekt loči na veje, vsi ukazi pa potekajo preko ukazne vrstice.

MongoAtlas je oblačna storitev za moderne aplikacije. Gre za ponudnika, kjer lahko gostujemo našo podatkovno bazo. Gre za robustno, visoko prilagodljivo platformo, in sicer ne glede na količino prometa. Samo ime pa pove, da gostujejo objektno orientirane dokumente.

3.4 Načrtovanje podatkovne baze in aplikacije

3.4.1 Načrt podatkovne baze

Pri načrtovanju aplikacije je pomembno, da natančno definiramo arhitekturo podatkovne baze in to, katere tabele oz. zbirke so povezane med sabo in kako. Če uporabljamo MongoDB podatkovno bazo, govorimo o shranjenih dokumentih v zbirkah. Gostovanje podatkovne baze nam bo omogočila oblačna platforma MongoAtlas, kjer kreiramo svoj cluster.

Temeljne zbirke:

- Vprašanja
- Predmeti
- Ankete
- Odgovori
- Skupine predmetov
- Datoteke
- Uporabniki
- Roki
- Volivec
- Klepetalnice
- Zapiski
- sporočilo

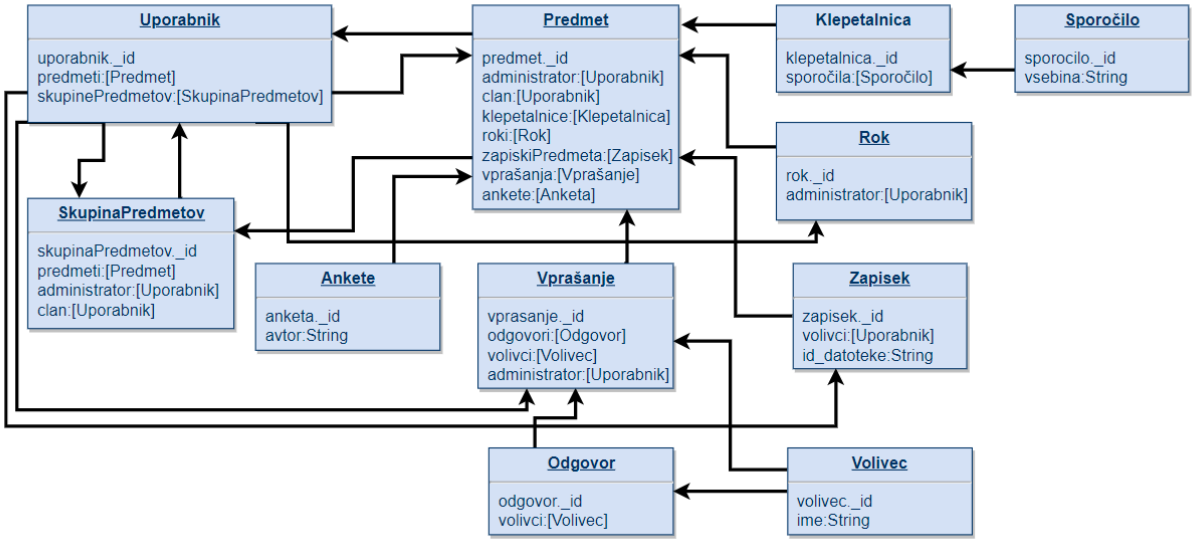
Zbirka lahko vsebuje tudi sklice na ostale objekte oz. zbirke. S pomočjo metode *populate()* lahko dobimo vse gnezdene podatke posamezne zbirke. V primeru SQL moramo dve tabeli združiti in nato filtrirati podatke, v primeru MongoDB podatkovne baze pa se uporabi metoda *populate()* za vse gnezdene zbirke posamezne zbirke.

V zbirki vprašanja se nahajajo vsa vprašanja specifičnega predmeta ali predmeta v skupini. Prav tako vsako vprašanje vsebuje reference dokumentov ostalih zbirk, kot so volivci in odgovori. Torej za vsako vprašanje lahko ugotovimo kateri odgovori so povezani na specifično vprašanje in kdo je že glasoval specifično vprašanje.

Predmet ima največjo zbirko. V njej lahko najdemo reference na več ostalih zbirk, to so npr. administratorji, udeleženci predmeta, klepetalnica, pomembni roki, zapiski, vprašanja in ankete. Tako lahko za specifičen predmet dobimo vse podatke, ki so povezani na specifičen predmet s specifičnim indeksom.

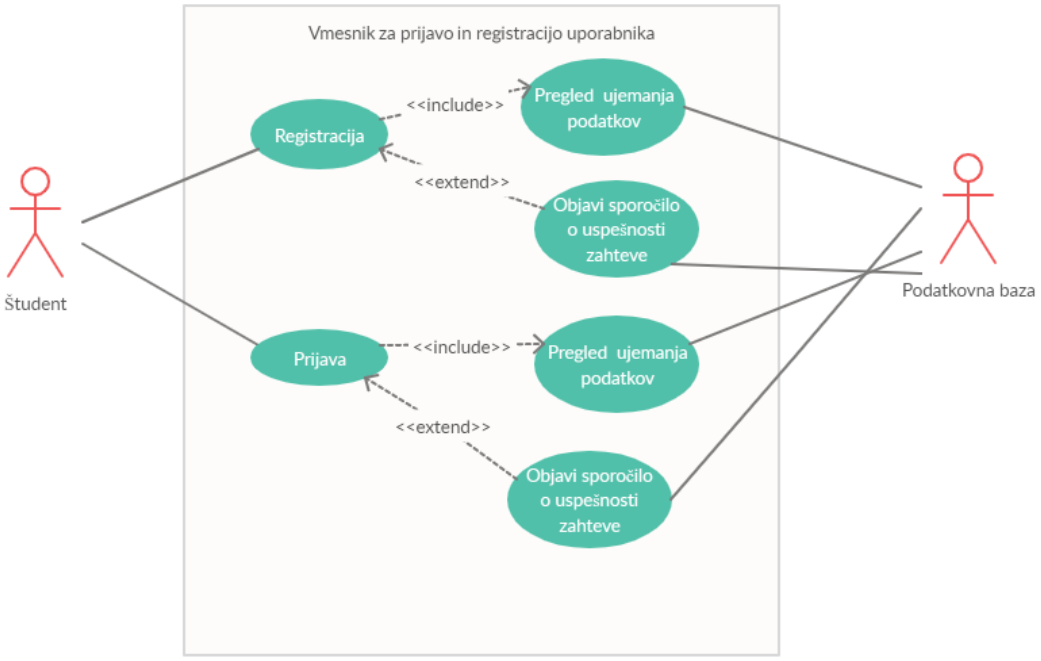
Zbirka zapiski vsebuje atribute, kot so avtor, ime datoteke, tip datoteke in referenca na zbirko volivci. Uporabnik, ki bo za določeno datoteko volil, ali mu je koristila ali ne, se bo povezal na to datoteko, tako bomo lahko v vsakem trenutku vedeli, kateri uporabnik je že volil in kateri ni.

