Razvoj softvera otvorenog koda

Ritoša, Matej

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matej Ritoša

RAZVOJ SOFTVERA OTVORENOG KODA

ZAVRŠI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matej Ritoša

Matični broj: 0016150816

Studij: Informacijski i poslovni sustavi

RAZVOJ SOFTVERA OTVORENOG KODA

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Marko Mijač

Varaždin, rujan 2024.

Matej Ritoša

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Razvoj softvera otvorenog kôda je proces izgradnje softvera čiji je izvorni kôd javno dostupan za korištenje, proučavanje, nadogradnju i unaprjeđenje. U teorijskom dijelu ovog rada će biti opisana osnovna motivacija koja stoji iza otvorenog kôda, licence, modeli razvojnog procesa, korišteni alati, prednosti i nedostaci te načini doprinosa i kolaboracije na projektima otvorenog kôda. Nakon uvodnog dijela o osnovnim činjenicama, prijeći će se na praktični dio u kojem će biti prikazan vlastiti doprinos odabranom softveru otvorenog kôda.

Ključne riječi: softver otvorenog kôda, razvoj softvera, licence, alati za razvoj, kolaboracija

Sadržaj

Sadržaj iii
1. Uvod 1
2. Softver otvorenog kôda 2
2.1. Povijest softvera otvorenog kôda 2
2.2. Motivacija iza otvorenog kôda 3
2.3. Prednosti i nedostaci 4
2.4. Primjeri popularnih softvera otvorenog kôda 6
2.4.1. Linux 6
2.4.2. Git 6
2.4.3. Mozilla Firefox 7
2.4.4. Visual Studio Code 7
2.5. Licence 7
3. Razvoj softvera otvorenog kôda 11
3.1. Usporedba životnog ciklusa otvorenog i zatvorenog kôda 11
3.2. Razvojni modeli 13
3.3. Korišteni alati 16
3.3.1. Alati za verzioniranje 16
3.3.2. Alati za praćenje grešaka 17
3.3.3. Platforme za razvoj softvera 18
3.4. Ostali alati 19
3.5. Načini doprinosa i kolaboracije 19
3.6. Fork & Pull model 20
3.7. Recenzija koda 21
3.8. Uloge u razvoju softvera otvorenog kôda 22
4. Primjer doprinosa softveru otvorenog koda 23

4.1. Odabrani softver 23
4.2. Doprinos softveru 25
4.2.1. Pravila doprinosa 25
4.2.2. Postavljanje projekta 25
4.2.3. Opis problema 27
4.2.4. Repliciranje problema 28
4.2.5. Rješavanje problema 30
4.2.6. Predaja rješenja 37
5. Zaključak 40
Popis literature 41
Popis slika 44
Popis tablica 45

1. Uvod

U današnje vrijeme razvoj softvera otvorenog koda igra važnu ulogu u softverskoj industriji. Ovaj inovativni pristup stvaranju softvera je revolucionarizirao način na koji se softver dijeli i poboljšava te je tako promijenio našu interakciju s tehnologijom i percepciju tehnologije. Softver otvorenog koda služi kao motivator za inovacije promičući suradnju, poboljšavajući kvalitetu softverskih rješenja i povećavajući njihovu fleksibilnost.

Ovaj će završni rad pokazati razlog sudjelovanja u projektima otvorenog kôda te s time i njegove prednosti i nedostatke. Uz to bit će opisana glavna ideja koja stoji iza otvorenog kôda i kako je ona započela. Kako bi se istakla veličina projekata otvorenog kôda spomenut će se neki od najpopularnijih softvera otvorenog kôda koji su doveli revolucionarne promjene u tehnološkoj industriji. Uvidom u razvojne modele, alate, i načine kolaboracije na projektima dati će se bolji pregled razvojnog procesa.

Osim teorijskog dijela, ovaj rad će se proširiti na praktični dio gdje će biti prikazan konkretni doprinos odabranom softveru gdje će biti prikazan detaljan opis odabranog softvera, problem koji će se rješavati, rješenje problema te njegova predaja. Glavna svrha ovog praktičnog dijela je pružiti bolje razumijevanje procesa razvoja otvorenog kôda te unaprijediti odabrani softver.

U konačnici, ovaj rad ima za cilj ne samo teorijski i praktično pokazati ključne aspekte razvoja softvera otvorenog kôda, već i istaknuti važnost suradnje i dijeljenja u procesu stvaranja visokokvalitetnih digitalnih rješenja. Kroz teoriju i praksu, naglasit će se da je razvoj softvera otvorenog kôda postao neizostavni dio u industriji razvoja softverskih rješenja.

2. Softver otvorenog kôda

Softver otvorenog kôda, poznat i kao Open Source Software (OSS), obuhvaća softverske sustave koji nude slobodan pristup korisnicima te im omogućuju da pregledavaju i mijenjaju izvorni kôd. Ovaj pristup ne samo da angažira programere, već i sve korisnike koji imaju potencijal postati programeri, omogućujući im uvid u unutarnje funkcije softvera i doprinos razvoju zajednice koja neprestano raste i poboljšava se.

2.1. Povijest softvera otvorenog kôda

Početak koncepta otvorenog softvera započinje 1960-ih u vrijeme kulture hakera, gdje se pojam haker tada nije povezivao s nekim ilegalnim radnjama već s eksperimentiranjem i petljanjem na nečemu iz znatiželje i s ciljem da se nauči nešto novo. Cijela ta kultura hakera je započela na MIT-u (*Massachusetts Institute of Technology*) gdje se je u njihovim studentskim zajednicama eksperimentiralo s različitim tehnologijama s idejom da bi sve informacije trebale biti besplatne.

Ideja za izgradnjom softvera otvorenog kôda, tj. tada zvanog slobodnog softvera konkretizirala se 1982.g kada je MIT-ev laboratorij za umjetnu inteligenciju kupio printer čije je licenca zabranjivala bilo kakve modifikacije na softveru. Richard Stallman koji je bio član tog laboratorija to nije mogao prihvatiti te je odlučio izgraditi vlastiti sustav koji se temeljio na slobodnom kôdu. Tako je 1983.g razvijen prvi softver otvorenog kôda zvan GNU [1, str. 2]. Dvije godine kasnije osnovan je „*Free Software Foundation*“ čija je uloga bila podupiranje GNU projekta i svih drugi projekata otvorenog kôda. Zatim, 1989. je kreirana prva licenca zvana GNU General Public Licence(GPL) koja je dala slobodu programerima da slobodno koriste i izmjenjuju softver.

Koncept slobodnog softvera motivirao je programere diljem svijeta, te je tako finac Linus Torvalds pokrenuo projekt zasnovan na slobodnom softveru koji je tako postao jedan od najpoznatijih operacijskih sustava, Linux. Njegovim uspjehom su bili potaknuti mnogi radovi te je tako 1997. godine napisan prvi esej, a potom 2001. i knjiga koja se bavila konceptom softvera otvorenog kôda zvana „The Cathedral and the Bazaar“ čiji je autor Eric Raymond. U knjizi je otvoreni kôd opisan kao bazar na kojemu programeri dijele svoja znanja i vještine te ih koriste za izgradnju softvera. Raymondova perspektiva o slobodnom softveru naglašava praktične prednosti otvorenog sudjelovanja u razvoju softvera što je potaknulo tvrtku Netscape 1998. godine da objavi izvorni kôd svojeg preglednika kako bi surađivao s pokretom slobodnog softvera što je kasnije rezultiralo i stvaranjem web preglednika Firefox.

Iako je softver s izvornim kôdom dostupnim svima tada bio poznatiji pod nazivom „slobodni softver“, često je bio krivo shvaćen kao besplatan softver. Iz tog je razloga naziv promijenjen u „softver otvorenog kôda“ koji je kasnije prihvaćen od ostalih članova pokreta otvorenog kôda.

2.2. Motivacija iza otvorenog kôda

Motivacija iza sudjelovanja u projektima otvorenog kôda proizlazi iz različitih izvora. Pošto je otvoreni kôd danas ključan dio u razvoju softvera važno je razumjeti i glavne motivatore za sudjelovanje i pokretanje takvih projekata. Neki od tih motivatora su intrinzični, proizlazeći iz samog zadovoljstva sudjelovanja i stvaranja, dok su drugi ekstrinzični, povezani s vanjskim nagradama ili koristima poput karijernog napretka ili poboljšanja vlastitih vještina.

Postoje mnogi faktori koji motiviraju programere za sudjelovanjem u razvoju softvera otvorenog kôda. Ti faktori mogu biti kognitivni i socijalni. Kognitivna motivacija ili bolje rečeno motivacija za samounapređenjem je najčešća. Ona je najjača kada zajednica prizna i nagrađuje doprinos svojih članova te s time se i povećava broj korisnika u projektima otvorenoga kôda [2]. Nju karakterizira to što potiče programera da nauči neku novu vještinu, razumije na koji način funkcionira kôd, pomogne drugim programerima tako da dijeli svoje ideje i stečeno znanje. Sve to su veliki motivatori koji potiču programere za sudjelovanjem u razvoju otvorenog kôda.

Softverski sustavi postaju sve kompleksniji što zahtjeva veliku kreativnost i širok repertoar znanja u području informacijske tehnologije kako bi se izgradili. Prema tome takvi javno dostupni softverski sustavi predstavljaju programerima bazu znanja koja ih potiče za sudjelovanjem i učenjem [2].

Kao što je već spomenuto, softver otvorenog kôda ima nekoliko istaknutih karakteristika. Dostupan je svima i omogućuje korisnicima da proučavaju izvorni kôd. Potiče programere da razmjenjuju svoje ideje i znanja te također otvara vrata za poboljšanja, ne samo softvera već i programerovih vještina. Ovakva je izloženost jako motivirajuća za sve korisnike otvorenog kôda jer potiče učenje i usavršavanje vještina u programiranju. Učenje unutar okruženja otvorenog izvornog kôda nije samo proučavanje kôda drugih, već to također uključuje rad sa zajednicom, dijeljenje mišljenja s kolegama programerima, međusobno podržavanje i stjecanje znanja jedni od drugih. Ova vrsta rada omogućuje programerima ne samo poboljšavanje sposobnosti programiranja, već i da razviju vještine poput timskog rada i učinkovite komunikacije.

Osim samounaprjeđenja mnogi programeri otvoreni kôd koriste kao priliku za isticanjem i poboljšavanjem svoje reputacije u zajednici. Tako mogu pokazati većim poslodavcima svoje znanje i vještine te izboriti se za bolju radnu poziciju.

Tablica 1: Motivacija iza otvorenog kôda

Kategorija	Motivacija
Intrinzične	Zadovoljstvo stvaranja
	Učenje novih vještina
	Rješavanje problema
	Povećanje kreativnosti
Ekstrinzične	Karijerni napredak
	Reputacija u zajednici
	Umrežavanje
	Financijske nagrade
Kognitivni	Samounapređenje
	Priznanje i nagrade
	Proučavanje izvornog kôda
Socijalni	Rad u zajednici
	Podrška i mentorstvo
	Timski rad i komunikacija

2.3. Prednosti i nedostaci

Softver otvorenog kôda ima mnoge prednosti i nedostatke koje je bitno uzeti u obzir prilikom sudjelovanja takvim projektima.

Jedna od glavnih prednosti otvorenog kôda je to što je besplatan, što značajno smanjuje troškove korisnika softvera. Uz to, njegova fleksibilnost omogućuje korisnicima da izmjenjuju softver prema njihovim željama i potrebama. Također, dostupnost softvera omogućuje rano uočavanje i ispravljanje grešaka, čime dolazi do veće kvalitete i zadovoljstva korisnika. Kolaboracija između sudionika rezultira širenjem znanja i novih ideja koje su ključne za napredak projekata. Velika zajednica omogućuje korisnicima da brzo dobiju pomoć i savjete

od iskusnijih programera. Isto tako veliki broj sudionika na projektima rezultira bržim pronalaskom sigurnosnih propusta. Decentralizirana organizacija potiče različite pristupe i inovativna rješenja, čime se dodatno povećava učinkovitost razvoja softvera. Uz dijeljenje znanja, sudionici u projektima otvorenog kôda mogu istaknuti svoje vještine i steći priznanje unutar zajednice, što im može pomoći u daljnjem razvoju karijere.

S druge strane neki od nedostataka softvera otvorenog kôda bi bila otežana organizacija zbog velikog broja sudionika u projektima. Sudionici ponekad mogu namjerno prijavljivati lažne nedostatke, što bi moglo rezultirati kočenjem razvojnog procesa. Isto tako, pošto se promjene stalno dešavaju može doći do nestabilnosti verzija i lošije dokumentacije koja ovisi uglavnom o zajednici. Uz sve to sudionici koji nisu kvalificirani za rad na projektima otvorenog kôda mogu znatno usporiti razvojni proces [3]. Isto tako pošto je zajednica u projektima otvorenog kôda jako velika mnogi programeri koji se prvi put pronalaze u takvim okruženjima mogu osjetiti nedostatak podrške od zajednice. Mnogi iskusniji članovi možda neće imati vremena posvetiti se početnicima što može biti jako obeshrabrujuće i demotivirajuće za nove sudionike. Osim toga jedan od većih nedostataka je to što je otvoreni kôd dostupan svima što može potaknuti mnoge da iskoriste to kako bi naštetili projekti i profitirali na temelju toga.

Tablica 2: Prednosti i nedostaci softvera otvorenog koda

Prednosti	Nedostaci
Softver je besplatan	Otežana organizacija
Fleksibilnost	Loša dokumentacija
Kvaliteta	Nedostatak podrške
Brzo uočavanje i ispravljanje grešaka	Nestabilne verzije zbog stalnih promjena
Kolaboracija	Nekvalificirani sudionici
Sigurnost	Otvoren za napade
Učenje	
Decentraliziranost	

2.4. Primjeri popularnih softvera otvorenog kôda

Postoji mnogo primjera softvera otvorenog kôda koji su ostavili veliki trag u suvremenom računalnom dobu pružajući alate za razna područja i svrhe. U ovom će poglavlju biti prikazani neki od najpopularnijih softvera otvorenog kôda.

2.4.1. Linux

Linux je jedan od najpoznatijih operacijskih sustava otvorenog kôda. Dobio je ime po jednom od začetnika pokreta otvorenog kôda, Linusu Torvaldsu, 1991. godine. Temelji se na Linux kernelu koji predstavlja srž operacijskog sustava. Uloga kernela je uspostava komunikacije hardverskog i softverskog dijela sustava te osiguravanje temeljnih funkcionalnosti operacijskog sustava. Osim kernela, operacijski sustav Linux sadrži mnoge druge programe otvorenog kôda koji ga grade i čine korisnim. Svi ti programi su temeljeni na GPL(General Public Licence) licenci koja osigurava da sav razvijeni softver bude otvorenog kôda [4], [5].

