**SVEUČILIŠTE ALGEBRA**

**Programsko inženjerstvo**

ZAVRŠNI RAD

**Razvoj multifunkcionalnog sustava za unaprjeđenje rada i centralizaciju podataka studentskih udruga**

Edi Graovac

Zagreb, kolovoz, 2024.

**SVEUČILIŠTE ALGEBRA**

**Programsko inženjerstvo**

ZAVRŠNI RAD

**Razvoj multifunkcionalnog sustava za unaprjeđenje rada i centralizaciju podataka studentskih udruga**

Edi Graovac

Mentor: Daniel Bele

Zagreb, kolovoz, 2024.

# Predgovor

Zahvaljujem Sveučilištu Algebra i mentoru Danielu Beleu, struč. spec. ing. comp., na stručnom vodstvu i prenesenom znanju tijekom pohađanja preddiplomskog studija te na pomoći pri odabiru teme i izradi završnog rada.

Također, zahvaljujem se i ostalim profesorima na znanju koje su nam prenijeli, na prihvaćanju studenata kao hvalevrijednih budućih kolega, na kontinuiranom bodrenju da se krene dalje i dostigne više.

Zahvaljujem se i kolegama iz studentske udruge eSTUDENT koji su me podržali i poticali da odaberem ovu temu i izradim je do uporabnog stanja.

**Sažetak**

Završnim radom prikazuje se razvoj sustava za organizaciju i praćenje rada članova udruge eSTUDENT. Cilj je ovog sustava olakšati rad članovima udruge kroz digitalizaciju procesa prikupljanja podataka. Potreba za digitalizacijom proizašla je iz značajnog povećanja broja volonterskih projekata koje Udruga organizira, što je rezultiralo i rastom broja članova.

Kako bi sustav bio što uspješniji, prilagođen je specifičnim pravilnicima unutar Udruge. Zbog tih specifičnosti, standardna programska rješenja nisu bila adekvatna za njihove potrebe. Sustav je razvijen s mogućnošću prilagodbe različitim korisničkim pravima, čime će se omogućiti pregled podataka prema korisnikovom položaju u Udruzi. Time će Udruga moći lakše pratiti aktivnost svojih članova, olakšavajući praćenje uvjeta za certifikate koje pojedini korisnik treba ispuniti.

Nadalje, sustav će značajno ubrzati proces izbora unutar Udruge. Kroz digitalnu platformu, članovi će moći glasovati u skladu s pravilnikom, čime će se ubrzati donošenje odluka na skupštinama. Također, baza podataka o partnerima bit će mnogo organiziranija u usporedbi s trenutnom Excel datotekom koja se koristi. To će omogućiti lakšu izmjenu i filtriranje podataka te će se izbjeći korištenje nepotrebnih podataka.

Razvoj sustava omogućit će trajno gašenje zastarjelih aplikacija na serveru, što će rezultirati uštedom novčanih sredstava.

**Ključne riječi:** sustav, članovi udruge, digitalizacija, eSUSTAV, aktivnosti, certifikati, partneri, izbori, administracija.

# Abstract

The final project presents the development of a system for organizing and monitoring the work of members of the eSTUDENT association. The goal of this system is to facilitate the work of the association's members by digitizing the data collection processes. The need for digitization arose due to a significant increase in the number of volunteer projects organized by the association, which has also led to an increase in the number of members.

To ensure the system's success, it closely follows specific regulations within the association. Due to these specific requirements, off-the-shelf software solutions were not suitable for their needs. The system is designed with the ability to adapt to different user permissions, allowing data to be viewed according to the user's role within the association. This will enable the association to better track member activity, making it easier to monitor the conditions for obtaining certificates that individual users need to fulfill.

Furthermore, the system will significantly speed up the election process within the association. Through a digital platform, members will be able to vote in accordance with the regulations, thereby speeding up decision-making during assemblies. Additionally, the database of partners will be much more organized compared to the current Excel file used by the association. This will allow for easier data modification and filtering, while unnecessary data will be avoided.

The development of the system will allow for the permanent shutdown of outdated applications on the server, resulting in cost savings.

**Keywords**: system, association members, digitalization, eSUSTAV, activities, certificates, partners, elections, administration.