Zbirka uporabnik vsebuje osnovne atribute, kot so e-naslov, ime, geslo in referenca na zbirko predmet in skupine predmetov. Tako točno vemo, v katerih predmetih ali skupinah predmetov uporabnik sodeluje.

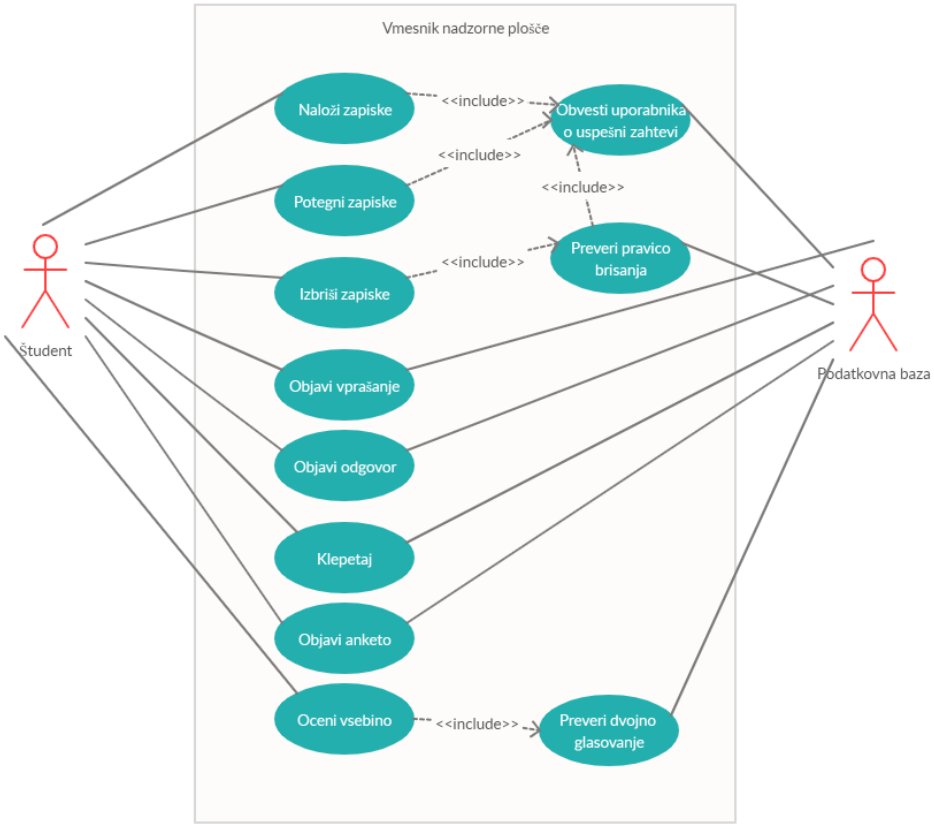


Slika 10: Načrt podatkovne baze

3.4.2 Načrtovanje vmesnikov spletne aplikacije



Slika 11: Vmesnik prijave in registracije



Slika 12: Vmesnik nadzorne plošče

3.4.2.1 Vmesnik za registracijo in prijavo

Grafični vmesnik domače strani omogoča prijavo in registracijo uporabnika. Uporabnik mora pri prijavi ali registraciji izpolniti vsa polja. Poslani podatki s forme na strežnik morajo biti skriti s POST metodo. Na strežnikovi strani moramo preveriti vse robne pogoje, ali je elektronski naslov že uporabljen, ali se vneseni gesli ujemata, ali se geslo sklada z geslom v podatkovni bazi itd. V primeru težav ali uspešne prijave ali registracije moramo uporabnika obvestiti, ali je bila njegova zahteva uspešna. Podrobnejši opis razvoja vmesnika je opisan v poglavju 4.1.1.

3.4.2.2 Vmesnik nadzorne plošče

Vmesnik nadzorne plošče nam omogoča pregled vseh operacij uporabnika v sistemu. Vsebino spreminjamo na podlagi izbranega predmeta ali skupine predmetov. Na predmet so vezani številni podatki, kot so klepetalnica, zapiski, vprašanja, odgovori na vprašanja in ankete. V sistemu je na voljo tudi glasovanje o vsebini, kot so: vprašanja, odgovori na vprašanja, zapiski in ankete. Vse glavne funkcionalnosti so porazdeljene v zavihke. Podatki, ki jih pošilja uporabnik na strežnik, se ustavijo v posameznem modulu, ta jih procesira in pošlje v podatkovno bazo. O tem, ali je zahteva uporabnika uspešna ali ne, ga obvestimo. Vsebino v posameznem predmetu lahko izbrišejo le administrator predmeta oz. skupine ali pa sam avtor vsebine. Podrobnejši razvoj vmesnika je opisan v poglavju 4.1.3.

4 PROTOTIP SISTEMA

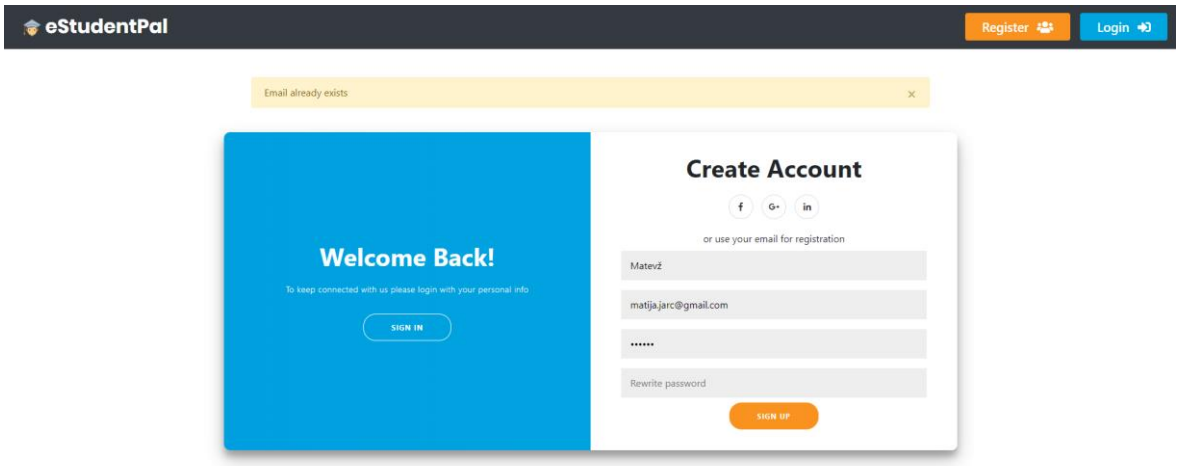
4.1 Vmesniki prototipa

Vsi vmesniki so sprogramirani v jeziku HTML, CSS, EJS in okvira Bootstrap. Vsak URL[8] naslov, ki ga lahko uporabnik obišče na spletni aplikaciji je definiran v Node.js datotekah. Ko uporabnik obišče specifičen URL naslov, se mu nato prikaže želena vsebina.