Jedna od glavnih karakteristika Linuxa su njegove distribucije. Linux distribucije su operacijski sustavi temeljeni na Linux kernelu koje su razvijene prema potrebama korisnika. Neke od tih distribucija su razvijene za jednostavniju upotrebu, poput Ubuntua ili Linux Minta. Dok s druge strane BlackBox i BlackArch predstavljaju distribucije koje se koriste za penetracijska testiranja. Iz toga se može zaključiti kako je Linuxova otvorenost omogućila svojim korisnicima personaliziranje sustava tako da dodaju, mijenjaju i modificiraju alate prema svojim potrebama. Isto tako druga važna karakteristika Linuxa je njegova sigurnost. Razlog zašto je Linux jedan od najsigurnijih operacijskih sustava leži u njegovoj zajednici programera koja iskorištava njegovu otvorenost te tako brzo otkriva i ispravlja sigurnosne propuste. Sve te karakteristike opravdavaju razlog zašto je Linux danas jedan od najpopularnijih softvera otvorenog kôda.

2.4.2. Git

Git je jedan od najpoznatijih i najkorištenijih sustava za verzioniranje. Kreirao ga je Linus Torvalds 2005. godine te je korišten kao jedan od glavnih alata u razvoju Linux kernela. Kasnije njegova popularnost se sve više povećavala što ga je učinila vodećim sustavom za verzioniranje kôda i neizostavni dio razvoja softvera pored ostale konkurencije kao što su Mercurial i Bazaar. Git omogućuje svojim korisnicima jednostavno i sigurno verzioniranje izvornog kôda na distribuirani način. Njegova distribuiranost je omogućila programerima izvanmrežni rad kao i međusobnu kolaboraciju timova koji su međusobno fizički udaljeni [6].

2.4.3. Mozilla Firefox

Mozilla Firefox je besplatni web preglednik otvorenog kôda koji je imao veliki utjecaj na razvoj interneta. Kreirala ga je tvrtka Mozilla Foundation 2002. godine te ga je učinila jednim od prvih web preglednika usmjerenih prema korisniku. Ono po čemu se ističe od ostalih web preglednika je njegova brzina, sigurnost i kontrola. [7] Dostupan je na Windows, Linux i Mac operacijskim sustavima te isto tako i na Android i iOS mobilnim uređajima. Postoji mnogo verzija Firefoxa poput, Quantum, Nightly, Beta i Developer Edition od kojih svaka ima svoje posebne značajke. Neke od glavnih značajki su: blokiranje reklama, prilagođavanje izgleda preglednika, privatno pretraživanje i upravljanje lozinkama [8].

2.4.4. Visual Studio Code

Visual Studio Code je besplatni uređivač kôda koji je razvijen od strane Microsofta. Vrlo je popularan alat među programerima jer omogućuje pisanje, uređivanje i debugiranje kôda te isto tako sadrži mnoge mogućnosti dodavanja vlastitih ekstenzija koje olakšavaju sam razvojni proces. Pošto je njegov kôd otvoren, programeri imaju mogućnost modificirati i nadograđivati alat prema njihovim potrebama. Podržava različite programske jezike i tehnologije za razvoj softvera te isto tako omogućuje verzioniranje kôda uz pomoć Git sustava za verzioniranje.

Osim ovih navedenih postoje i mnogi drugi softveri otvorenog kôda koji su se istaknuli u računalnome svijetu. Neki od tih koje je vrijedno spomenuti su: LibreOffice, PHP, Python, Audacity, Blender i mnogi drugi.

2.5. Licence

Softver otvorenog kôda koliko god bio revolucionaran pokret ima i svoje nedostatke. Jedan od tih nedostataka je i njegova prednost, a to je otvorenost i transparentnost. Bilo tko, tko ima pristup izvornom kôdu može mu naštetiti na razne načine, poput kopiranja izvornog kôda, neovlaštenog distribuiranja, modificiranja kôda i sl. Kako bi spriječili potencijalne prijetnje u razvoju softvera otvorenog kôda važno je da se on zasniva na licenci.

Licenciranje softvera označava dopuštena prava za korištenje računalnog programa. Kako Miriam Ballhausen spominje u svom istraživačkom članku licence u okviru otvorenoga kôda pružaju svojim korisnicima četiri bitne slobode, to su sloboda korištenja, učenja, dijeljenja i modificiranja softvera [9]. Sve te slobode moraju biti očuvane s pomoću licenci kako bi rad unutar zajednica otvorenog kôda bio što sigurniji i produktivniji.

Ukratko prema Andrew St. Laurentu licenca „otvorenog kôda“ osnovna uloga licence u razvoju otvorenog kôda je sprječavanje bilo kakvog iskorištavanje tuđeg rada [10, str. 4].

Odabir licence u razvoju softvera otvorenog kôda je jako važan dio razvojnog ciklusa jer utječe na budući uspjeh projekta. Nakon što programer odabere licencu za svoj softver, može koristiti bilo koju od već odobrenih licenci, Međutim, ako želi stvoriti novu open source licencu, ona mora biti odobrena od strane Open Source Initiative (OSI). OSI je neprofitna organizacija čija je uloga provjeravanje i odobravanje licenci otvorenog kôda. Kako bi licenca bila odobrena OSI provjerava udovoljava li ona Definiciji Otvorenog Kôda („Open Source Definition“) [11]. Definicija se sastoji od sljedećih točaka:

- *Slobodna redistribucija:* Licenca ne bi smjela zahtijevati od bilo koga tko redistribuira softver (bilo to besplatno ili po određenoj cijeni), da mora platiti naknadu izvornom vlasniku kôda.
- *Izvorni kôd:* Softver mora sadržavati izvorni kôd i mora se moći distribuirati u izvornom i kompajliranom obliku. Korisnik mora imati mogućnost modificiranja kôda.
- *Izvedeni radovi:* Licenca mora dopustiti izmjene originalnog softvera i kreiranje novih radova temeljenih na originalu pod istim uvjetima kao i originalni softer.
- *Integritet autorovog izvornog kôda:* Licenca može ograničiti distribuciju izvornog kôda u modificiranom obliku samo ako dozvoljava distribuciju "patch datoteka" s izvornim kôdom radi modifikacije programa tijekom izgradnje. Također, licenca mora eksplicitno dopustiti distribuciju softvera koji je izgrađen iz modificiranog izvornog kôda.
- *Nema diskriminacije prema drugima ili grupi:* Licenca ne smije diskriminirati bilo koju osobu ili grupu.
- *Nema diskriminacije u područjima djelovanja:* Licenca ne smije ograničiti da se softver koristi u različitim područjima *djelovanja*. Na primjer u svrhu istraživanja ili poslovanja.
- *Distribucija licence:* Prava povezana s programom se moraju odnositi na sve kojima se program redistribuira bez potrebe za izradom nove licence.
- *Licenca ne smije biti specifična za proizvod:* Ne smije biti razlike u pravima između korisnika koji su dobili program iz različitih distribucija, sve dok se pridržavaju uvjeta licence programa. Ovo osigurava pravednost i jednakost prava među korisnicima softvera, bez obzira na način na koji su došli do programa.

- *Licenca ne smije ograničavati drugi softver:* Licenca ne smije ograničavati ostali softver koji se distribuira s licenciranim softverom.
- *Licenca mora biti tehnološki neutralna:* Licenca ne smije ograničavati korisnike da koriste određenu tehnologiju [12].

Obično kada se odabire licenca pravilo je da se ona ne mijenja do kraja njegovog razvoja. Stoga je važno da programer koji započinje projekt dobro razumije i prouči postojeće licence. Iako autori svojih softverskih rješenja imaju mogućnost definiranja vlastitih pravila za korištenje njihovog softvera otvorenog kôda, često se koriste već poznate licence.

Zbog velikog broja licenci u razvoju softvera otvorenog kôda važno je razumjeti razlike između njih kako bi što lakše mogli odabrati onu koja odgovara projektu. Licence se dijele u tri kategorije koje najbolje opisuju o kakvoj se licenci radi. Kategorije su podijeljene na ne copyleft, jake copyleft i slabe copyleft licence.

Ne copyleft licence su najblaži primjeri licenci što se tiče ograničavanja korisnika izvornog kôda. Jedna od najpopularnijih takvih licenci je MIT licenca koja je poznata po tome što je najpopustljivija. Korisnici te licence mogu mijenjati, brisati, distribuirati, prodavati i koristiti izvorni kôd sve dok uključuju licencu i autorska prava. Uz nju tu je još i BSD licenca koja isto kao i MIT licenca omogućuje korisnicima bilo kakvo korištenje izvornog kôda sve dok se on nalazi pod licencom i autorskim pravima. Jedina razlika između tih dviju licenci je u tome što BSD licenca zabranjuje korištenje imena projekta ili sudionika za promicanje drugih proizvoda. Treća, isto tako popularna ne copyleft licenca je Apache licenca koja je vrlo slična MIT licenci, no jedina bitna razlika je u tome što Apache licenca koristi patentnu zaštitu što znači da korisnici softvera koji se nalazi pod ovom licencom dobivaju prava na sve patente što sprječava autore softvera ili druge sudionike da poduzmu bilo kakve radnje protiv korisnika zbog kršenja patenata.

Jake copyleft licence su druga kategorija licenci koje kao i ne copyleft licence omogućuju korištenje izvornog kôda softvera, no razlika je u tome što jake copyleft licence traže da se izvedeni radovi distribuiraju pod jednakim uvjetima, što bi značilo da ako netko modificira izvorni kôd softvera ne može ga kasnije distribuirati kao vlasnički softver već bi svatko trebao imati pravo na njegovo korištenje i mijenjati. Najpopularniji primjer takve licence je GPL (GNU General Public Licence). Ono što je bitno kod nje je da se modificirani izvorni kôd distribuira pod jednakim uvjetima kao i originalni.

Posljednja kategorija su slabe copyleft licence koje su iste kao i jake copyleft licence no jedina je razlika u tome što slabe dopuštaju povezivanje s vlasničkim softverom bez da kôd bude otvoren. LGPL (GNU Lesser General Public License) je jedan od primjera takve licence

koja je napravljena posebno za biblioteke i omogućuje povezivanje s vlasničkim aplikacijama. Druga isto tako popularna licenca iz ove kategorija je Mozilla Public Licence koja ograničava da izvorni kôd mora biti objavljena pod istom licencom, ali ako se implementira neki novi kôd

Naziv	Kategorija	Karakteristike
MIT	Ne copyleft	Jednostavna, neograničeno korištenje izvornog kôda i distribucije
BSD	Ne copyleft	Neograničeno korištenje izvornog kôda i distribucije, zabranjeno korištenje imena projekta ili sudionika za promoviranje
Apache	Ne copyleft	Neograničeno korištenje izvornog kôda i distribucije, uključuje patentnu zaštitu
GPL	Jaka copyleft	Modificirani izvorni kôd distribuira pod jednakim uvjetima kao i originalni
LGPL	Slaba copyleft	Napravljena za biblioteke, omogućuje povezivanje s vlasničkim aplikacijama
Mozilla Public Licence	Slaba copyleft	Izvorni kôd mora biti objavljen pod istom licencom, a novi kôd može biti pod različitim

onda može biti pod drugim licencama [11].

Tablica 3: Usporedba najpopularnijih licenci

Važno je pažljivo odabrati licencu koja odgovara potrebama projekta i omogućava željeni nivo slobode i zaštite autorskih prava. Postoji mnogo web stranica koje olakšavaju programerima taj proces. Evo nekoliko takvih stranica:

- **Choose a License** - Ova web stranica daje jednostavne smjernice i olakšava kompletni proces odabira licence. Programeri mogu brzo pronaći licencu koja odgovara njihovim potrebama putem jasnih i sažetih opisa. Stranici se može pristupiti putem sljedeće poveznice: https://choosealicense.com/.
- **Open Source Initiative (OSI)** - Na ovoj stranici se nalaze sve postojeće open source licence s njihovim opisima. OSI također daje smjernice korisnicima za odabir licence ovisno o vrsti projekta. Stranici se može pristupiti putem sljedeće poveznice: https://opensource.org/licenses.

- **GNU Project** - Ako korisnici žele koristiti licence poput GPL ili LGPL, web stranica GNU Projecta može im pomoći da dobiju više informacija. Stranica pruža detaljne opise, često postavljana pitanja i vodiče za korištenje GNU licenci. Stranici se može pristupiti putem sljedeće poveznice: https://www.gnu.org/licenses/.
- **Licence Selector** – korisnici putem ove web stranice mogu na jednostavan način odabrati licencu jer sadrži mnogo pitanja i odgovora koji mogu pomoći pri odabiru licence te isto tako daju preporuke za iste. Stranici se može pristupiti putem sljedeće poveznice: https://ufal.github.io/public-license-selector/.

Sve u svemu jako je važno da programeri budu svjesni pravila i obaveza koje proizlaze iz odabrane licence te da se pridržavaju tih pravila kako bi se osigurala legalnost i održivost projekta otvorenog kôda.

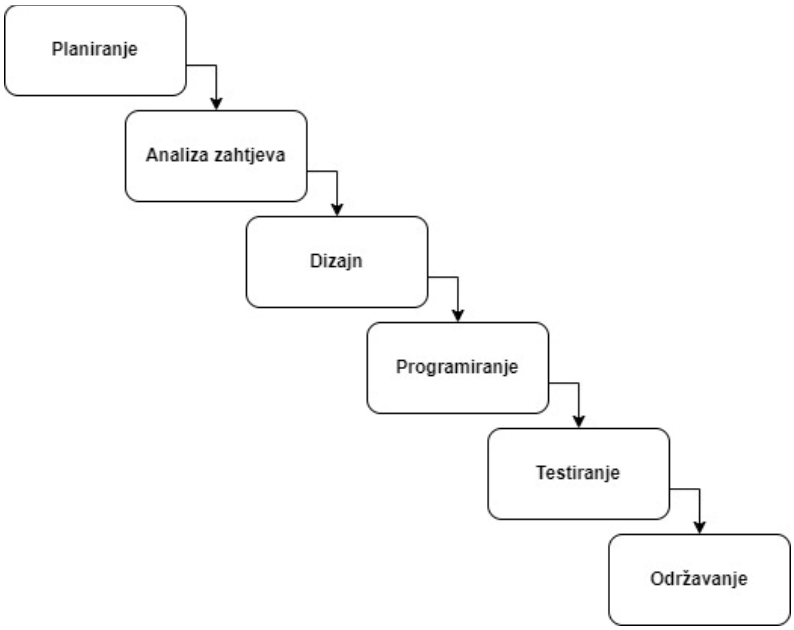
3. Razvoj softvera otvorenog kôda

3.1. Usporedba životnog ciklusa otvorenog i zatvorenog kôda

Životni ciklus softvera je proces koji obuhvaća sve faze razvoja softvera, od početnog definiranja zahtjeva pa sve do njegovog objavljivanja i održavanja [13]. Isto tako životni ciklus opisuje metodologiju prema kojoj će se razvijati softver. To je jako važno kako ne bi došlo do zaostataka u razvoju, slabije kvalitete softvera i dodatnih troškova. Razvojni proces softvera otvorenog kôda se podosta razlikuje od tradicionalnog razvoja zatvorenog kôda. Tradicionalni proces se sastoji od sljedećih faza:

- **Planiranje –** u ovoj se fazi postavljaju neka od glavnih pitanja koja se tiču izvedivosti projekta, troškova i modela razvoja.
- **Analiza zahtjeva –** skupljaju se i analiziraju zahtjevi
- **Dizajn –** kreira se potrebna dokumentacija važna za razvoj. Dokumentacija se sastoji od različitih dijagrama i opisa zahtjeva kako bi programeri što bolje razumjeli kako softver treba raditi.
- **Programiranje –** predstavlja doslovno programiranje. Dokumentacija se prebacuje u stvarni proizvod.
- **Testiranje –** razvijeni softver se testira korištenjem različitih tehnika i testnih slučajeva kako bi se što prije utvrdili i uklonili nedostaci.