# Sadržaj

[1. Uvod 1](#_Toc174875926)

[2. Opis problema decentraliziranih podataka i prijedlog rješenja 3](#_Toc174875927)

[2.1. Analiza problema 3](#_Toc174875928)

[2.2. Prijedlog i opis rješenja 3](#_Toc174875929)

[2.3. Usporedba s postojećim rješenjima 5](#_Toc174875930)

[2.3.1. Analiza Salesforce Nonprofit Cloud alata 5](#_Toc174875931)

[2.3.2. Analiza Blackbaud alata 6](#_Toc174875932)

[2.3.3. Analiza Wild Apricot alata 7](#_Toc174875933)

[2.3.4. Zaključak analize 8](#_Toc174875934)

[2.4. Usklađivanje ideje s pravilnicima Udruge 9](#_Toc174875935)

[3. Arhitektura programskoga multifunkcionalnog sustava 10](#_Toc174875936)

[3.1. Skica i opis arhitekture sustava 10](#_Toc174875937)

[3.2. Korištene tehnologije 11](#_Toc174875938)

[3.2.1. PostgreSQL 11](#_Toc174875939)

[3.2.2. Django REST framework 12](#_Toc174875940)

[3.2.3. React 14](#_Toc174875941)

[3.2.4. Docker 16](#_Toc174875942)

[3.2.5. NGINX 17](#_Toc174875943)

[3.3. Verzioniranje kôda 19](#_Toc174875944)

[4. Implementacija multifunkcionalnog sustava 20](#_Toc174875945)

[4.1. Izrada i konfiguracija baze podataka 20](#_Toc174875946)

[4.2. Implementacija Google OAuth2 autentifikacije 27](#_Toc174875947)

[4.3. Izrada poslužiteljskog sloja 28](#_Toc174875948)

[4.3.1. Postavljanje poslužiteljskog sloja 30](#_Toc174875949)

[4.3.2. Podatkovni sloj unutar poslužiteljskog sloja 34](#_Toc174875950)

[4.3.3. Izrada poslovne logike unutar poslužiteljskog sloja 36](#_Toc174875951)

[4.3.4. Postavljanje Django administratorske stranice unutar poslužiteljskog sloja 42](#_Toc174875952)

[4.4. Izrada korisničkog sučelja sustava 44](#_Toc174875953)

[4.4.1. Postavljanje korisničkog sučelja sustava 45](#_Toc174875954)

[4.4.2. Dozvole korisnika kroz korisničko sučelje 46](#_Toc174875955)

[4.4.3. Konfiguracija komunikacije korisničkog sučelja i poslužiteljskog sloja 51](#_Toc174875956)

[4.4.4. Vizualni identitet korisničkog sučelja 54](#_Toc174875957)

[4.5. Korištenje Dockera za virtualizaciju modula sustava 56](#_Toc174875958)

[5. Provođenje ankete kao analiza uspješnosti programskog rješenja 60](#_Toc174875959)

[Zaključak 62](#_Toc174875960)

[Popis kratica 64](#_Toc174875961)

[Popis slika 65](#_Toc174875962)

[Popis tablica 66](#_Toc174875963)

[Popis kôdova 67](#_Toc174875964)

[Literatura 68](#_Toc174875965)

# Uvod

U današnjem dinamičnom okruženju, organizacije se suočavaju s izazovima upravljanja velikom količinom podataka, a učinkovita upravljanje tim podacima postaje ključno za uspješno funkcioniranje bilo koje organizacije. Udruga eSTUDENT, kao jedna od vodećih studentskih udruga u Hrvatskoj, koja okuplja talentirane studente s ciljem razvoja njihovih profesionalnih vještina, nije iznimka. Kako bi članovi Udruge mogli što bolje koristiti svoje vrijeme i resurse, a ujedno osigurati transparentnost i učinkovitost, neophodno je razviti sustav koji će omogućiti lako praćenje i upravljanje radom svakog člana.

Jedan od ključnih izazova s kojima se Udruga eSTUDENT suočila jest upravo organizacija i praćenje rada članova unutar različitih timova, kao i njihovo ispunjavanje uvjeta za dobivanje certifikata. Certifikati su važan motivacijski alat, ali i priznanje za uložen trud i angažman. Međutim, trenutni sustavi, koji se koriste za praćenje aktivnosti, poput zasebnih aplikacija i velikih Google Sheet tablica, pokazali su se neučinkovitima. Podaci su često bili fragmentirani, teško dostupni i podložni ljudskim pogreškama. Članovi nisu imali jasan uvid u svoje aktivnosti, što je dodatno opterećivalo voditelje timova, koji su morali provoditi značajno vrijeme na informiranje članova o njihovom napretku. Ovi problemi ne samo što utječu na učinkovitost rada unutar Udruge već i na motivaciju članova, koji se mogu osjećati nezadovoljno zbog nedostatka transparentnosti i nepristupačnosti informacija o njihovom doprinosu.