4.1.1 Vmesnik za registracijo uporabnika

Uporabnik lahko na domači strani spletne aplikacije opravi registracijo v sistem. Pri tem mora izpolniti nekaj osnovnih podatkov, kot so ime, elektronski naslov, geslo in ponovitveno geslo. HTML nam omogoča, da podatke zapisujemo v forme. V formah moramo definirati URL naslov, na katerega pošiljamo podatke. Prav tako moramo definirati tip metode, torej, ali bomo izbrali GET ali POST metodo. V tem primeru, gre namreč za občutljive podatke, uporabimo metodo POST, da se podatki ne prikazujejo v URL naslovu, tako pa preprečimo zlorabo podatkov. V Node.js definiramo poti, ki

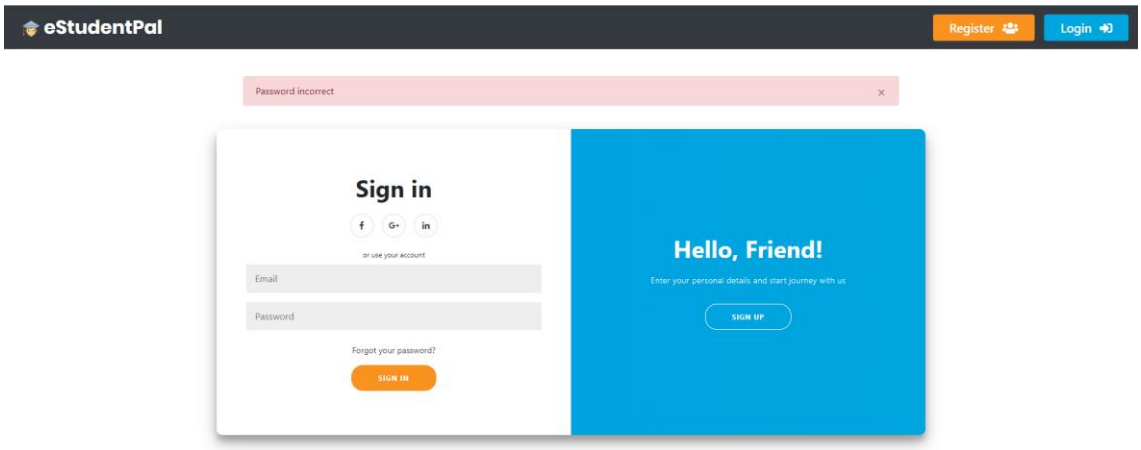
»poslušajo« na prihajajoče podatke. Sprva moramo preveriti vse robne pogoje, to so: ali so vsa polja izpolnjena; ali je elektronski naslov že uporabljen; ali se gesli ujemata. V primeru napake moramo uporabnika vrniti na vmesnik registracije in ga obvestiti o napakah, da jih v ponovnem poskusu ne ponovi. To jasno prikazuje spodnja slika. Node.js ima na voljo veliko knjižnic, zato za povratne informacije uporabniku uporabimo knjižnico *connect-flash.* V primeru, da uporabnik uspešno vnese pravilne podatke, jih moramo zavarovati. To naredimo s pomočjo knjižnice *bcryptjs.* Knjižnica nam omogoča, da gesla spremenimo v daljše binarne zapise s pomočjo matematičnih funkcij.



Slika 13: Vmesnik za registracijo

4.1.2 Vmesnik za prijavo uporabnika

Vmesnik za prijavo uporabnika ima enake mehanizme kot pri registraciji. Uporabnik izpolni polja v formi, ki nato prispejo na specifičen URL z metodo POST. Nato preverimo robne pogoje in v primeru napake obvestimo uporabnika. Edina razlika je pri uporabi knjižnice *bcryptjs.* Uporabnike iščemo v podatkovni bazi s pomočjo vpisanega elektronskega naslova. Ko dobimo specifičnega uporabnika, pogledamo njegovo shranjeno binarno geslo, ki ga dešifriramo s pomočjo *bcryptjs knjižnice.* Če se gesli ujemata, uporabnika shranimo v sejo in ga prijavimo v sistem. Uporabnik se lahko tudi odjavi v sistemu, posledično ga odstranimo iz seje, vsebina, ki potrebuje prijavo, pa mu ni več na voljo.