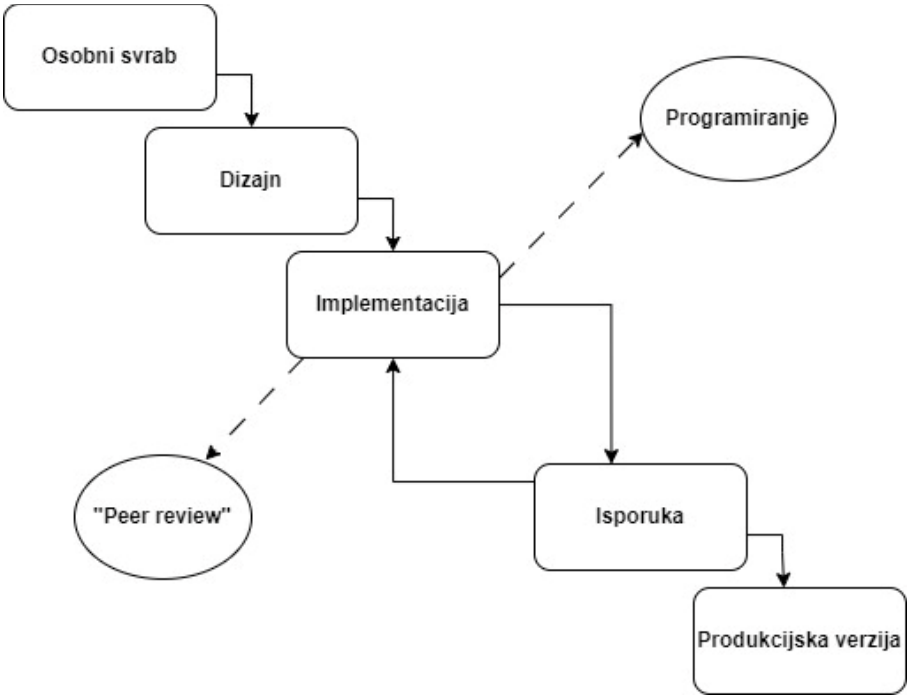
- **Održavanje –** ovo je faza koja zapravo najduže traje i dolazi nakon što je softver isporučen. U ovoj fazi se ispravljaju greške, dodaju poboljšanja i izmjenjuju dijelovi softvera [14].



Slika 1: Životni ciklus tradicionalnog razvoja softvera

S druge strane razvoj softvera otvorenog kôda započinje, kao što je Raymond rekao u svojoj knjizi „*The Cathedral and the Bazaar*“ [15], češanjem osobnog svraba programera, što znači da programeri često započinju projekte kako bi riješili vlastite probleme ili zadovoljili osobne potrebe. Analize zahtjeva praktički nema pošto su pokretači projekta ti koji najbolje razumiju na koji način bi on trebao raditi. Nakon toga dolazi do dizajna aplikacije. Ovaj korak je isti kao i kod tradicionalnog načina razvoja te se u njemu kreiraju važni dokumenti, prototipi, dijagrami koji će olakšati razumijevanje načina rada softvera koji se razvija. Dizajn je jako bitno napraviti prije nego li se omogući kolaboracija drugih programera kako bi prilikom njihovog uključivanja u projekt mogli imati sve potrebne materijale za razvoj [15]. Sljedeći je korak implementacija u kojem se kreirani dizajn aplikacije pretvara u softver koji se može koristiti. Glavna razlika u implementaciji tradicionalnog i otvorenog softvera je ta što u razvoju

otvorenog kôda korisnici mogu birati što žele implementirati i nisu ograničeni na bilo kakav način dok kod tradicionalnog pristupa to nije moguće. Isto tako jedan od ključnih procesa faze implementacije je i „peer review“. „Peer review“ predstavlja međusobno pregledavanje kôda programera kako bi se spriječili nedostatci i greške u kôdu te isto tako kako bi se povećala njegova kvaliteta i funkcionalnost [16]. Nakon što je softver implementiran tako da omogućuje sve zamišljene funkcionalnosti, on se odmah isporučuje. Razlog tome je kako bi se što prije dobila povratna informacija korisnika o mogućim propustima, nedostacima i sl. Kada se isprave svi nedostaci i utvrdi da je softver stabilan, on se isporučuje kao „Produkcijska verzija“ [14].



Slika 2: Životni ciklus razvoja softvera otvorenog kôda

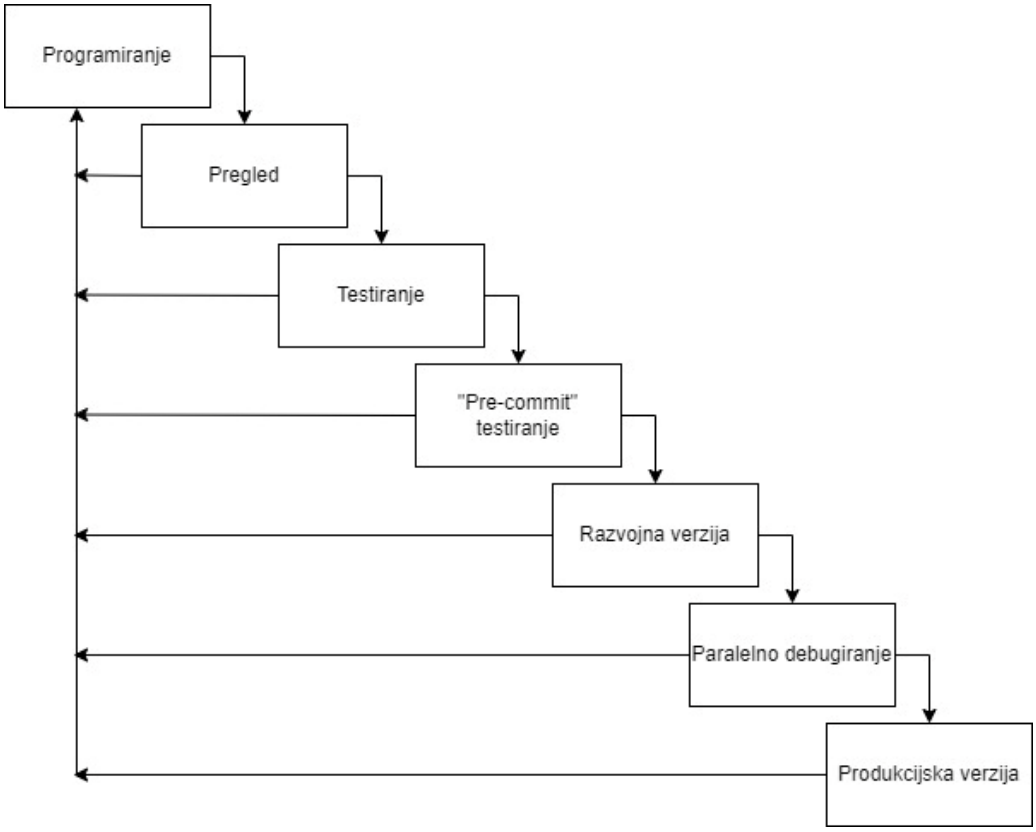
3.2. Razvojni modeli

Postoji mnogo pristupa koji pokušavaju objasniti filozofiju otvorenog kôda, no još niti jedan razvojni model otvorenog kôda nije priznat kao standard. Razlog tome je to što svaki projekt ima različiti pristup što se tiče organizacije i rada. U ovom odjeljku će biti opisani neki od poznatijih model razvoja softvera otvorenog kôda.

Prvi model koji je isto tako poslužio za usporedbu sa životnim ciklusom tradicionalnog razvoja je „Jorgensenov“ model. Model je široko prihvaćen u mnogim projektima otvorenog kôda te se sastoji od šest glavnih faza, a to su:

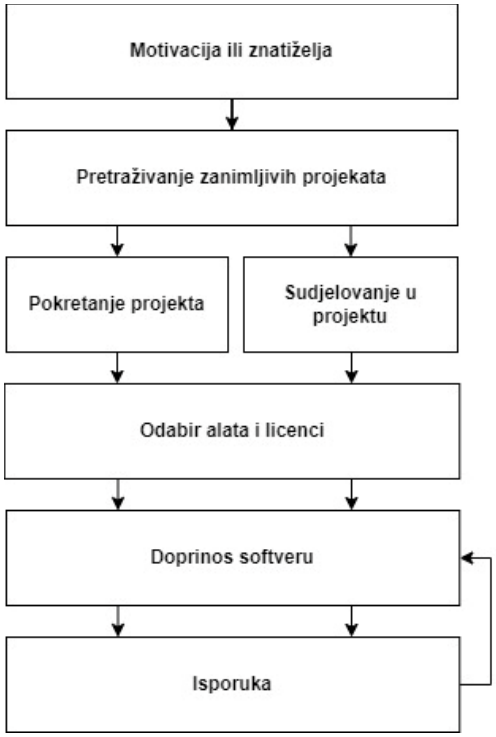
1. *Programiranje:* Ovu fazu pokreću programeri koji su došli na ideju za izradom softvera. Faza se uglavnom temelji na programiranju softvera koji kasnije postaje javno dostupan drugim sudionicima kako bi se on mogao unaprijediti.
2. *Pregled*: Programeri si međusobno pregledavaju kôd.
3. *„Pre-commit" testiranje*: Nakon pregleda razvijeni kôd se testira kako bi se pronašle greške i izbacio nepotrebni kôd.
4. *Razvojna verzija:* Ako se kôd smatra dobrim uključuje se u razvojnu verziju softvera.
5. *Paralelno debugiranje:* Kada se kôd doda u razvojnu verziju sudionici na projektu provode detaljnu analizu kôda kako bi pronašli i prijavili bilo kakve nedostatke i pogreške.
6. *Produkcijska verzija:* Faza u kojoj se razvojna verzija, ako je ona stabilna prebacuje u produkcijsku.

Kroz sve ove faze se konstantno iterira kako bi završni proizvod bio što kvalitetniji i stabilniji [14].



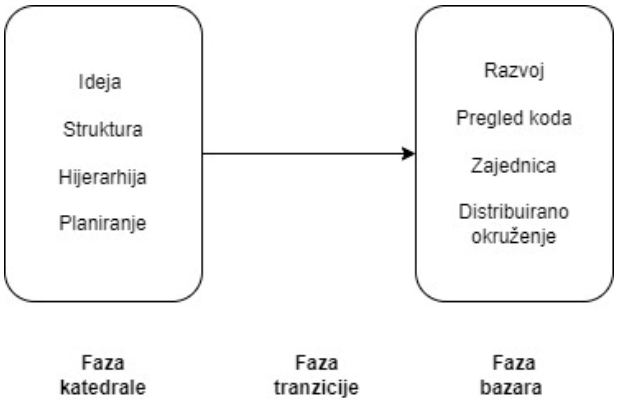
Slika 3: „Jorgensenov" razvojni model

Sljedeći poznatiji razvojni model je „*Wu i Linov*“ model u kojemu slijed razvoja prvo započinje početnom motivacijom i znatiželjom, nakon čega dolazi do pretraživanja projekata koji su nam zanimljivi. Sve to rezultira ili pokretanjem vlastitog projekta otvorenog kôda ili sudjelovanjem u jednom. Neovisno što odlučili sljedeći korak u modelu je odabir glavnih alata i licence koja će se koristiti za razvoj. Korisnici zatim mogu dopridonijeti softveru bilo to pisanjem dokumentacije, programiranjem ili odabirom licence. Nakon što ciklus prođe kroz sve faze, službena verzija softvera se isporučuje te se sve faze ponavljaju [17].



Slika 4: Wu i Linov razvojni model

Andrea Capiluppi i Martin Michlmayr govore o tome kako se razvojni model softvera otvorenog kôda temelji i na zatvorenom i otvorenom stilu razvoja [19]. Njihov se model sastoji od tri faze. Prva faza je „Faza katedrale“ gdje se koristi razvojni stil izgradnje katedrale.Tako se zove jer se naglašava struktura, hijerarhija i planiranje u razvoju softvera, što je karakteristično za zatvoreni stil. Nakon što je originalna ideja razrađena, ona se objavljuje što rezultira pokretanjem „Faze prijelaza“.Ova faza predstavlja vezu između katedralne faze i faze bazara. Ono što ju karakterizira je to da se ona pokreće kada je dizajn početne ideje stabilan te kada originalni autor više ne želi individualno razvijati softver. Posljednja faza, tzv. „Faza bazara gdje razvoj napreduje unutar distribuiranog okruženja. U ovakvom se okruženju na prvo mjesto stavlja kolaboracija između sudionika koji zajedno doprinose projektu [18].



Slika 5: Andrea Capiluppi i Martin Michlmayrov razvojni model

Osim ovih navedenih modela, postoji mnogo drugih modela u razvoju softvera otvorenog kôda. Iako svaki model razvoja softvera otvorenog kôda predlaže vlastiti pristup, postoje mnoge sličnosti između svih tih modela. Jedna od tih sličnosti je početna motivacija koja pokreće programere da sudjeluju ili pokrenu projekt otvorenog kôda. U gotovo svim modelima, ova faza se smatra početnom fazom razvoja. Sljedeća sličnost je rana i česta isporuka softvera. Ova praksa omogućava brzo dobivanje povratnih informacija od korisnika i kontinuirano unapređenje softvera. Isto tako jedna od glavnih faza u tim modelima je faza pregleda koja uključuje detaljnu analizu kôda od strane članova zajednice. Posljednja sličnost je kontinuirano poboljšanje kvalitete koje predstavlja konstantno ponavljanje svih faza razvojnog ciklusa softvera s ciljem da on postane što bolji i kvalitetniji.

3.3. Korišteni alati

Kako bi razvojni ciklus otvorenog kôda prošao što bolje i bez ikakvih propusta poput lošijih sigurnosnih mjera, grešaka u kôdu, loše dokumentacije, loše kolaboracije između programera, važna je uloga alata koji uvelike olakšavaju praćenje razvoja u svim odjelima od inženjeringa pa sve do marketinga. Ti alati daju programerima sve potrebne resurse koji su im potrebni za izradu što kvalitetnijeg softvera. Primjeri tih alata dijele se na alate za upravljanje izvornim kôdom, alate za praćenjem stanja projekta i alate za komunikaciji i suradnju.