Najveći izazovi u rješavanju ove problematike su centralizacija podataka i stvaranje sustava koji će biti dovoljno fleksibilan da podrži različite potrebe različitih timova unutar Udruge, a istovremeno jednostavan za korištenje kako bi ga članovi mogli koristiti bez dodatnog opterećenja. Potrebno je razviti rješenje koje će integrirati sve relevantne informacije na jednome mjestu, omogućiti brzu i jednostavnu analizu podataka te pružiti transparentan uvid članovima u njihove aktivnosti.

Ovaj je završni rad usmjeren na rješavanje navedene problematike razvojem novog programskog rješenja koje će omogućiti centralizirano praćenje aktivnosti članova, automatizirati procese prikupljanja i obrade podataka te time olakšati rad svima unutar Udruge. Kroz ovaj rad, detaljno će se analizirati trenutni problemi u organizaciji podataka, prikazati arhitektura predloženog rješenja te koraci njegove implementacije, s ciljem osiguravanja učinkovitijeg i transparentnijeg sustava upravljanja radom članova eSTUDENT-a.

# Opis problema decentraliziranih podataka i prijedlog rješenja

U ovom su poglavlju predstavljeni problemi trenutnog načina sakupljanja podataka, predlaže se programsko rješenje. Istražuju se postojeća rješenja te se predlaže novo koje rješava nedostatke i opisanu problematiku.

## Analiza problema

Udruga eSTUDENT svake godine organizira više od dvadeset volonterskih projekata, u dva grada, na kojima sudjeluje tisuće studenata. Posljedica povećanja broja projekata, ali i studenata koji sudjeluju na tim projektima, jest rast broja članova u Udruzi. Ovaj rast stvorio je potrebu za boljom organizacijom podataka i učinkovitijom kontrolom rada članova. Podaci su se prikupljali putem nekoliko zastarjelih internih aplikacija koje međusobno nisu bile povezane iako su sadržavale većinom identične informacije. Zbog ograničenja tih aplikacija, u posljednjih nekoliko godina sve više podataka se pohranjivalo na Google Drive platformu u obliku Google Docs i Google Sheets datoteka. Datoteke su postajale prevelike za brzo otvaranje, što je otežavalo rad.

Jedan od najvećih nedostataka bio je nedostatak pristupa određenim podacima unutar tih dokumenata. Udruga ima hijerarhijsku strukturu, koja omogućava pristup podacima na temelju pozicije, što je značilo da su članovi morali tražiti podatke od svojih nadređenih, često sve do najviših razina. Uza sve to, česte promjene vodstva dovodile su do dupliciranja podataka, kao i njihova gubitka jer nisu postojali jasni uvjeti za njihovo upravljanje, a broj članova je bio prevelik za detaljnu kontrolu, što je dodatno pogoršalo situaciju.

## Prijedlog i opis rješenja

Za uspješno rješavanje problema organizacije podataka i praćenja aktivnosti unutar udruge eSTUDENT, potrebno je razviti specifično programsko rješenje koje će u potpunosti pratiti pravilnik Udruge i njenu hijerarhijsku strukturu. Predlažem izgradnju integriranog sustava sastavljenog od šest zasebnih, ali međusobno povezanih modula, koji će omogućiti sveobuhvatno upravljanje svim ključnim aspektima rada Udruge. Ovaj sustav ne samo što će konsolidirati podatke na jednome mjestu već će i omogućiti transparentnost, učinkovitost i jednostavnost korištenja, čime će znatno unaprijediti zadovoljstvo članova Udruge.

Modul eAKTIVNOSTI omogućit će unos svih aktivnosti unutar Udruge te praćenje prisutnosti članova na tim aktivnostima. Ovaj će modul osigurati da svi relevantni podaci o angažmanu članova budu centralizirani i lako dostupni, čime će se eliminirati potreba za fragmentiranim i nepouzdanim evidencijama u različitim tablicama ili aplikacijama.

Modul eINFO bit će ključan za praćenje ispunjavanja uvjeta potrebnih za dobivanje certifikata o uspješnom radu na kraju godine. Svi članovi imat će uvid u vlastiti napredak i jasnu sliku o tome koje uvjete još trebaju ispuniti. Time se smanjuje potreba za konstantnim kontaktiranjem nadređenih i omogućava članovima da proaktivno upravljaju svojim obvezama.