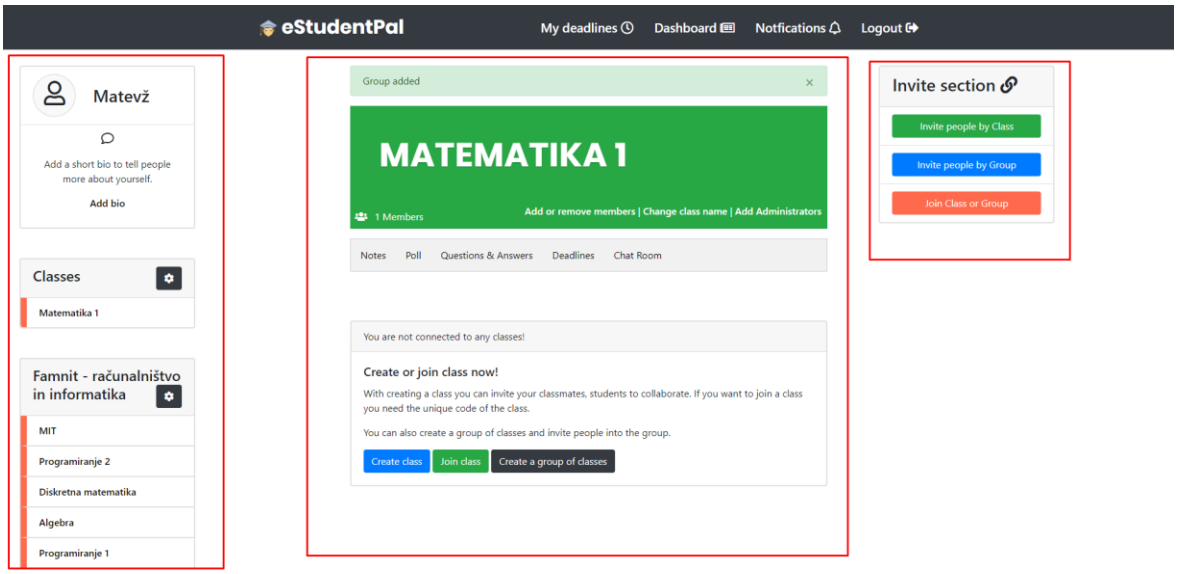


Slika 14: Vmesnik za prijavo

4.1.3 Vmesnik nadzorne plošče

Glavni vmesnik sistema je vmesnik nadzorne plošče, tukaj se vzpostavi socialno okolje za posameznega uporabnika. Vmesnik je oblikovan v mreži treh stolpcev, tako ga lahko razdelimo v tri komponente v smeri z leve proti desni (slika 9). Prva komponenta prikazuje osnovne informacije uporabnika, kot so ime, opis. V primeru, da je uporabnik prijavljen na specifičen predmet ali predmete v skupini, se mu prikažejo v prvi komponenti. S pomočjo nastavitvenega gumba se mu odpre okno, ki ga implementiramo s pomočjo okvirja Bootstrap. Okno nam nato ponuja številne funkcije, kot so: izbris predmeta, dodajanje predmeta, izbris skupine, dodajanje skupine, sprememba imena skupine. Če želimo spremeniti ime skupine, moramo imeti administrativno pravico.
Druga komponenta vsebuje največ funkcionalnosti, prikazuje ime predmeta in število članov. Študenti lahko v tej komponenti dodajajo zapiske, ankete, vprašanja, odgovore, roke ali uporabljajo javno klepetalnico. Za vse te glavne funkcije sem ustvaril posamezen modul, ki sprejema podatke, zapisuje podatke v podatkovno bazo in jih prikaže uporabniku. Vsebina v nadzorni plošči se spreminja in prikazuje dinamično, in sicer glede na indeks posameznega razreda. Vsak uporabnik vsebuje reference na posamezne skupine predmetov in predmete. Predmeti potem vsebujejo reference na vse ostale zbirke, ki prikazujejo vsebino, ki jo dodajo študenti. Za prikaz vsebine posameznega uporabnika moramo izvesti metodo *populate()*, ki je vgrajena v mongoose paket. To metodo izvedemo na zbirkah, ki so na enaki stopnji in na vseh gnezdenih zbirkah. Tretja komponenta prikazuje socialne gumbe, s katerimi lahko povabimo druge študente v naše skupine ali posamične predmete. Vsak predmet vsebuje unikaten ID, s pomočjo katerega ločimo razrede med sabo. Če se posameznik želi pridružiti, mu pošljemo ta unikaten ID. Uporabnik ID vstavi v polje in ga pošlje na strežnikovo stran, tam modul točno ve, katerega uporabnika mora dodati kateremu predmetu oz. skupini. Skupine in predmete ločimo med sabo. Skupine imajo prav tako unikaten ID. Ta ID je daljši zapis, ki ga dobimo

s pomočjo matematičnih funkcij. Možnosti, da bi lahko poljuben uporabnik ugotovil ta ID in se pridružil v skupino oz. predmet, je zelo malo.



Slika 15: Vmesnik nadzorne plošče

4.1.3.1 Nalaganje podatkov v podatkovno bazo

Pri nalaganju datotek v podatkovno bazo smo uporabili naslednje knjižnice: *multer, multer-gridfs-storage, gridfs-stream in crypto.* Na strani strežnika definiramo poti, ki »poslušajo« za nalaganje datotek, prenos datotek, izbris datotek in ocenjevanje datotek. Vsak predmet ima svojo unikatno vsebino in v tem primeru zapiske, zato moramo na definirane poti na strežnikovi strani pošiljati indeks predmeta, na katerem izvajamo številne operacije. Uporabnik lahko naloži v sistem eno datoteko na poskus. Če je datoteka uspešno naložena v podatkovno bazo, potem uporabnika obvestimo z obveščevalnim sporočilom, ki ga omogoča knjižnica *connect-flash.* Prav tako vse uporabnike, ki so prijavljeni na isti predmet, obvestimo, da je prišlo do nove aktivnosti. Ko so datoteke prikazane študentom, lahko študenti te datoteke prenesejo na svoje naprave s pomočjo gumba prenesi. Če je študent zadovoljen z vsebino datoteke, jo lahko oceni. Tako omogočamo, da datoteke s kvalitetno vsebino izstopajo. V primeru, da je študent naložil napačno datoteko, jo lahko izbriše v primeru, da je avtor ali administrator skupine oz. predmeta.

4.1.3.2 Tvorba ankete

Tvorba anket ima zopet svoj modul, kjer so definirane poti za številne operacije, kot so tvorba ankete, izbris ankete in glasovanje. Poleg ankete pošljemo indeks predmeta, v katerem se nahajamo. Uporabnik lahko izbriše anketo v primeru, da je avtor ali

administrator razreda oz. skupine. Sistem za glasovanje omogoča, da lahko v anketi glasujemo le enkrat.

4.1.3.3 Objavljanje vprašanj in odgovorov

Študentje lahko v sistem objavljajo vprašanja in potencialne rešitve za ostale študente. V tem modulu sta vključena dva modela, vprašanje in odgovor. Model vprašanje lahko vsebuje več objektov modela odgovor. Na strežnikovi strani so definirane poti, ki so zadolžene za naslednje procese: dodajanje vprašanj, dodajanje odgovora za specifično vprašanje, izbris vprašanja, izbris odgovora za specifično vprašanje, glasovanje za vprašanje, glasovanje za specifičen odgovor v vprašanju. Pri vsakem procesu pošiljamo indeks predmeta, v katerem se nahajamo. Pri določenih definiranih poteh potrebujemo še dodatne parametre, kot so indeks vprašanja in indeks odgovora. Tako lahko natančno posodobimo ali izbrišemo element, ki se nahaja v podatkovni bazi. V tem primeru lahko vidimo rabo zbirk ene v drugi. Eno vprašanje lahko ima več odgovorov, s pomočjo funkcije *populate()* pa tako prikažemo vprašanja in njihove odgovore. Vsako vprašanje in odgovor imata tudi prikazano število glasov. Vsebino lahko izbrišemo le v primeru, če smo avtor ali administrator predmeta oz. skupine. Ob dodani vsebini se obvesti vse člane, ki so prijavljeni v ta razred. Za prikaz dinamične vsebine smo uporabili EJS, uporaba je videna na sliki 10.

```
<%questions.forEach(function(question,key){%>
<div class="card mt-3">
    <div class="card-body">
        <div class="row">
            <div class="col-8">
                <p class="card-text">
                    <%=question.questionDescription%>
                </p>
                <p class="question-author-info">
                    Avtor:
                    <%=question.author%> (
                    <%=question.date%>)</p>

                <%if(isGroupClass){%>
                <form action="/dashboard/question/delete/group/<%=currentClassId%>/<%=userId%>/<%=userName%>/<%=question._id%>"
                    method="POST">
                    <button type="submit" class="noBtnStyle">Izbriši</button>
                </form>
                <%}else{%>
                <form action="/dashboard/question/delete/<%=currentClassId%>/<%=userId%>/<%=userName%>/<%=question._id%>"
                    method="POST">
                    <button type="submit" class="noBtnStyle">Izbriši</button>
                </form>
                <%}%>
            </div>
            <div class="col-4">
                <%if(isGroupClass){%>
                <div class="arrow-button text-center mb-2">
                    <form action="/dashboard/question/vote/up/group/<%=currentClassId%>/<%=userId%>/<%=userName%>/<%=question._id%>"
                        method="POST">
                        <button type="submit" class="noBtnStyle"><i class="fas fa-arrow-alt-circle-up"></i></button>
                    </form>
```