3.3.1. Alati za verzioniranje

Pošto se u razvoju softvera uglavnom radi u tekstualnim datotekama, neophodno je te datoteke pohraniti na mjesto koje će biti dostupno svima i u bilo kojem trenutku. Razlog tome

je što programeri koji sudjeluju u projektima otvorenog kôda dolaze iz različitih dijelova svijeta. Kako bi se taj problem riješio, koriste se sustavi za verzioniranje, poznati kao Version Control Systems(VCS).

Sustavi za verzioniranje sadrže potrebne alate koji pojednostavljuju i daju slobodu prilikom rada na projektu. Pod davanjem slobode misli se na to da svaki programer može koristiti izvorni kôd bez ometanja drugih članova projekta tako da se prilikom povlačenja izvornog kôda svaki put kreira nova verzija koja ne se ne preklapa s verzijom drugih članova. Osim toga još jedna karakteristika ovih alata je vraćanja na prošle verzije što omogućuje lakše identificiranje i ispravljanje grešaka, lakše testiranje u različitim fazama razvoja te lakši oporavak podataka.

Postoji više tipova sustava za verzioniranje izvornog kôda: lokalni, centralizirani i distribuirani. Ono što je važno znati jest da se kod lokalnih sustava za verzioniranje datoteke projekta i sam projekt nalaze lokalno, na primjer, na našem računalu, kod centraliziranih sustava postoji samo jedan repozitorij koji se nalazi na serveru, a kod distribuiranih se koristi više repozitorija, pri čemu svaki programer ima kopiju repozitorija na svom računalu [19].

U razvoju softvera otvorenog kôda se posljednjih godina koriste distribuirani sustavi za verzioniranje, poput Gita, Mercuriala i Bazaara dok su se u počecima koristili uglavnom centralizirani sustavi, kao što su CSV i Subversion. Razlog prelaska na distribuirane sustave je taj što omogućuju veću fleksibilnost i sigurnost zbog postojanja više repozitorija [20].

3.3.2. Alati za praćenje grešaka

U razvojnom ciklusu otvorenog kôda, jedan od ključnih izazova je rješavanje i upravljanje greškama. Zbog tog razloga alati za upravljanje greškama, takozvani Bug Trackeri postaji neizostavni dio razvojnog procesa. Uloga tih alata je omogućiti programerima koji sudjeluju u projektu otvorenog kôda lakše prijavljivanje, praćenje i rješavanje problema.

Bitno je istaknuti da bug trackeri predstavljaju platformu za prijavljivanje grešaka u softveru što omogućuje svim članovima zajednice jednostavno prijavljivanje, ali i odabir grešaka koje žele ispraviti. Tako bilo tko može uz pomoć alata pronaći problem koji odgovara njegovom znanju i vještinama te tako dopridonijeti zajednici [21].

Neki od najpoznatijih bug trackera koji se koriste u projektima otvorenog kôda su: Bugzilla, GitHub Issues, GitLab i JIRA. Neke od glavnim mogućnosti ovih alata su sljedeće:

- **Prijavljivanje grešaka** - korisnici mogu prijavljivati greške putem sučelja koje pruža alat. Način na koji će se opisati greška ovisi o zajednici, no najčešće je to tako da se

prvo napiše naslov koji opisuje grešku, zatim se napiše ono što se očekivalo da softver radi, te zatim što se zapravo dogodilo. Uz to se može dodati i dijelove kôda, slike i sl.

- **Praćenje statusa greške** – svaka greška ima svoj životni vijek te je važno znati u kojoj je fazi kako bi programeri lakše pratili napredak u rješavanju grešaka.
- **Prioritizacija** - omogućuje prioritizaciju grešaka prema njihovoj važnosti.
- **Dodjeljivanje zaduženja –** korisnik koji želi dopridonijeti projektu može izabrati grešku koju želi ispraviti

3.3.3. Platforme za razvoj softvera

Kako bi programeri na što jednostavniji i organiziraniji način sudjelovali u projektima otvorenog kôda važno im je da sve alate koji su im potrebni za kolaboraciju imaju na jednome mjestu. Ovdje dolaze u pomoć platforme posvećene razvoju softvera otvorenog koda koje pružaju prostor u kojem mogu raditi zajedno na stvaranju, dijeljenju i poboljšanju softverskih rješenja sve unutar jedne aplikacije [22]. Platforme dolaze s nizom značajki koje podržavaju svaku fazu procesa razvoja softvera od planiranja i kodiranja do testiranja, postavljanja i održavanja. Postoji mnogo takvih platformi, no neke od najpopularnijih su GitHub i GitLab.

GitHub se ističe kao jedna od najpoznatijih platformi za razvoj softvera otvorenog kôda. Osnovan 2008. brzo se pojavio kao platforma za hosting i razmjenu repozitorija kodova [23]. Razlog njegove popularnosti su mnoge mogućnosti koje pruža. GitHub koristi Git, distribuirani sustav za kontrolu verzija koji razvojnim programerima omogućuje praćenje promjena koda, vraćanje na verzije i besprijekornu suradnju s drugima uz očuvanje integriteta podataka. Osim toga GitHub omogućuje kreiranje fork repozitorija i pull zahtjeva čime olakšava cjelokupni proces kolaboracije unutar zajednice. Sljedeća mogućnost koju pruža je praćenje grešaka čime razvojnim timovima daje mogućnost lakšeg organiziranja i prijavljivanja nedostataka unutar projekta te kreiranje diskusija kojima se nastoji riješiti greške. Još jedna od mogućnosti je kreiranje automatiziranih procesa koji pomažu pri recenziji pull zahtjeva suradnika. Osim toga putem platforme je moguće pretraživati različite repozitorije koji se tiču softvera otvorenog koda što je jako korisno za osobe koje žele sudjelovati u njima.

Druga platforma koja je isto tako među popularnijima je GitLab. GitLab, kao i GitHub omogućuje svojim korisnicima kontrolu verzija za podršku suradnje i praćenje verzija. Osim toga u GitLabu se nalaze alati za planiranje koji omogućuju lakše organiziranje i praćenje timskog napretka . Isto tako omogućuje pisanje dokumentacije putem wiki stranica čime olakšava dijeljenje i upravljanje dokumentacijom. GitLab se također sastoji od alata za praćenje važnih metrika unutar projekta, što je jako korisno za osobe koje su namijenjene za

vođenje projekta kako bi mogli vidjeti je li proces razvoja ide u dobrom smjeru ili ne. Još jedna mogućnost je i kontinuirana integracija kojom se ubrzavaju mnogi automatizirani procesi [24].

Razvoj softvera s otvorenim kodom temelji se na zajedničkom radu, transparentnosti i stalnom unapređenju. Platforme poput GitHub-a, GitLab-a pružaju potrebne alate i infrastrukturu za podršku tim vrijednostima. Njihova sposobnost da povežu programere i olakšaju suradnju čini ih ključnim resursima u suvremenom razvoju softvera. Korištenjem ovih platformi, programeri mogu ne samo doprinositi postojećim projektima, već i stvarati nova rješenja koja mogu koristiti širokoj zajednici, potičući inovacije i tehnološki napredak. Otvoreni kod tako postaje osnova razvoja softvera, omogućavajući transparentnost, sigurnost i suradnju na globalnoj razini.

3.4. Ostali alati

Pored alata za verzioniranje i praćenje grešaka, postoje i mnogi drugi alati koji su neizostavni u razvoju softvera otvorenog kôda.

Alati za skeniranje izvornog kôda i provjeru licenci ključni su dio osiguravanja kvalitete i sigurnosti kôda. Alati poput Black Duck Hub, FOSSA, Copyright review tools i WhiteSource omogućuju automatsko otkrivanje ranjivosti u kôdu te provjeru usklađenosti s licencom. Ovi alati omogućuju programerima lakše uočavanje propusta u kôdu i brže rješavanje istih.

Sljedeći su alati za upravljanje izdanjima, koji pripremaju softver za isporuku tako da automatski integriraju promjene u kôdu i izvršavaju testove. Ovdje se ističu Docker Hub i GitHub Release kao popularni alati koji omogućavaju automatizaciju i olakšavaju proces izdavanja softvera.

Isto tako osim alata koji su direktno povezani s kôdom važni su i alati koji omogućuju komunikaciju i suradnju između članova zajednice. Primjeri tih alata bi bili Github Discussions, Slack i Twiki [25].

3.5. Načini doprinosa i kolaboracije

Kolaboracija u projektima otvorenog kôda je jedan od ključnih dijelova njegovog razvojnog procesa. U takvim se projektima sve zasniva na zdravoj suradnji i komunikaciji između sudionika kako bi razvoj bio što bolji i kvalitetniji. Korisnici mogu pokazati svoj doprinos putem platformi poput GitHub-a na kojem se nalaze mnogi projekti otvorenog kôda.

Važno je da prilikom prvog doprinosa korisnici pronađu projekt prema njihovim vještinama i znanjima kako bi doprinos bio što kvalitetniji i vrjedniji za zajednicu. Isto tako jako je važno izabrati projekt koji će korisniku pružiti zadovoljstvo prilikom doprinosa. Kada se pronađe projekt potrebno je dobro proučiti dokumentaciju i smjernice doprinosa kako bi kolaboracija tekla bez ikakvih problema. Nakon što se korisnik upozna sa smjernicama potrebno je odlučiti na koji način se želi pomoći softveru. Jedan od tih načina je pronalaženje i prijavljivanje nedostataka unutar softvera. Korisnici koji žele pomoći u razvoju mogu jednostavno koristiti aplikaciju i ako uoče neki nedostatak prijaviti ga kroz neki od alata za praćenje grešaka.

Sljedeći mogući način doprinosa je pisanje i poboljšavanje dokumentacije. Kao što je već znano, softver otvorenog kôda nema jasno definiranu dokumentaciju koja je napisana od strane stručnjaka, već je ona pisana kroz veliki niz iteracija razvoja, stoga je važno da se dokumentacija konstantno poboljšava kako bi budući sudionici projekata imali dobru podlogu za razvoj i kako bi što brže i efikasnije mogli doprinijeti zajednici.

Jedan od najpopularnijih i najčešćih načina doprinosa koji predstavlja i najvažniji dio razvoja je programiranje. Svaki programer, bilo početnik ili napredan ima veliku ulogu u razvoju softvera otvorenog kôda. Rješavanje grešaka, dodavanje novih funkcionalnosti, refaktoriranje kôda, sve su načini putem kojih programeri mogu dati svoj doprinos. Osim pisanja kôda, pomaganje zajednici dijeljenjem vlastitog znanja i ideja je isto tako od velike važnosti.

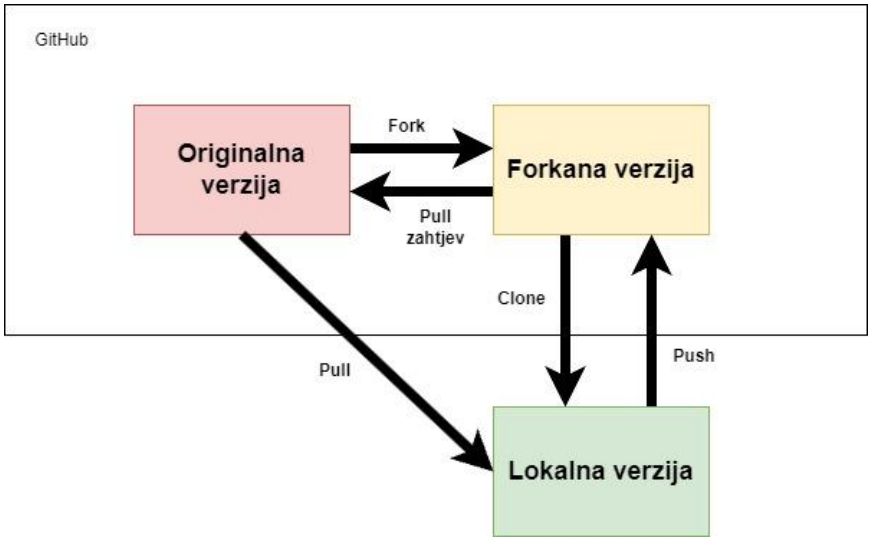
Uz sve te načine doprinosa važna je dobra komunikacija koja predstavlja ključ uspješnog razvoja softvera otvorenog kôda. Što je bolja komunikacija između sudionika na projektu to će oni biti više motivirani, što će rezultirati bržim razvojem, sigurnijim softverom i kvalitetnijim razvojnim procesom.

3.6. Fork & Pull model

Sada kada smo se upoznali s različitim načinima kolaboracije važno je razumjeti cijeli proces doprinosa projektu otvorenog kôda. Model za doprinos i kolaboraciju koji se koristi u projektima otvorenog koda je Fork & Pull model. Model se sastoji od nekoliko važnih koraka koji omogućuju nesmetani rad i jednostavnu kolaboraciju.

Prvi korak u Fork & Pull modelu je forkanje glavnog repozitorija. Forkani repozitorij predstavlja kopiju glavnog repozitorija kako bi se mogao dati doprinos bez utjecanja na originalni projekt. Sljedeći korak je kloniranje forkanog repozitorija na lokalno računalo uz pomoć naredbe *git clone {url_projekta}* ili preuzimanjem .zip datoteke, nakon čega možemo pristupiti izvornom kôdu i raditi promjene. Kada je projekt kloniran lokalno, git je automatski

postavio lokalni repozitorij zvan „origin“, da pokazuje na forkani repozitorij, što omogućuje commitanje promjena i spremanje istih na njega. Nakon što se izvorni kôd nalazi na našem računalu važno je kreirati posebnu granu na kojoj će se moći brisati, izmjenjivati i dodavati kôd te tako izolirati nove promjene od originalnog projekta. Kreiranje nove grane se radi uz pomoć naredbe *git branch {naziv_grane}* te prebacivanje na tu granu uz pomoć *git checkout {naziv_grane}*. Sljedeći korak, nakon što su promjene spremljene, je njihovo slanje na forkani repozitorij, što se radi uz pomoć naredbe *git push {naziv_repozitorija} {naziv_grane}*. Sada kada se promjene nalaze na forkanom repozitoriju potrebno je kreirati pull zahtjev s pomoću kojega tražimo od održavatelja projekta da pregledaju i recenziraju izmjene u kôdu. Unutar pull zahtjeva je potrebno napisati broj problema koji se riješio, njegov opis te kratki opis na koji način se došlo do rješenja. Na kraju ako su promjene prihvaćene od strane održavatelja može se izbrisati grana na kojoj su rađene promjene s pomoću naredbe *git branch -d {naziv_grane},* a ako nisu potrebno je ponovno raditi izmjene [26] [27]. Na slici 6 se može vidjeti grafički prikazan Fork & Pull model.



Slika 6: Fork & Pull model[26]

3.7. Recenzija koda

Pregled koda predstavlja važnu ulogu u razvoju softvera otvorenog koda. Nakon što suradnik na projektu preda rješenje nekog problema jako je važno da ono bude pregledano od strane održavatelja projekta te kako bi se identificirali mogući propusti u kodu, dala povratna informacija i osiguralo da je kod pisan prema standardima kvalitete. Primarni cilj je poboljšati

upravljivost, standard i pouzdanost koda, što doprinosi ostvarenju projekta. Također promiče razmjenu znanja i kontinuirano učenje unutar tima [28].