Modul ePARTNERI služit će za praćenje suradnji s partnerima Udruge, pohranjivanje kontaktnih podataka te vođenje evidencije o projektima na kojima su sudjelovali partneri. Ova baza podataka omogućit će Udruzi da lakše upravlja poslovnim odnosima, a voditeljima timova da brzo pristupe potrebnim informacijama.

Modul SUPRACH osmišljen je kao alat za praćenje učinkovitosti članova. Svi članovi morat će ispuniti evaluacijske obrasce dva puta godišnje, ocjenjujući rad svojih suradnika i voditelja. Ovaj će modul omogućiti prikupljanje povratnih informacija, što će pomoći u identifikaciji problema unutar timova i unapređenju rada cijele Udruge.

Modul eIZBORI omogućit će provedbu anonimnih i sigurnih izbora unutar Udruge, bilo na kraju godine ili prilikom održavanja skupština. Automatizirani sustav glasovanja osigurava transparentnost i pravilnost izbora, čime se dodatno jača povjerenje članova u demokratske procese unutar Udruge.

Modul eADMIN bit će alat namijenjen administratorima za dodavanje novih korisnika putem .csv parsera te će omogućiti osnovne CRUD operacije u bazi podataka. Ovaj modul osigurava jednostavno upravljanje korisnicima i podacima, što će dodatno poboljšati operativnu učinkovitost Udruge.

Pristup pojedinim modulima bit će strogo kontroliran, s dozvolama koje će se dodjeljivati prema poziciji u timu unutar Udruge (Tablica 4.2). Voditelji timova i koordinatori imat će mogućnost praćenja stanja svojih članova i podsjećanja na obveze. Za autorizaciju će se koristiti Google OAuth2, čime će se članovima omogućiti jednostavan pristup, bez potrebe za pamćenjem dodatnih korisničkih podataka. Iskorištavanje besplatne Google domene za neprofitne organizacije osigurava da se ovaj proces odvija na siguran i efikasan način.

Konsolidacija svih podataka unutar jednog sustava znatno će smanjiti potrebu za korištenjem velikog broja Excel datoteka, omogućujući da sve informacije budu lako dostupne na jednome mjestu. Ovakav pristup ne samo što povećava operativnu učinkovitost već i zadovoljstvo članova Udruge, koji će sustav doživljavati kao jednostavniji i efikasniji. Ovim rješenjem, eSTUDENT će postaviti temelje za dugoročno uspješno upravljanje svojim članstvom i resursima, što će u konačnici rezultirati boljim ostvarivanjem ciljeva Udruge.

## Usporedba s postojećim rješenjima

Postojeće platforme za upravljanje neprofitnim organizacijama nude širok spektar funkcionalnosti, no nijedna od njih ne pokriva sve specifične potrebe naše Udruge. U nastavku slijedi analiza triju najpopularnijih rješenja: Salesforce Nonprofit Cloud, Blackbaud i Wild Apricot.

### Analiza Salesforce Nonprofit Cloud alata

Salesforce Nonprofit Cloud je iznimno robustan i prilagodljiv sustav dizajniran za potrebe neprofitnih organizacija. Baziran je na poznatoj Salesforce platformi koja se koristi u raznim industrijama za upravljanje odnosima s klijentima. [1]

Funkcionalnosti Salesforce Nonprofit Cloud [1]:

* Upravljanje donacijama: Omogućava praćenje svih donacija, uključujući jedinstvene i ponavljajuće donacije, s detaljnim uvidom u donatorske profile i povijest donacija.
* Upravljanje odnosima s donatorima: Centralizirana baza podataka, koja bilježi interakcije s donatorima, omogućuje personaliziranu komunikaciju i praćenje angažmana.
* Volonteri i događaji: Alati za planiranje i upravljanje događajima, kao i volonterima, uključujući registraciju, praćenje prisutnosti i angažmana.
* Analitika i izvještavanje: Napredni alati za analizu podataka s prilagodljivim izvještajima i vizualizacijama, omogućuju donošenje informiranih odluka.
* Integracije: Širok spektar integracija s drugim alatima i aplikacijama, kao što su sustavi za e-mail marketing, prikupljanje sredstava i društvene mreže.