Slika 16: Uporaba EJS

EJS omogoča, da lahko uporabljamo if stavke in for zanke v HTML datoteki. Z njimi lahko dinamično predstavljamo vsebino. V zgornjem primeru lahko vidimo, da lahko s

pomočjo FOR zanke izvedemo številne iteracije skozi polje objektov vprašanje. Za posamezen objekt lahko nato izpišemo njegove atribute. Vsak objekt vsebuje tudi svoj unikaten indeks, ki ga lahko prenesemo v url forme. Tako lahko vsak trenutek vemo, kateri element poskušamo izbrisati ali posodobiti. Node.js ima definirane poti, nato s pomočjo vgrajene funkcije *render()* pošiljamo spremenljivke v HTML datoteko, ki so lahko različnega tipa. V zgornjem primeru lahko vidimo uporabo spremenljivke *isGroupClass*. Ta spremenljivka je tipa boolean, v tem primeru se uporablja v if stavku. Odvisno od vrednosti te spremenljivke se spreminja tudi struktura HTML elementov.

4.1.3.4 Dodajanje pomembnih rokov

Študentje lahko dodajajo tudi pomembne roke oz. datume za oddajo domačih nalog, prijave na izpit itd. V modulu na strežnikovi strani so definirane vse poti, ki so zadolžene za izvajanje operacij, kot sta dodajanje roka in izbris roka. Na vsako pot se pošljeta indeks predmeta in indeks roka v primeru izbrisa. V primeru dodajanja in izbrisa roka se obvesti vse uporabnike, ki so prijavljeni na ta predmet.

4.1.3.5 Klepetalnica

Ob kreiranju predmeta se generira unikatna klepetalnica. To pomeni, da pogovor, ki poteka znotraj tega predmeta, lahko vidijo le uporabniki, ki so prijavljeni na ta predmet.
V tem modulu potekajo mehanizmi programske opreme nekoliko drugače od prejšnjih. Ker gre za pogovor v realnem času, smo morali uporabiti sockete. Socket deluje kot omrežje, kjer se izmenjujejo podatki znotraj vozlišča. [9] Za implementacijo smo uporabili knjižnico *socket.io.* Za uspešen pogovor smo morali definirati »poti« na uporabniški strani in na strani strežnika. Na uporabniški strani smo morali definirati, kje gostujemo sistem in na katerem portu. Socketu moramo definirati tudi port, na katerem posluša za aktivnosti aplikacije.

```
// user connected and disconnected
socket.on('connection', function (socket) {
    console.log("new user connected");

    socket.on('send-chat-message', function (data) {
        console.log("received message: " + data);
        console.log(data);

        var chatMessage = data.msg;
        var chatRoomId = data.chatRoomId;
        var senderName = data.senderName;
        console.log("server chat room id: " + chatRoomId);
        Chat.create({ message: chatMessage, sender: senderName }, function (err, newChat) {
            if (!err) {
                ChatRoom.findById(chatRoomId, function (err, newChatRoom) {
                    if (!err) {
                        console.log("i found the chatroom: " + newChatRoom);
                        newChatRoom.chats.push(newChat);
                        newChatRoom.save(function (err) {
                            if (!err) {
                                socket.broadcast.emit('chat-message', data.msg);
                            }
                        });
                    }
                })
            }
        })
    });
});
```

Slika 17: Uporaba socketov

Ko uporabnik obišče specifičen URL na določenem portu, se poveže na vzpostavljen socket, v konzolo dobimo sporočilo, da se je pravkar prijavil nov uporabnik. Uporabnik na uporabniški strani izpolni polje, v katerega vstavi sporočilo. To polje je del forme s specifičnim indeksom. S pomočjo jezika JavaScript lahko dobimo vsebino tega polja, ki ima definiran indeks, in jo pošljemo na strežnikovo stran. Ta pot je definirana z imenom »send-chat-message«. Nato uporabimo model sporočilo in kreiramo objekt, ki ga pošljemo v podatkovno bazo. Vsebino, ki smo jo prejeli na strežnikovi strani s pomočjo funkcije *socket.broadcast.emit()*, razpršimo do ostalih uporabnikov. Tako poteka komunikacija v realnem času.

4.2 Podatkovna baza

Pri izdelavi podatkovne baze moramo paziti na poimenovanje zbirk in njenih atributov. Moramo si zastaviti neka slovnična pravila, ki se bodo skladala skozi celoten projekt zaradi preglednosti. Pri implementaciji zbirk in atributov se poskušamo izogibati šumnikov zaradi potencialnih napak. Struktura dokumentov v zbirki je v obliki JSON. Razlaga posameznih zbirk je opisana v tabeli 1.

Tabela 1: Slovar podatkovne baze

Zbirka	Razlaga	Atributi
Uporabnik	Zbirka Uporabnik je namenjena za kreacijo	**_id:** ObjectId **ime:** String