Proces recenzije koda u projektima otvorenog koda započinje pull zahtjevom. Mnogi projekti otvorenog koda koriste platforme za kontrolu verzija kao što su GitHub, GitLab ili Bitbucket. Programeri podnose pull zahtjeve kako bi predložili promjene za spajanje s glavnim repozitorijem. Pull zahtjev služi kao zahtjev za spajanje izmjena koda i olakšava održavateljima njihov pregled. Nakon što je pull zahtjev predan, kod prolazi kroz automatizirane alate koji provode analizu koda, testove i druge procjene kako bi se osigurala kvaliteta predanog rješenja. Ovi alati mogu identificirati probleme kao što su pogreške u sintaksi, je li kod dobro formatiran i pridržavanje smjernica za programiranje, čime se olakšava posao osobama koje pregledavaju kod. Kada je kod prošao automatizirane alate, ulogu pregleda preuzimaju održavatelji koji osobno pregledavaju rješenje i daju povratnu informaciju o tome je li predani kod zadovoljavajući ili je potrebno odraditi neke promjene. Ako je rješenje dobro ono se integrira s glavnim repozitorijem, a ako nije osoba koja je recenzirala kod daje prijedloge za poboljšanjem.

3.8. Uloge u razvoju softvera otvorenog kôda

U svaki projekt otvorenog koda uključeni su pojedinci, svaki sa svojim ulogama. Bitno je organizirati te sudionike na temelju njihovih uloga kako bi se osiguralo jednostavno i učinkovito odvijanje procesa razvoja. Svaka je uloga ključna u životnom ciklusu projekta, počevši od njegovog početka pa sve do isporuke. Evo nekih ključnih uloga koje se obično vide u projektima otvorenog koda:

- **Autor -** osoba koja je odgovorna za pokretanje projekta. Ova je uloga temeljna jer uključuje razvoj koji postavlja temelje i utvrđuje viziju projekta. Doprinosi autora često uključuju pisanje baze koda, projektiranje arhitekture i implementaciju osnovnih funkcionalnosti. Njihova vizija i smjer postavljaju ciljeve projekta i početni put prema naprijed. Stručnost i inovativnost kreatora temelj su na kojem će se projekt razvijati i napredovati.
- **Vlasnik –** osoba koja kontrolu nad repozitorijem. Ova uloga daje mogućnost donošenja odluka u vezi upravljanja i vođenja projekta. Vlasnici projekta ne moraju uvijek biti autori istog već je njima dodijeljena ta uloga kako bi nadgledavali projekt i kako bi razvojni ciklus tekao u skladu s zadanim ciljevima.
- **Održavatelji -** osobe zaduženi za upravljanje vizijom i organizacijskim aspektima projekta [29]. Oni igraju ulogu u osiguravanju održivosti i dugoročnog uspjeha

projekata. Održavatelji pregledavaju doprinos suradnika na projektu, integriraju promjene programera,daju zaduženja , nadgledavaju izdanja i prate u kojem smjeru ide projekt. Održavatelji osiguravaju da projekt ostane aktivan, zdrav i da odgovara potrebama korisnika. Oni također mogu biti autori ili vlasnici projekta.

- **Suradnici -** obuhvaćaju sve pojedince koji su dali doprinos projektu. Doprinosi mogu doći u oblicima kao što su programiranje, pisanje dokumentacije, prijave grešaka, testiranje, predlaganje poboljšanja i pomoć u podršci zajednice. Suradnici su ključni za rast i poboljšanje projekta, jer donose vještine i vlastito iskustvo. Njihov uloga pomaže u raspodjeli zadataka i potiče kreativnost. Osigurava da se projekt nastavlja razvijati kako bi zadovoljio zahtjeve korisnika. Suradnici igraju ulogu u poboljšanju funkcionalnosti, upotrebljivosti i ukupne kvalitete projekta sudjelovanjem i suradnjom.
- **Članovi zajednice -** članovi zajednice su pojedinci koji su uključeni u projekt. Mogu se pridružiti raspravama i ponuditi povratne informacije o problemima na koje su naišli prilikom korištenja projekta. Njihovo sudjelovanje je ključno jer pruža uvid u to kako se softver koristi u kontekstu stvarnog svijeta. Članovi zajednice pomažu u prepoznavanju grešaka, predlažu poboljšanja i potvrđuju nove funkcionalnosti. Štoviše, oni značajno pridonose promicanju projekta privlačeći i nove korisnike i suradnike. Povratne informacije i sudjelovanje zajednice vode razvoj projekta kako bi se osigurala njegova kontinuirana kvaliteta [29].

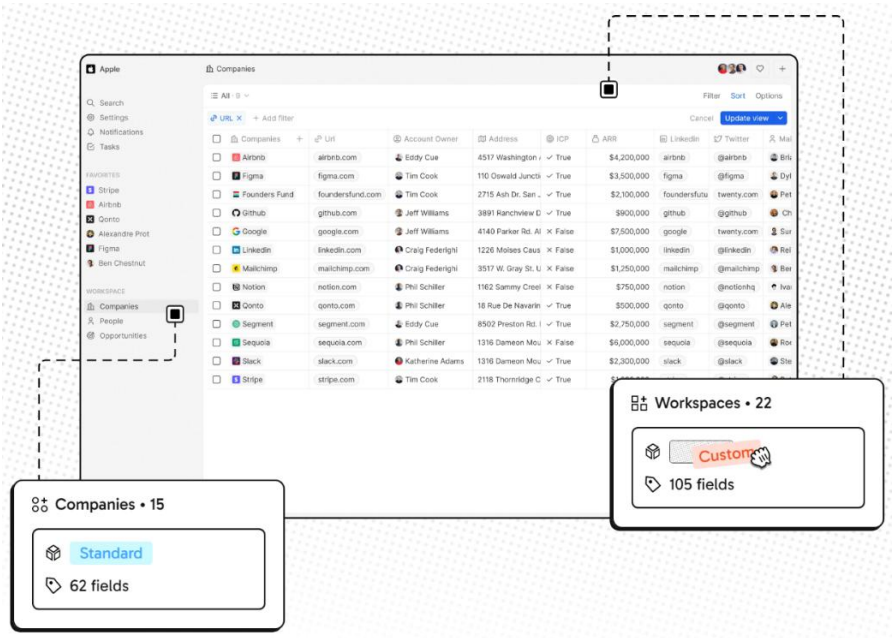
4. Primjer doprinosa softveru otvorenog koda

Unutar ovog poglavlja bit će prikazan doprinos odabranom softveru otvorenog kôda. Praktični primjer obuhvatit će opis tog softvera, kao i pravila doprinosa istome. Nakon toga, bit će opisan opseg problema te s njime povezano rješenje.

4.1. Odabrani softver

Softver otvorenog kôda na kojem će biti prikazan doprinos zove se Twenty. Twenty je softversko rješenje otvorenog kôda temeljeno na GPL licenci koje nudi svojim korisnicima jedinstveni pristup upravljanju odnosima s kupcima. Ono što odvaja Twenty od ostale konkurencije je pružanje potpune kontrole i slobode svojim korisnicima što im omogućuje da doprinose, sami hostaju i forkaju svoja CRM rješenja. Za razliku od drugih tradicionalnih CRM platformi koje nameću krute strukture, Twenty je dizajniran da radi s postojećim izvorima

podataka. Također Twenty se ističe po njegovom korisničkom iskustvu koje je inspirirano mnogim korisnički prijateljskim alatima poput Notiona [30].



Slika 7: Aplikacija Twenty [24]

Twenty nudi niz značajki dizajniranih kako bi olakšale procese upravljanja klijentima. Od praćenja klijenata i stvaranja prilika do upravljanja zadacima i integracije e-pošte. Osim toga, platforma pruža fleksibilnost prilagodbe modela podataka kako bi se zadovoljile specifične poslovne potrebe, dodatno poboljšavajući njezinu korisnost.

Twenty je trenutačno u fazi razvoja alfa verzije, s preko 200 doprinositelja aktivnih na projektu. Razvoj aplikacije prati se na GitHub platformi, olakšavajući korisnicima praćenje promjena i doprinošenje. Korisnici mogu doprinositi na različite načine, uključujući prijavljivanje grešaka i traženje novih značajki putem GitHub issues. Najčešći način doprinosa je kroz pisanje kôda, što uključuje refaktoriranje, ispravljanje grešaka, dodavanje novih značajki te optimizaciju kôda.

Postoji nekoliko razloga za odabir Twenty aplikacije. Prvo, Twenty ima aktivnu zajednicu koja je uvijek spremna pomoći u bilo kojem trenutku. Komunikacija između programera na projektu se odvija putem GitHuba i Discorda što je olakšao i sam doprinos. Drugo, aplikacija se koristi poznatim tehnologijama za razvoj što je bio jedan od većih motivatora za doprinos. Osim navedenih razloga, Twenty aplikacija predstavlja izvrsnu priliku za početnike u open source zajednici. Projekt je strukturiran na način koji olakšava uključivanje novih članova i pruža im podršku u učenju i razvoju vještina.

4.2. Doprinos softveru

4.2.1. Pravila doprinosa

Prije nego li se moglo doprinijeti softveru potrebno je proučiti priloženu dokumentaciju i datoteke koje se tiču pravila ponašanja i načina sudjelovanja.

U aplikaciji Twenty pravila ponašanja su opisana u datoteci CODE_OF_CONDUCT.md. U toj datoteci su navedene smjernice koje obvezuju sve sudionike zajednice na poštivanje i pridržavanje određenih standarda. Ova pravila uključuju obvezu poštivanja različitih pozadina i identiteta svih sudionika, demonstriranje empatije i ljubaznosti prema drugima te poštivanje različitih mišljenja i iskustava. Također, pravila zabranjuju upotrebu uvredljivog ili agresivnog jezika, uznemiravanje ili objavljivanje privatnih informacija bez dopuštenja. Svi sudionici zajednice su odgovorni za poštivanje ovih pravila kako bi se osiguralo sigurno, poštovano i ugodno okruženje za sve.Nakon pravila slijedi dokument CONTRIBUTION.md koji je bitno proučiti. U tom dokumentu su opisane smjernice za doprinos projektu koje govore na koji način preuzeti izvorni kôd projekta i kako primijeniti odrađene promjene na istom.

4.2.2. Postavljanje projekta

Da bismo riješili problem, potrebno je postaviti razvojno okruženje. Koristit će se dokumentacija aplikacije Twenty koja opisuje sve korake. Prvi korak je instaliranje potrebnih alata:

1. Git: Koristi se za verzioniranje i olakšava praćenje promjena u kôdu.
2. Node.js: Omogućuje izvršavanje JavaScript kôda na strani poslužitelja.
3. Yarn: Služi kao upravitelj paketima i ovisnostima u projektu.
4. nvm: Olakšava upravljanje verzijama Node.js-a.

Nakon što su instalirani svi potrebni alati projekt se klonira na naše računalo sa sljedećom naredbom:

git clone git@github.com:twentyhq/twenty.git

Zatim se pozicioniramo u korijenski direktorij projekta s naredbom cd twenty iz kojeg izvršavamo sve sljedeće naredbe:

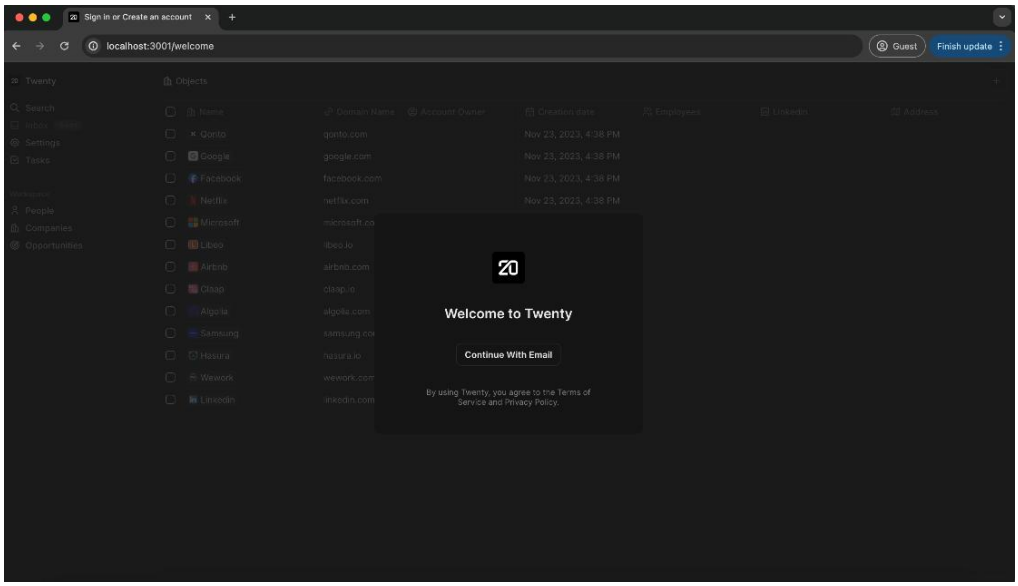
make postgres-on-macos-arm

Ova naredba instalira PostgresSql s osnovnom i testom bazom podataka. Zatim naredba za postavljanje varijabli okruženja koje je potrebno postaviti kroz .env datoteku:

cp ./packages/twenty-front/.env.example ./packages/twenty-front/.env

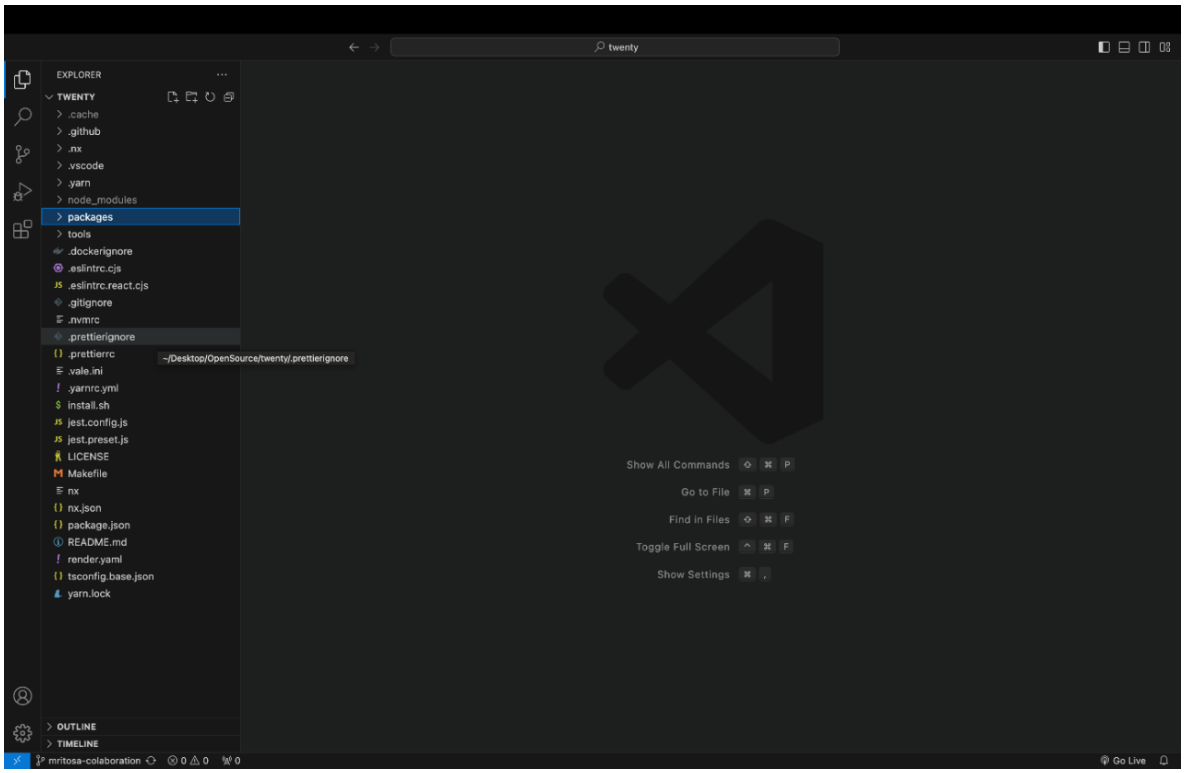
cp ./packages/twenty-server/.env.example ./packages/twenty-server/.env

Te na kraju je ostalo pokrenuti server i popuniti bazu podataka s početnim podacima uz pomoć naredbe yarn. Nakon što se prođu svi koraci, naredbom npx nx start se pokreće aplikacija kojoj se može pristupiti preko porta http://localhost:3001/.