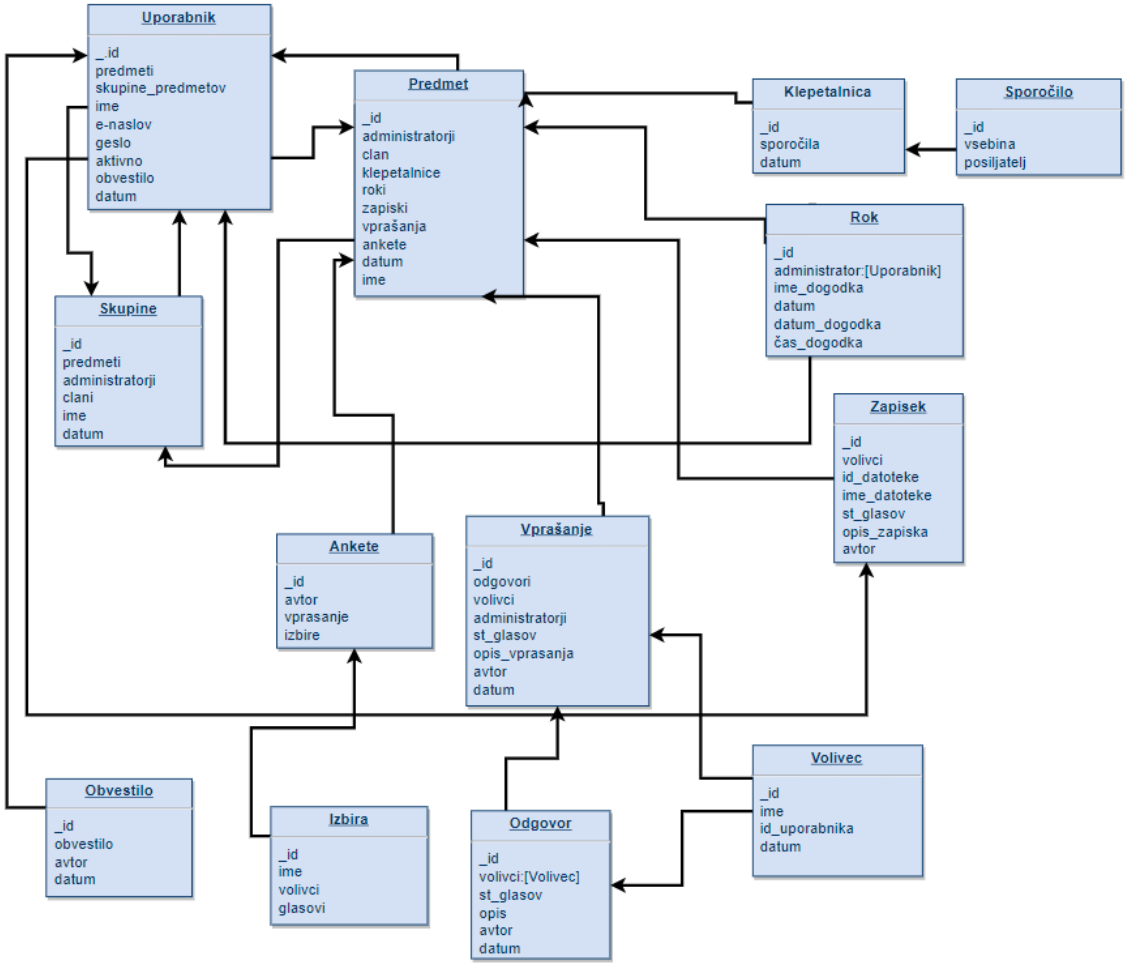
	novega računa in prijavo v sistem. Uporabniki se razlikujejo glede na gnezdene objekte predmetov in skupine predmetov. Vsakemu kreiranemu uporabniškemu objektu se doda unikaten ID, s pomočjo katerega ločimo uporabnike med sabo.	**e-naslov:** String **geslo:** String **aktivno:** String **predmeti: [**Predmet.Obj] **skupine_predmetov: [**Skupine.Obj] **obvestila:** [Obvestilo] **datum:** Date
Predmet	Zbirka je namenjena kreaciji dokumentov oz. objektov predmet. Predmet vsebuje osnovne atribute, kot sta ime in datum kreacije. Ta zbirka vsebuje največ gnezdenih zbirk. V atribut administrator shranjujemo vse uporabnike, ki imajo administratorske pravice. Zbirka predmet beleži tudi vse člane, ki so prijavljeni na ta predmet. Ostali atributi vsebujejo objekte, ki so pomembni za vsebino samega sistema. To so klepetalnica, pomembni roki, zapiski, vprašanja in ankete. Ob kreaciji objekta predmet se mu doda unikaten ID, s pomočjo katerega ločimo predmete med sabo.	**_id**: ObjectId **ime:** String **datum:** Date **administatorji: [**Uporabnik.Obj] **clani: [**Uporanbik.Obj] **klepetalnica: [**Klepetalnica.Obj] **pomembni_roki: [**Rok.Obj] **zapiski: [**Zapisek.Obj] **vprašanja: [**Vprašanje.Obj] **ankete: [**Anketa.Obj]
Skupine	Zbirka skupine vsebuje osnovne atribute, kot sta ime in datum kreacije. Ob kreaciji skupine moramo definirati tudi, katere predmete vsebuje. Temu	**_id:** ObjectId **ime:** String **datum:** Date **predmeti:** [Predmet.Obj] **administratorji:** [Uporabnik.Obj]

	služi atribut predmeti. Drugi atributi nam pomagajo pri definiciji kateri uporabnik je administrator in kateri člani so prijavljeni v kreirano skupino. Ob kreaciji dokumenta skupina se generira unikaten ID, s pomočjo katerega ločimo skupine med sabo.	**clani:** [Uporabnik.Obj]
Zapisek	Zbirka zapiski vsebuje atribut id_datoteke, to je unikaten zapis datoteke, ki ga dobimo od HTML forme. Prav tako beležimo kdo je avtor, opis zapiska, koliko glasov ima zapisek in kateri uporabniki so že volili vsebino. Ob kreaciji zapiska se generira unikaten ID dokumenta, s pomočjo katerega ločimo zapiske med sabo.	**_id:** ObjectId **id_datoteke:** String **ime_datoteke:** Strin **st_glasov:** Number **opis_zapiska:** String **avtor:** String **volivci:** [Volivec.Obj]
Volivec	Ob kreaciji dokumenta v zbirki volivec potrebujemo ID uporabnika, ki je pravkar opravil volitev in njegovo ime. Poleg tega zapišemo tudi datum kreacije dokumenta in generiramo unikaten ID dokumenta, s pomočjo katerega ločimo volivce med sabo.	**_id:** ObjectId **ime:** String **id_uporabnika:** String **datum:** Date
Vprašanje	Zbirka vprašanje vsebuje vsa vprašanja posameznega razreda. Vsebuje tudi število glasov, kdo je avtor vprašanja, datum kreacije, kateri uporabnik je volil	**_id:** ObjectId **st_glasov:** Number **opis_vprasanja:** String **avtor:** String **datum:** Date **volivci:** [Volivec.Obj]

	vsebino, kdo je administrator. Prav tako vsebuje vse odgovore na ta specifičen objekt. Ob kreaciji dokumenta vprašanje se generira unikaten ID, s pomočjo katerega ločimo vprašanja med sabo.	**odgovori:** [Odgovor.Obj] **administratorji:** [Uporabnik.Obj]
Odgovor	Zbirka odgovor vsebuje vse odgovore za specifična vprašanja. Vsak odgovor vsebuje število glasov, opis vsebine, avtorja in datum kreacije. Prav tako beležimo vse uporabnike, ki so volili specifičen odgovor. Ob kreaciji dokumenta v zbirki odgovor se generira unikaten ID, s pomočjo katerega ločimo odgovore med sabo.	**_id:** ObjectId **st_glasov:**Number **opis:** String **avtor:** String **datum:** Date **volivci:** [Volivec.Obj]
Klepetalnica	Zbirka klepetalnica shranjuje vse klepetalnice posameznega razreda ali skupine in njihova sporočila. Ob kreaciji se mu določi unikaten ID, s pomočjo katerega ločimo klepetalnice med sabo.	**_id:** ObjectId **datum:** Date **sporočila:** [Sporočilo.Obj]
Sporočilo	Zbirka sporočila vsebuje vsa sporočila posamezne klepetalnice. Njena dva atributa sta vsebina in pošiljatelj. Ob kreaciji sporočila se generira unikaten ID, s pomočjo katerega ločimo sporočila med sabo.	**_id:** ObjectId **vsebina:** String **posiljatelj:** String