Slika 8: Početni ekran aplikacije Twenty

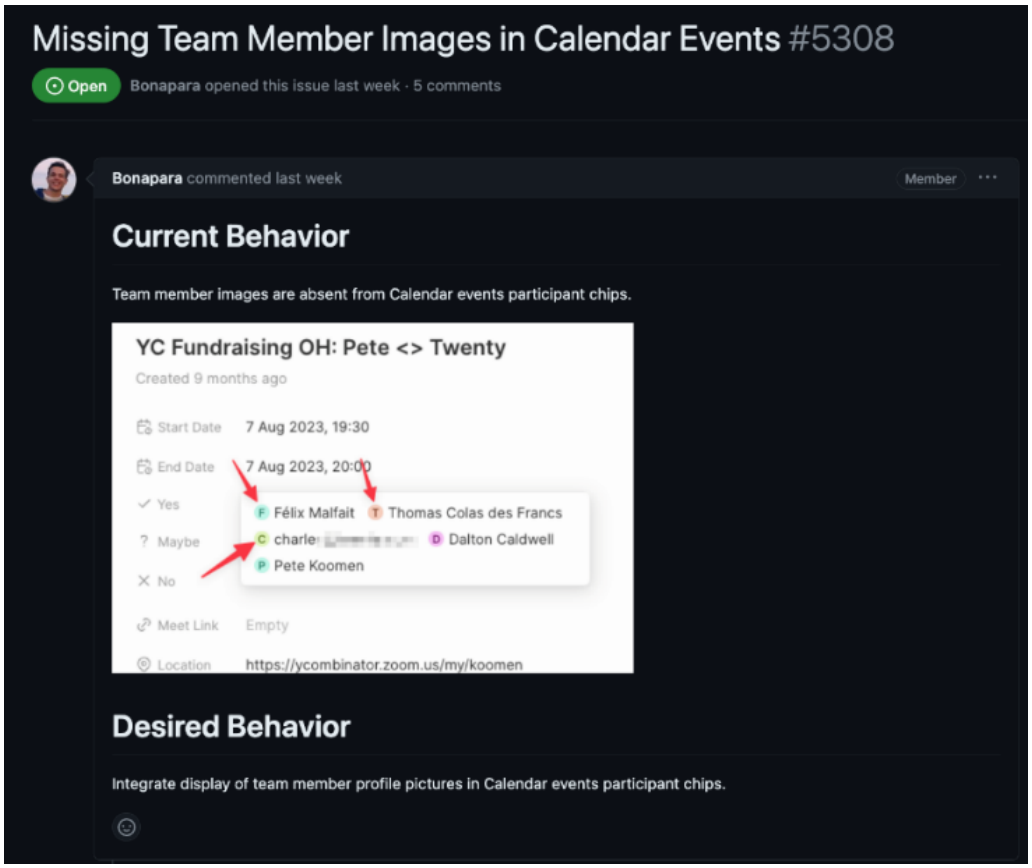
Za pisanje kôda koristit će se razvojno okruženje Visual Studio Code koji je ujedno i prijedlog od strane izvornog tima Twenty aplikacije. Nakon što se pokrene Visual Studio potrebno je otvoriti prethodno klonirani direktorij aplikacije. Na sljedećoj slici je prikazano razvojno okruženje spremno za rješavanje problema.



Slika 9: Prikaz razvojnog okruženja Visual Studio Code

4.2.3. Opis problema

Softversko rješenje Twenty ima veliki broj problema koji se trebaju riješiti. U ovom će primjeru biti prikazan problem koji se tiče frontend dijela aplikacije. Kao što već znamo Twenty je aplikacija za upravljanje odnosima s kupcima te s time ima i mogućnosti pregleda događaja i aktivnosti. Jedan korisnik aplikacije je prijavio problem koji se tiče prikaza profilnih slika sudionika u kalendarskom događaju.

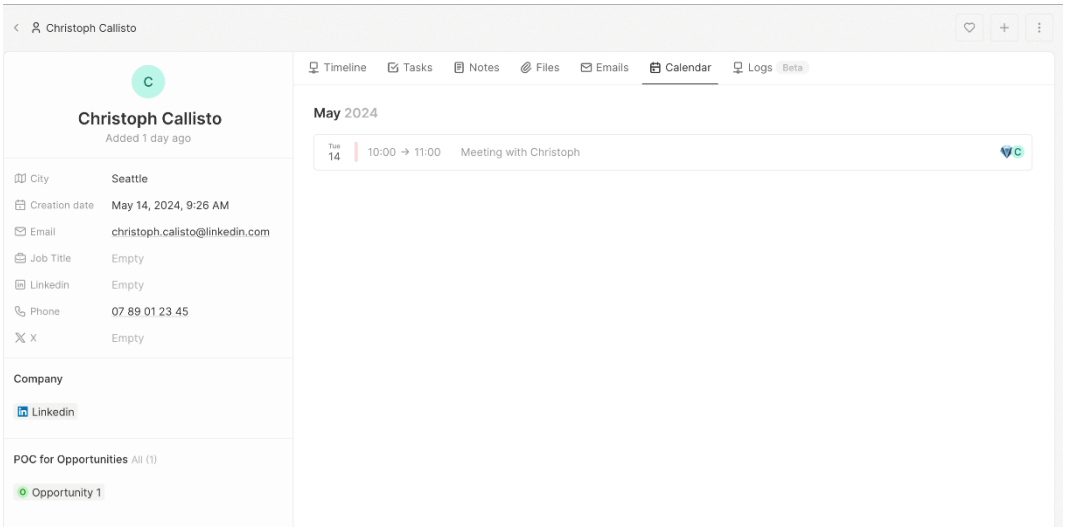


Slika 10: Opis problema s GitHub-a

Trenutni problem koji se javlja prilikom pristupa podacima određenog događaja u kalendaru sastoji se u nedostatku prikaza slika profila članova tima unutar kartice sudionika. Kada korisnici pristupaju informacijama o događaju, umjesto očekivanih slika profila, kartice sudionika prikazuju samo generirane profilne slike s početnim slovom sudionika. Kako bi se taj problem riješio potrebno je integrirati prikaz profilne slike sudionika.

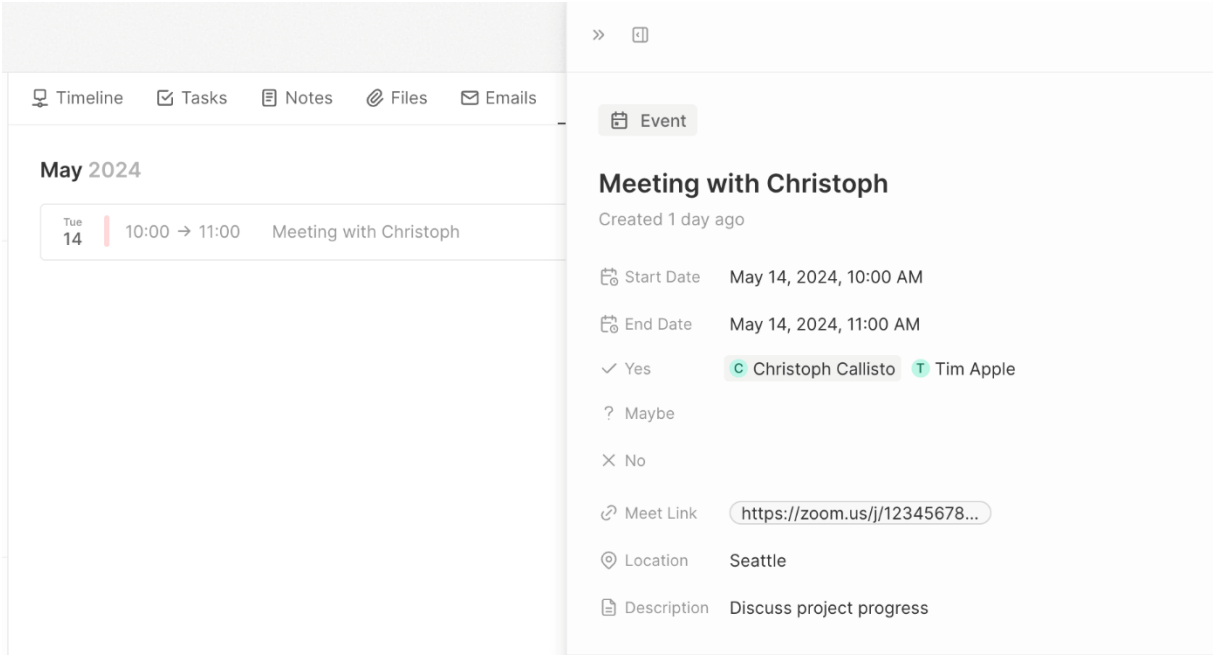
4.2.4. Repliciranje problema

Prije nego li se krene s rješavanjem problema, potrebno ga je replicirati kako bi se bolje razumio izvor problema. Prilikom ulaska u aplikaciju potrebno se pozicionirati na stranicu koja prikazuje događaje u kalendaru sudionika klikom na „Calendar“ unutar navigacije. Na sljedećoj se slici može vidjeti kako korisnik ima samo jedan zakazani događaj.



Slika 11: Ekran za prikaz događaja u kalendaru

Nakon što se klikne na njega, otvara se ladica s desne strane koja prikazuje sve podatke o odabranom događaju, gdje se može vidjeti kako opisani problem uistinu postoji.



Slika 12: Ladica za prikaz podataka događaja

4.2.5. Rješavanje problema

Prije nego li se izvorni kod pregleda i na njemu naprave promjene nužno je da se on nalazi na posebnoj feature grani kako bi se nesmetano moglo ispraviti grešku koja je nastala. Feature grana se kreira uz pomoć naredbe git branch {naziv-grane}, te nakon što je ona kreirana se s naredbom git checkout {naziv-grane} prebacujemo na nju. Prebacivanjem na feature granu nam omogućuje izmjenjivanje izvornog kôda aplikacije bez da na bilo koji način oštetimo kôd koji se nalazi na glavnoj grani.

Nakon što je problem repliciran i svi preduvjeti za rješavanje problema su ispunjeni, vrijeme je da se problem riješi. Prvi korak je pronalazak izvornog kôda koji je zaslužan za prikazivanje podataka sudionika u kalendarskom događaju. Uz pomoć tražilice unutar Visual Studio Codea pretražujemo sve datoteke koje sadrže ključnu riječ „participant“ kako bi pronašli datoteku zaslužnu za prikaz.

Pretragom je utvrđeno da se kôd odgovoran za prikaz nalazi u datoteci "ParticipantChip.tsx". Unutar datoteke definirana je funkcionalna komponenta "ParticipantChip" koja je zadužena za prikaz informacija o sudioniku aktivnosti, poput imena i avatara.

Komponenta ParticipantChip dohvaća tri svojstva koja su joj proslijeđena. U ovom slučaju to su „participant“, „variant“ i „className“. Što se tiče našeg problema svojstvo participant je najzanimljivije jer se iz njega izvlače potrebni podaci za prikaz profilne slike i imena sudionika.

```
export const ParticipantChip = ({
  participant,
  variant = 'default',
  className,
}: {
  participant: any;
  variant?: ParticipantChipVariant;
  className?: string;
}) => {
  const { person, workspaceMember } = participant;

  const displayName = getDisplayNameFromParticipant({
    participant,
    shouldUseFullName: true,
  });
```

Nakon što su potencijalni URL-ovi avatara određeni, varijabla "avatarUrl" se popunjava odabirom prve ne-null vrijednosti. Ovo osigurava da će varijabla "avatarUrl" sadržavati pravi URL avatara koji je dostupan, bilo da pripada objektu "person" ili "workspaceMember".

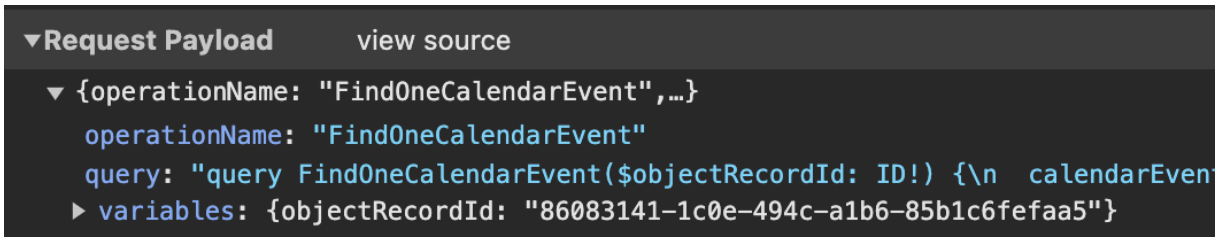
```
      const avatarUrl = person?.avatarUrl ?? workspaceMember?.avatarUrl
?? '';
```

Kada se dohvati avatarov URL on se prosljeđuje funkciji *getImageAbsoluteURIOrBase64* koja provjerava dobiveni URL i vrši odgovarajuće transformacije kako bi osigurala da konačni URL bude apsolutan i upotrebljiv u aplikaciji.

```
export const getImageAbsoluteURIOrBase64 = (imageUrl?: string | null) =>
{
  if (!imageUrl) {
    return null;
  }

  if (imageUrl?.startsWith('data:') || imageUrl?.startsWith('https:')) {
    return imageUrl;
  }

  const serverFilesUrl = REACT_APP_SERVER_BASE_URL;

  return `${serverFilesUrl}/files/${imageUrl}`;
};
```

Iako su sve metode unutar komponente ParticipantChip dobro implementirane problem ne leži u njima već u dohvaćenim podacima sudionika. Ispisom podataka sudionika u konzoli utvrđeno se da se ne dohvaćaju podaci objekata „Person“ i „WorkspaceMember“ u kojim se nalazi svojstvo „avatarURL“. S time se može zaključiti kako je problem u logici dohvaćanja podataka sa servera.