Rok	Rok vsebuje vse pomembne dogodke posameznega razreda ali skupine. Dokument vsebuje datum in čas dogodka, datum kreacije, ime dogodka in kdo je administrator. Ob kreaciji roka, se generira unikaten ID s pomočjo katerega razlikujemo roke med sabo.	**_id:** ObjectId **ime_dogodka:** String **datum:** Date **datum_dogodka:** String **čas_dogodka:** String **administrator:** **[**Uporabnik.Obj]
Anketa	Zbirka anketa vsebuje vse ankete, ki so povezane na posamezen predmet. Vsebuje ime vprašanja, ime avtorja in vse izbire za katere lahko uporabniki glasujejo pri posamezni anketi. Ob kreaciji dokumenta v zbirki anketa se generira unikaten ID, s pomočjo katerega lahko ločimo ankete med sabo.	**_id:** ObjectId **vprasanje:** String **avtor:** String **izbire:** [Izbira.Obj]
Izbira	Zbirka izbira vsebuje vse izbire posamezne ankete posameznega razreda. Vsebuje ime, kdo vse je že volil to izbiro in število glasov. Ob kreaciji izbire se generira unikaten ID, s pomočjo katerega ločimo izbire med sabo.	**_id:** ObjectId **ime:** String **volivci:** [Volivec.Obj] **glasovi:** Number
Obvestilo	Zbirka obvestila vsebuje vsa obvestila za uporabnika o spremembah v določenem predmetu ali skupini predmetov. Ob kreaciji obvestila se generira unikaten ID, s pomočjo	**_id:**ObjectId, **obvestilo**: String, **avtor**:String **datum:** Date

	katerega lahko razločimo obvestila med sabo.	



Slika 18: Arhitektura podatkovne baze

Gostovanje podatkovne baze omogoča oblačna platforma MongoAtlas, ki ponuja brezplačno uporabe do 500 MB podatkov. Ustvarili smo svoj cluster v srednji Evropi, platforma generira unikaten link z vgrajenim geslom uporabnika, ki ga nato uporabimo pri povezavi na strežnikovi strani, kjer poteka pretok informacij od uporabniškega vmesnika v podatkovno bazo. Cluster je mreža računalnikov, ki sodelujejo skupaj in predstavljajo enoten sistem za izvajanje specifičnih nalog. [1]

4.3 Uporabljene knjižnice

Za razvoj sistema smo uporabili 19 knjižnic, kar nam je omogočalo hitrejši in enostaven razvoj. V tem poglavju bomo opisali nekaj glavnih knjižnic. Vse knjižnice se shranijo v

datoteko, imenovano package.json, tako lahko sistem prenesemo na katerokoli napravo in izvedemo ukaz *npm install*, ki naloži vse knjižnice na napravo.

4.3.1 Express

Express je ena od najbolj uporabljenih knjižnic pri razvoju Node.js aplikacij. V le nekaj vrsticah lahko generiramo lokalni server aplikacije. Je zelo prilagodljiva in minimalistična knjižnica. Uporaba nam omogoča implementacijo robustnih poti oz. URL naslovov, na katere »posluša aplikacija«. Prav tako nam omogoča posredovanje podatkov preko http klicev. Podatke lahko dobimo preko url naslova ali preko forme v HTML dokumentu. Ko podatki pridejo na definirano pot, lahko v tej poti izvedemo številne funkcije in operacije. Nato lahko posodobljene podatke dinamično prikažemo s funkcijo *res.render()*, v katero lahko vstavimo objekt, ki vsebuje posredovane spremenljivke, ki nosijo poljubne vrednosti. [11]

4.3.2 Connect-flash

Pri razvoju projekta je pomembno, da uporabniku pošiljamo povratne informacije, če se je proces, ki ga je izvajal končal, ali je prišlo do napake. To obveščanje omogoča knjižnica connect-flash, ki deluje skupaj s knjižnico express. Preden uporabimo funkcijo *redirect()*, zapišemo sporočilo v poseben del seje. Ko se grafični vmesnik uspešno naloži se sporočilo prebere in izbriše iz seje. Tako uporabniku povemo, ali je bila operacija uspešna ali ne.

4.3.3 Method-override

Node.js omogoča številne tipe metod pri definiranih poteh Node.js aplikacije. Te metode so POST, GET, PUT, DELETE. Knjižnica *method-override* mi omogoča, da lahko pošiljamo podatke preko HTML forme na pot, ki je definirana z metodo DELETE. Knjižnica služi, da je projekt bolj pregleden in da točno vemo, kaj katera pot počne, ali poskušamo pridobiti informacije iz podatkovne baze, ali poskušamo izbrisati podatke iz podatkovne bazi, ali poskušamo posodobiti podatke v podatkovni bazi in ali želimo naložiti podatke v podatkovno bazo.

4.3.4 Passport

Passport knjižnica je ena najbolj uporabljenih knjižnic za avtentikacijo uporabnika v sistem. Je fleksibilna knjižnica, kjer lahko definiramo mnogo strategij, preko katerih lahko opravljamo proces registracije in prijave. Za registracijo in prijavo lahko uporabljamo socialne platforme, kot so Google plus, Facebook, Twitter, ki jih definiramo v svojih strategijah. Ko se uporabnik prijavi v sistem, shranimo njegov unikaten ID v sejo. Ko

uporabnik izvaja operacije v sistemu, pride do številnih obiskov poti na strežnikovi strani, kjer uporabimo njegov ID tako, da v vsakem trenutku vemo, kateri uporabnik izvaja specifične operacije.