Kako bi se pronašao zahtjev koji se šalje potrebno je otići u „Network“ karticu koja je dostupna unutar alata za developere na našem pregledniku. Prilikom snimanja mrežnog prometa može se vidjeti kako se za dohvaćanje pokreće metoda *FindOneCalendarEvent*.



Slika 13: Varijable zahtjeva

Nakon pronalaska metode koja je odgovorna za dohvat podataka, utvrđeno je da se metoda poziva unutar „RightDrawerCalendarEvent“ komponente. Komponenta dohvaća podatke korištenjem metode useFindOneRecord za dohvaćanje podataka o kalendarskom događaju. Prvo se dohvaća ID kalendarskog događaja iz stanja aplikacije. Zatim se koristi dobiveni ID kako bi se pronašao odgovarajući zapis kalendarskog događaja. Ako je događaj pronađen, prikazuju se detalji tog događaja.

```
 export const RightDrawerCalendarEvent = () => {
const { setRecords } = useSetRecordInStore();
const viewableCalendarEventId = useRecoilValue(viewableCalendarEventIdState);
const { record: calendarEvent } = useFindOneRecord<CalendarEvent>({
  objectNameSingular: CoreObjectNameSingular.CalendarEvent,
  objectRecordId: viewableCalendarEventId ?? '',
  onCompleted: (record) => setRecords([record]),
});

if (!calendarEvent) return null;

return <CalendarEventDetails calendarEvent={calendarEvent} />;
};
```

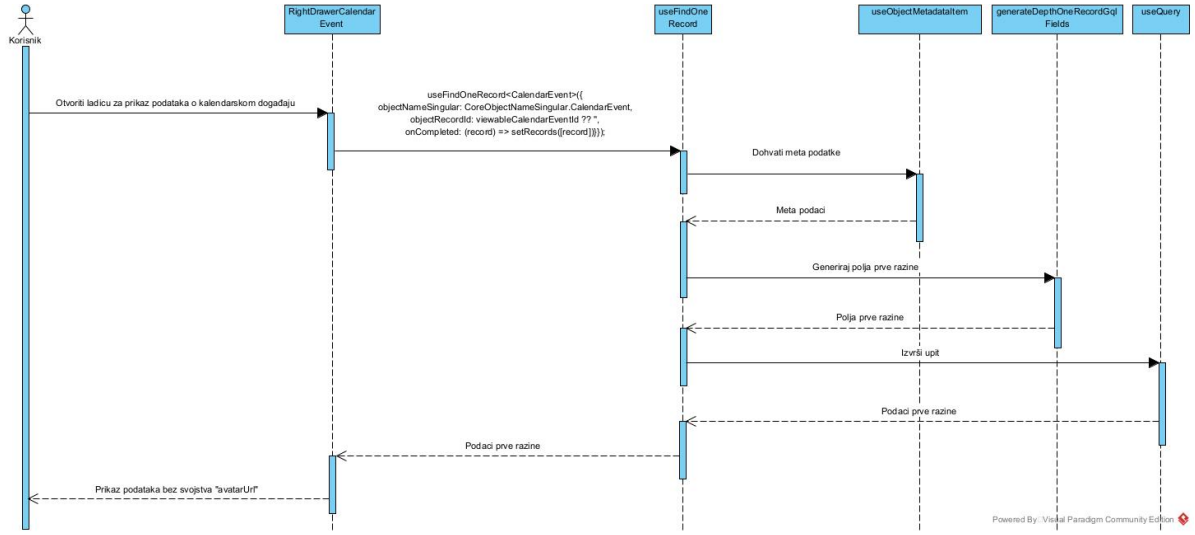
Pošto metoda FindOneRecord izvršava upit bitno je razumjeti kako radi, kako bi se mogao utvrditi problem. Ova metoda funkcionira tako da dohvaća pojedinačni zapis iz baze na temelju naziva objekta i ID-a. Prilikom poziva te metode, prosljeđuju mu se parametri koji sadrže naziv objekta (objectNameSingular), ID zapisa (objectRecordId), polja GraphQL upita za dohvat podataka o zapisu (recordGqlFields), funkciju koja se poziva nakon uspješnog dohvata podataka (onCompleted), te opcionalni parametar skip koji označava treba li preskočiti izvršavanje upita. Kada funkcija primi sve potrebne parametre prvo što radi je da dohvaća meta podatke na temelju parametra *objectNameSingular uz pomoć metode useObjectMetadataItem. Nakon što dohvati sve meta podatke komponenta s metodom computedRecordGqlFields* provjerava je li proslijeđen parametar *recordGqlFields.* U slučaju

da parametar nije proslijeđen komponenta s pomoću metode *generateDepthOneRecordGqlFields* generira polja za dohvaćanje i sprema ih u varijablu *computedRecordGqlFields.* Konačno kada se obrade svi parametri izvršava se upit metodom *useQuery.*

```
export const useFindOneRecord = <T extends ObjectRecord =
ObjectRecord>({
  objectNameSingular,
  objectRecordId = '',
  recordGqlFields,
  onCompleted,
  skip,
}: ObjectMetadataItemIdentifier & {
  objectRecordId: string | undefined;
  recordGqlFields?: RecordGqlOperationGqlRecordFields;
  onCompleted?: (data: T) => void;
  skip?: boolean;
}) => {
  const { objectMetadataItem } = useObjectMetadataItem({
    objectNameSingular,
  });

  const computedRecordGqlFields =
    recordGqlFields ?? generateDepthOneRecordGqlFields({
objectMetadataItem });

  const { findOneRecordQuery } = useFindOneRecordQuery({
    objectNameSingular,
    recordGqlFields: computedRecordGqlFields,
  });

  const { data, loading, error } = useQuery<{
    [nameSingular: string]: RecordGqlNode;
  }>(findOneRecordQuery, {
    skip: !objectMetadataItem || !objectRecordId || skip,
    variables: { objectRecordId },
    onCompleted: (data) => {
      const recordWithoutConnection = getRecordFromRecordNode<T>({
        recordNode: { ...data[objectNameSingular] },
      });
```

```
      if (isDefined(recordWithoutConnection)) {
        onCompleted?.(recordWithoutConnection);
      }
    },
  });
```

Kada je kôd funkcije useFindOneQuery jasniji moguće je zaključiti da je problem nastao prilikom generiranja polja graphQL upita. Naime, pošto komponenti nisu proslijeđena točno određena polja koja treba dohvatiti, komponenta je generirala polja samo prve razine čime je izostalo polje „avatarUrl“. Na slici 14 se može vidjeti dijagram koji prikazuje gdje je točno došlo do problema.



Slika 14: Dijagram slijeda problema u softveru

Kako bi se taj problem riješio potrebno je iskoristiti varijablu recordGqlFields te s njom redefinirati selektore postojećeg upita unutar RightDrawerCalendarEvent komponente.

Konačno rješenje bi izgledalo ovako:

```
const recordGqlFields = {
  conferenceLink: true,
  description: true,
  endsAt: true,
  externalCreatedAt: true,
  id: true,
  isCanceled: true,
  isFullDay: true,
```

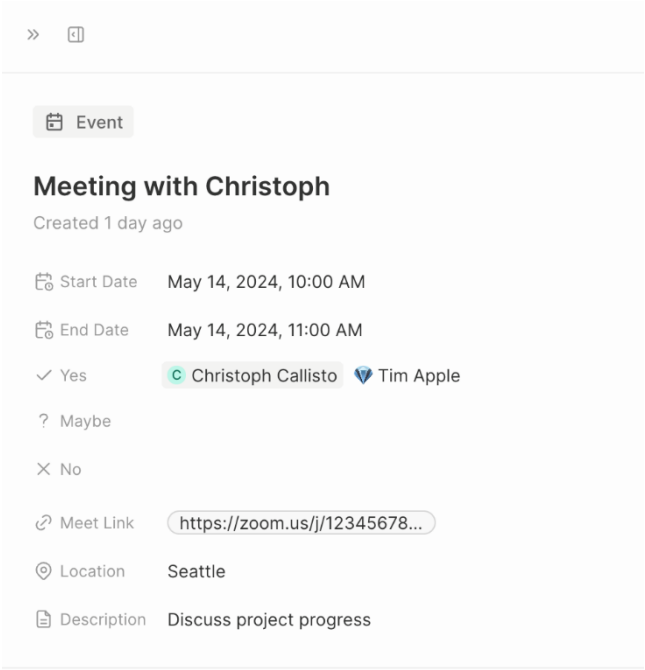
```
    location: true,
    startsAt: true,
    title: true,
    visibility: true,
    calendarEventParticipants: {
      id: true,
      person: true,
      workspaceMember: true,
      isOrganizer: true,
      responseStatus: true,
      handle: true,
      createdAt: true,
      calendarEventId: true,
      updatedAt: true,
      displayName: true,
    },
  };

  export const RightDrawerCalendarEvent = () => {
    const { setRecords } = useSetRecordInStore();
    const viewableCalendarEventId =
  useRecoilValue(viewableCalendarEventIdState);
    const { record: calendarEvent } = useFindOneRecord<CalendarEvent>({
      objectNameSingular: CoreObjectNameSingular.CalendarEvent,
      objectRecordId: viewableCalendarEventId ?? '',
      recordGqlFields: recordGqlFields,
      onCompleted: (record) => setRecords([record]),
    });

    if (!calendarEvent) return null;

    return <CalendarEventDetails calendarEvent={calendarEvent} />;
  };
```

Unutar metode je dodana varijabla recordGqlFields u kojoj su definirani selektori za dohvaćanje podataka. Kao što se vidi u priloženom kôdu recordGqlFields u ovom slučaju sadrži objekte „Person“ i „WorkspaceMember“ koji su bitni za prikaz profilne slike sudionika na kalendarskom događaju. Nakon što je varijabla definirana ona se prosljeđuje metodi *useFindOneRecord.* U konačnici odgovor zahtjeva sada izgleda ovako:

```
"__typename": "CalendarEventParticipantEdge",
"node": {
    "__typename": "CalendarEventParticipant",
    "id": "e1ab9e1b-df6e-438e-a788-11c96dcecdd3",
    "person": null,
    "isOrganizer": false,
    "responseStatus": "ACCEPTED",
    "handle": "tim@apple.com",
    "createdAt": "2024-05-14T07:26:20.901Z",
    "calendarEventId": "86083141-1c0e-494c-a1b6-85b1c6fefaa5",
    "updatedAt": "2024-05-14T07:26:20.901Z",
    "displayName": "Tim Apple",
    "workspaceMember": {
        "__typename": "WorkspaceMember",
        "createdAt": "2024-05-14T07:26:20.901Z",
        "userEmail": "tim@apple.dev",
        "colorScheme": "Light",
        "userId": "20202020-9e3b-46d4-a556-88b9ddc2b034",
        "avatarUrl": "profile-picture/original/0a641ad8-bf79-49e7-aca5-f96ab12f1946.png",
        "locale": "en",
        "updatedAt": "2024-05-14T07:29:22.327Z",
        "id": "20202020-0687-4c41-b707-ed1bfca972a7",
        "name": {
            "__typename": "FullName",
            "firstName": "Tim",
            "lastName": "Apple"
        }
    }
}
```

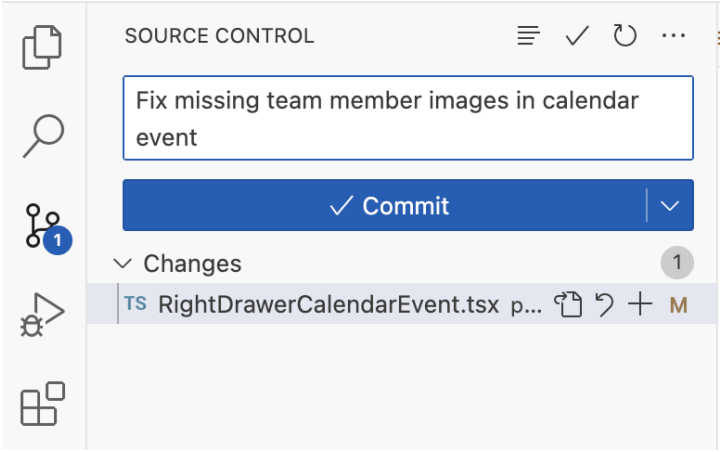

Slika 15: Odgovor zahtjeva

Pošto je sada omogućeno da zahtjev vraća sve potrebne podatke za prikaz, komponenta *ParticipantChip* ima sve potrebne resurse kako bi prikazala profilnu sliku sudionika. Vratimo li se natrag u aplikaciju vidljivo je da su sada profilne slike sudionika vidljive.



Slika 16: Kartica za prikaz podataka o događaju nakon riješenog problema

Sve promjene u kodu su potom commitane putem razvojnog okruženja Visual Studio Code kao što je vidljivo na sljedećoj slici. Na slici se može vidjeti kako izgleda commit i datoteke na kojoj su napravljene promjene.

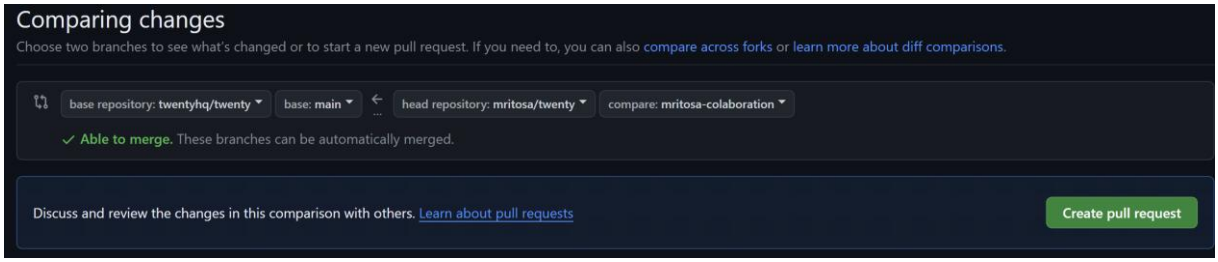


Slika 17: Prikaz commit-a

Nakon što su promjene uspješno pohranjene potrebno je pushati cijelu feature granu na naš forkani repozitorij koji se nalazi na GitHub platformi. To je moguće uz pomoć naredbe git push {naziv-grane}. Ovime se omogućilo da se promjene koje su napravljene u kodu mogu predati na pregled.