4.3.5 Mongoose

Uporaba knjižnice Mongoose nam omogoča, da se povežemo na podatkovno bazo. Gre za orodje, ki nam pomaga pri modeliranju MongoDB objektov, ki so v JSON obliki. Ukazi se izvajajo asinhrono. Pri projektu smo definirali številne modele oz. sheme, in sicer, kakšne atribute mora vsebovati specifičen objekt. Pri tem je pomagala Mongoose knjižnica. Mongoose omogoča eleganten način poizvedovanja podatkov iz podatkovne baze. Vsak klic poizvedovanja vsebuje povratno funkcijo, kjer preverimo, ali je prišlo do napake. [8]

5 ZAKLJUČEK

Cilj zaključne naloge je bil opisati načrtovanje in razvoj prototipa sistema za študentsko medsebojno pomoč, ki bi služil kot zbirališče vseh študentov. Upoštevati smo morali definirane zahteve in diagrame za načrtovanje in implementacijo sistema.

Z implementacijo prototipa smo omogočili študentom, da lahko izvajajo številne funkcije na enem mestu kot so: deljenje znanja, postavljanje vprašanj, postavljanje anket, uporaba javne klepetalnice predmeta in nalaganje zapiskov. Uspešno smo upoštevali vse uporabniške zahteve in jih implementirali v svoj prototip. Študentom sedaj ni treba uporabljati številnih platform za doseganje enakih ciljev. Za kreiranje študentskega okolja za posamezno skupino študentov ni treba pridobiti priznanja fakultete, kajti to lahko stori posameznik sam.

Sam prototip služi kot nadgradnja sedanjih elektronskih učilnic posameznih fakultet. V prototipu lahko študentje izvajajo operacije, ki jih potrebujejo in jih izvajajo zunaj nadzora fakultete. Za izvajanje takšnih operacij so se morali študentje poslužiti številnih platform. Prototip omogoča, da lahko vse izvajajo v eni platformi.

Sistem ima še veliko prostora za nadgradnjo. Za obveščanje uporabnikov, bi lahko uporabili paket *socket.io,* tako bi namreč obveščanje potekalo v realnem času. Prav tako bi sistem lahko omogočal zaseben pogovor med dvema uporabnikoma. Sam prototip še ni bil testiran, treba bi bilo dodatno testiranje programske opreme, da ne bi prišlo do večjih napak pri testiranju prototipa.

6 LITERATURA IN VIRI

[1] Computer cluster (8. Julij 2019). *Wikipedija prosta enciklopedija.* Dostopno 07.08.2019 na internetu
https://en.wikipedia.org/wiki/Computer_cluster

[2] CSS (11. April 2017). *Wikipedija prosta enciklopedija.* Dostopno 25.07.2019 na internetu
https://sl.wikipedia.org/wiki/CSS

[3] Dinesh, Thakur. What is build and fix model or ad hoc model? And Explain its Advantages and Disadvantages. *Ecomputer notes.* Dostopno 11. 07. 2019 na internetu: http://ecomputernotes.com/software-engineering/build-and-fix-model

[4] Embedded JavaScript templating. *EJS*. Dostopno 26. 07. 2019 na internetu:
https://ejs.co/

[5] HTML (16. Januar 2017). *Wikipedija prosta enciklopedija.* Dostopno 25.07.2019 na internetu
https://sl.wikipedia.org/wiki/HTML

[6] Linear sequential model/waterfall model/classic life cycle (22. Januar 2019). *Ed technology.* Dostopno 12. 07. 2019 na internetu:
http://www.edtechnology.in/software-engineering/linear-sequential-model-waterfall-model-classic-life-cycle/

[7] Meenakshi, Agarwai. SDLC and Seven SDLC Phases In a Nutshell. *TechBeamers.* Dostopno 27. 07. 2019 na internetu:
https://www.techbeamers.com/software-development-life-cycle-sdlc/

[8] Mongoose. *NPM.* Dostopno 26. 07. 2019 na internetu:
https://www.npmjs.com/package/mongoose

[9] Network socket (5. Avgust 2019). *Wikipedija prosta enciklopedija.* Dostopno 07.08.2019 na internetu
https://en.wikipedia.org/wiki/Network_socket

[10] Node.js - Introduction. *TutorialsPoint.* Dostopno 25. 07. 2019 na internetu:
https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

[11] Node.js – Express Framework. *TutorialsPoint.* Dostopno 26. 07. 2019 na internetu: https://www.tutorialspoint.com/nodejs/nodejs_express_framework

[12] 5 Popular software development models with their pros and cons (5. Oktober 2017). *GlobalLuxSoft.* Dostopno 12. 07. 2019 na internetu: https://medium.com/globalluxsoft/5-popular-software-development-models-with-their-pros-and-cons-12a486b569dc

[13] SDLC – V-Model. *TutorialsPoint.* Dotstopno 14.07.2019 na internetu: https://www.tutorialspoint.com/sdlc/sdlc_v_model

[14] SDLC – waterfall model. *TutorialsPoint.* Dostopno 14. 07. 2019 na internetu: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model

[15] Software development life cycle (part 5). *Qastation.* Dotstopno 14.07.2019 na internetu: https://qastation.wordpress.com/tag/multi-waterfall/

[16] Spiral model: advantages and disadvantages. *Idconline*. Dostopno 24.07.2019 na internetu: http://www.idc-online.com/technical_references/pdfs/data_communications/Spiral_Model_Advantages_and_Disadvantages.pdf

[17] Spiralni razvojni model. *eSistemi.* Dostopno 24.07.2019 na internetu: http://esistemi.si/spiralni-razvojni-model

[18] Sukanović, Elvis (december, 2012). Razvoj računalniške aplikacije za izračun predstavitev stroškov porabe različnih energentov skozi vse faze razvoja. Dostopno 07.08.2019 na internetu: https://www.famnit.upr.si/sl/studij/zakljucna_dela/view/30

[19] Swersky, Dave (31. Maj 2018). The SDLC: 7 phases, popular models, befits & more [2019]. *Raygun.* Dostopno 11.07.2019 na internetu: https://raygun.com/blog/software-development-life-cycle/#what-is-the-sdlc

[20] The Software Development Life Cycle (SDLC). *Pelican engineering*. Dostopno 14.07.2019 na internetu: http://www.pelicaneng.com/DevDocs/sdlc.pdf

[21] The seven phases of the system-development life cycle. *Innovative architecs.* Dostopno 11. 07. 2019 na internetu: https://www.innovativearchitects.com/KnowledgeCenter/basic-IT-systems/system-development-life-cycle.aspx

[22] What is prototype model – advantages, disadvantages and when to use it. *TRY QA.* Dostopno 14. 07. 2019 na internetu: http://tryqa.com/what-is-prototype-model-advantages-disadvantages-and-when-to-use-it/

[23] What is waterfall model – examples, advantages, disadvantages & when to use it. *TRY QA.* Dostopno 14. 07. 2019 na internetu: http://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/