4.2.6. Predaja rješenja

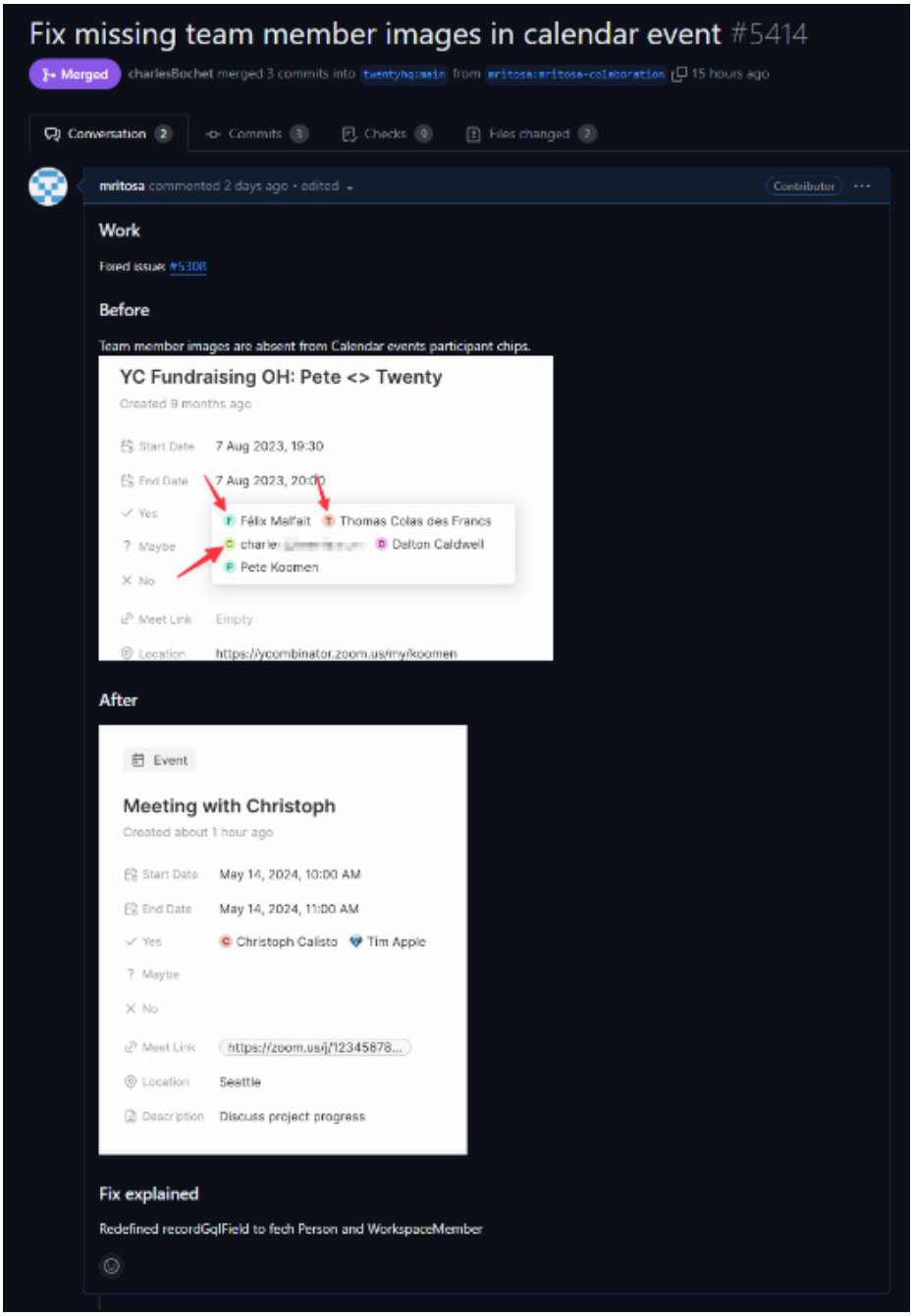
Rješenje se predaje tako da se kreira se pull request putem GitHub platforme. Odabire se grana na kojoj su promjene napravljene u ovom slučaju „mritosa-colaboration“ i glavna, „main“ grana aplikacije Twenty.



Slika 18: Odabir grana za spajanje

Klikom na gumb „Create pull request“ otvara se ekran za unos opisa promjena koje su napravljene na aplikaciji. Opis sadrži sljedeće:

- Naslov Pull Requesta
- Broj Issuea koji je riješen
- Kako je izgledao problem prije i nakon što je riješen
- Objašnjenje rješenja



Slika 19: Opis Pull Requesta

Kada je Pull Request uspješno predan, prolazi kroz proces pregleda (code review). Jedan od glavnih programera provjerava ispravnost promjena nakon čega daje i povratnu

informaciju o njima. U našem slučaju predano rješenje je bilo dobro što je rezultiralo spajanjem naše grane s glavnom granom. Spajanje grana predstavlja i zadnji korak u procesu kolaboracije nakon čega su promjene vidljive svim ostalim članovima razvojnog tima.

Slika 20: Poruka o uspješnom spajanju grana

5. Zaključak

Povećana popularnost softvera otvorenog kôda označava promjenu u načinu na koji se razvijeni softver dijeli i poboljšava. Ovaj način razvoja softvera je usmjeren ka pružanju pristupa izvornom kôdu i kolaboraciji koji rezultiraju povećanjem inovacija, poboljšanom sigurnošću i prilagodljivošću softverskih rješenja. Primjeri kao što su Linux, Git i Mozilla Firefox pokazuju uspjeh projekata otvorenog kôda u ispunjavanju visokih standarda kvalitete i stjecanju priznanja.

Stvaranje softvera otvorenog kôda uključuje faze koje se kreću od programiranja do održavanja i poboljšanja kroz suradnju s korisnicima i programerima. Osnovni alati kao što su sustavi kontrole verzija, mehanizmi za praćenje grešaka i platforme kao što je GitHub ključni su za upravljanje tim projektima. Osim pregleda, licenciranje je ključno za zaštitu prava kreatora i korisnika kako bi se osigurala održivost pristupa otvorenom kôdu.

Glavna prednost razvoja otvorenog kôda leži u povećanju sigurnosti softvera. Međutim, iako se često navodi kao prednost, u određenim se situacijama može smatrati i nedostatkom. Čineći izvorni kôd otvoreno dostupnim, veća zajednica programera može ga pregledati radi slabosti što rezultira sigurnijim softverskim aplikacijama, ali isto tako, otvorenost koda može omogućiti zlonamjernim korisnicima da iskoriste slabosti u vlastitu korist.

Nadalje, softver otvorenog kôda potiče kreativnost omogućujući programerima da unaprijede postojeće projekte i prilagode rješenja zahtjevima. Ovaj kontinuirani ciklus poboljšanja i dijeljenja ubrzava napredak i uspostavlja raznoliku mrežu međusobno povezanih alata i aplikacija. Kolaborativna bit razvoja otvorenog kôda uklanja ulazne prepreke dopuštajući sudionicima da se pridruže s njihovim idejama i vještinama.

Praktična strana ovog rada pokazuje na koji način se teorijski koncepti provode u praksi u scenarijima stvarnog svijeta. Također nadahnjuje pojedince da se uključe u zajednički rad na stvaranju i poboljšanju softvera. Kroz sudjelovanje u projektima otvorenog kôda, programeri stječu stručnost, usavršavaju svoje programerske sposobnosti i razvijaju portfelj koji je cijenjen u tom području.

Ovime možemo zaključiti kako je razvoj softvera otvorenog kôda postao neizostavni dio softverske industrije i kako se njegova filozofija treba sve više primjenjivati kako bi softverska rješenja bila što kvalitetnija i korisnija za svoje korisnike.

Popis literature

[1] C. DiBona i S. Ockman, *Open Sources: Voices from the Open Source Revolution*. O'Reilly Media, Inc., 1999.

[2] Y. Ye i K. Kishida, „Toward an understanding of the motivation of open source software developers", u *25th International Conference on Software Engineering, 2003. Proceedings.*, svi. 2003, str. 419–429. doi: 10.1109/ICSE.2003.1201220.

[3] M. Heron, V. L. Hanson, i I. Ricketts, „Open source and accessibility: advantages and limitations", *J Interact Sci*, sv. 1, izd. 1, str. 2, svi. 2013, doi: 10.1186/2194-0827-1-2.

[4] J. G. MacKinnon, „Review of The Linux Operating System: Debian GVU/Linux", *Journal of Applied Econometrics*, sv. 14, izd. 4, str. 443–452, 1999.

[5] „What is Linux kernel?", Educative. Pristupljeno: 04. svibanj 2024. [Na internetu]. Dostupno na: https://www.educative.io/answers/what-is-linux-kernel

[6] D. Spinellis, „Git", *IEEE Software*, sv. 29, izd. 3, str. 100–101, svi. 2012, doi: 10.1109/MS.2012.61.

[7] „Firefox | Definition, History, & Facts | Britannica". Pristupljeno: 04. svibanj 2024. [Na internetu]. Dostupno na: https://www.britannica.com/technology/Firefox-Web-browser

[8] „What is Firefox? | Definition from TechTarget", WhatIs. Pristupljeno: 04. svibanj 2024. [Na internetu]. Dostupno na: https://www.techtarget.com/whatis/definition/Firefox

[9] M. Ballhausen, „Free and Open Source Software Licenses Explained", *Computer*, sv. 52, izd. 6, str. 82–86, lip. 2019, doi: 10.1109/MC.2019.2907766.

[10] A. M. S. Laurent, *Understanding Open Source and Free Software Licensing: Guide to Navigating Licensing Issues in Existing & New Software*. O'Reilly Media, Inc., 2004.

[11] R. Sen, C. Subramaniam, i M. L. Nelson, „Determinants of the Choice of Open Source Software License", *Journal of Management Information Systems*, sv. 25, izd. 3, str. 207–240, pros. 2008, doi: 10.2753/MIS0742-1222250306.

[12] „The Open Source Definition", Open Source Initiative. Pristupljeno: 07. travanj 2024. [Na internetu]. Dostupno na: https://opensource.org/osd

[13] N. Ruparelia, „Software development lifecycle models", *ACM SIGSOFT Software Engineering Notes*, sv. 35, str. 8–13, svi. 2010, doi: 10.1145/1764810.1764814.

[14] M. Saini i K. Kaur, „A Review of Open Source Software Development Life Cycle Models“, 2014. Pristupljeno: 29. travanj 2024. [Na internetu]. Dostupno na: https://www.semanticscholar.org/paper/A-Review-of-Open-Source-Software-Development-Life-Saini-Kaur/198e62d2d305bc23b0e8101a36a13cf875e001e1

[15] „The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary“, *Choice Reviews Online*, sv. 39, izd. 05, str. 39-2841-39–2841, sij. 2002, doi: 10.5860/CHOICE.39-2841.

[16] V. Tiwari, „Software Engineering Issues in Development Models of Open Source Software“, *International Journal of Computer Science and Technology, IJCST*, sv. 2, str. 38–44, lip. 2011.

[17] „The Open Source Development Model: Overview, Benefits and Recommendations“. Pristupljeno: 02. svibanj 2024. [Na internetu]. Dostupno na: https://aaaea.org/Al-muhandes/2008/February/open_src_dev_model.htm

[18] M. W. Wu i Y.-D. Lin, „Open source software development: An overview“, *Computer*, sv. 34, str. 33–38, lip. 2001, doi: 10.1109/2.928619.

[19] A. Capiluppi i M. Michlmayr, *From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects*, sv. 234. 2007. doi: 10.1007/978-0-387-72486-7_3.

[20] N. N. Zolkifli, A. Ngah, i A. Deraman, „Version Control System: A Review“, *Procedia Computer Science*, sv. 135, str. 408–415, sij. 2018, doi: 10.1016/j.procs.2018.08.191.

[21] C. Rodriguez-Bustos i J. Aponte, „How Distributed Version Control Systems impact open source software projects“, u *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, Zurich: IEEE, lip. 2012, str. 36–39. doi: 10.1109/MSR.2012.6224297.

[22] „Analysis of Bug Tracking Tools“. Pristupljeno: 26. travanj 2024. [Na internetu]. Dostupno na: https://www.ijser.org/paper/Analysis-of-Bug-Tracking-Tools.html

[23] „What is a DevOps platform?“ Pristupljeno: 10. srpanj 2024. [Na internetu]. Dostupno na: https://about.gitlab.com/topics/devops-platform/

[24] „What Is GitHub? | Definition from TechTarget“, IT Operations. Pristupljeno: 26. travanj 2024. [Na internetu]. Dostupno na: https://www.techtarget.com/searchitoperations/definition/GitHub

[25] „Features | GitLab“. Pristupljeno: 10. srpanj 2024. [Na internetu]. Dostupno na: https://about.gitlab.com/features/

[26]„Tools for Managing Open Source Programs“. Pristupljeno: 24. travanj 2024. [Na internetu]. Dostupno na: https://www.linuxfoundation.org/resources/open-source-guides/tools-managing-open-source-programs

[27]T. Beuzen, „The Git Fork-Branch-Pull Workflow“, Tomas Beuzen. Pristupljeno: 08. srpanj 2024. [Na internetu]. Dostupno na: https://www.tomasbeuzen.com/post/git-fork-branch-pull/

[28]„About collaborative development models“, GitHub Docs. Pristupljeno: 08. srpanj 2024. [Na internetu]. Dostupno na: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/getting-started/about-collaborative-development-models

[29]„What is Code Review?“, BrowserStack. Pristupljeno: 09. srpanj 2024. [Na internetu]. Dostupno na: https://browserstack.wpengine.com/guide/what-is-code-review/

[30]„How to Contribute to Open Source“, Open Source Guides. Pristupljeno: 09. srpanj 2024. [Na internetu]. Dostupno na: https://opensource.guide/how-to-contribute/

[31]„Twenty - The #1 Open-Source CRM“. Pristupljeno: 14. svibanj 2024. [Na internetu]. Dostupno na: https://www.twenty.com/

Popis slika

Slika 1: Životni ciklus tradicionalnog razvoja softvera 12
Slika 2: Životni ciklus razvoja softvera otvorenog kôda 13
Slika 3: „Jorgensenov“ razvojni model 14
Slika 4: Wu i Linov razvojni model 15
Slika 5: Andrea Capiluppi i Martin Michlmayrov razvojni model 16
Slika 6: Fork & Pull model[26] 21
Slika 7: Aplikacija Twenty [24] 24
Slika 8: Početni ekran aplikacije Twenty 26
Slika 9: Prikaz razvojnog okruženja Visual Studio Code 27
Slika 10: Opis problema s GitHub-a 28
Slika 11: Ekran za prikaz događaja u kalendaru 29
Slika 12: Ladica za prikaz podataka događaja 29
Slika 13: Varijable zahtjeva 32
Slika 14: Dijagram slijeda problema u softveru 34
Slika 16: Odgovor zahtjeva 36
Slika 17: Kartica za prikaz podataka o događaju nakon riješenog problema 36
Slika 18: Prikaz commit-a 37
Slika 19: Odabir grana za spajanje 37
Slika 20: Opis Pull Requesta 38
Slika 21: Poruka o uspješnom spajanju grana 39

Popis tablica

Tablica 1: Motivacija iza otvorenog kôda....4
Tablica 2: Prednosti i nedostaci softvera otvorenog koda....5
Tablica 3: Usporedba najpopularnijih licenci....10