Prednosti Salesforce Nonprofit Cloud [1]:

* Visoka prilagodljivost: Može se prilagoditi specifičnim potrebama organizacije uz mogućnost izrade prilagođenih aplikacija unutar Salesforce ekosustava.
* Snažna podrška zajednice i resursa: Salesforce ima veliku zajednicu korisnika i opsežnu bazu znanja s uputama, vodičima i podrškom.
* Skalabilnost: Sustav se lako može proširiti kako organizacija raste.

Nedostaci Salesforce Nonprofit Cloud [1]:

* Visoki troškovi: Licenciranje i prilagodbe mogu biti skupi, što je velik izazov za organizacije s ograničenim budžetom.
* Tehnička složenost: Za implementaciju i održavanje često je potrebno visoko tehničko znanje ili angažiranje stručnjaka.
* Dugotrajna obuka: Zbog složenosti sustava, korisnici trebaju proći opsežnu obuku kako bi učinkovito koristili sve funkcionalnosti.

### Analiza Blackbaud alata

Blackbaud je specijalizirana platforma namijenjena isključivo neprofitnim organizacijama, s fokusom na upravljanje donacijama, prikupljanje sredstava i upravljanje odnosima s donatorima. [2]

Funkcionalnosti Blackbauda [2]:

* Upravljanje donacijama i prikupljanje sredstava: Omogućava detaljno praćenje donacija, planiranje kampanja za prikupljanje sredstava te analizu uspjeha tih kampanja.
* CRM za neprofitne organizacije: Poseban CRM sustav dizajniran je za upravljanje odnosima s donatorima, volonterima i članovima, uz mogućnost segmentiranja baza podataka za ciljane kampanje.
* Organizacija događaja: Alati za organizaciju i upravljanje događajima, uključujući registraciju sudionika, prodaju ulaznica i praćenje donacija tijekom događaja.
* Upravljanje članstvom: Omogućuje upravljanje članovima, praćenje njihovih angažmana i obnavljanje članstva.
* Financijsko upravljanje: Integrirani alati za praćenje financijskih tokova, uključujući računovodstvo, plaćanje i izvještavanje.

Prednosti Blackbauda [2]:

* Specijaliziranost za neprofitne organizacije: Fokus na ključne funkcionalnosti potrebne neprofitnim organizacijama, s alatima posebno prilagođenim prikupljanju sredstava.
* Podrška i usluge: Blackbaud nudi opsežnu podršku, uključujući personaliziranu obuku i konzultantske usluge.
* Integracija financija: Direktna integracija s financijskim sustavima omogućuje praćenje donacija i troškova unutar iste platforme.

Nedostaci Blackbauda [2]:

* Visoki troškovi: Kao i Salesforce, Blackbaud dolazi s visokim troškovima licenciranja i implementacije.
* Složenost: Sustav je složen za korištenje i često zahtijeva obuku ili vanjsku podršku za implementaciju i održavanje.
* Manjak fleksibilnosti: Iako je specijaliziran, sustav može biti manje fleksibilan za organizacije koje trebaju prilagodljiva rješenja.

### Analiza Wild Apricot alata

Wild Apricot je pristupačna i jednostavna platforma za upravljanje neprofitnim organizacijama, posebno pogodna za manje udruge. Fokusira se na upravljanje članstvom i događajima, s osnovnim funkcionalnostima za upravljanje donacijama. [3]

Funkcionalnosti Wild Apricota [3]:

* Upravljanje članstvom: Omogućuje jednostavno upravljanje bazom članova, uključujući registraciju, obnavljanje članstva i upravljanje kontaktima.
* Organizacija događaja: Alati za planiranje i upravljanje događajima, registraciju sudionika, naplatu ulaznica i praćenje prisutnosti.
* Online plaćanja: Integracija s alatima za online plaćanja omogućuje primanje donacija i naplatu članarina putem interneta.
* Jednostavan CRM: Osnovni CRM alat za praćenje interakcija s članovima i donatorima.
* Web-stranica i komunikacija: Omogućuje izradu jednostavne web-stranice i upravljanje e-mail kampanjama unutar iste platforme.

Prednosti Wild Apricota [3]:

* Pristupačnost: Relativno niski troškovi u usporedbi s drugim platformama čine ga pogodnim za manje organizacije.
* Jednostavnost korištenja: Intuitivno sučelje i jednostavan postupak implementacije omogućuju brzo uvođenje sustava, bez potrebe za opsežnom obukom.
* Sveobuhvatno rješenje za osnovne potrebe: Pokriva osnovne potrebe upravljanja članstvom i događajima, uključujući online plaćanja.

Nedostatci Wild Apricota [3]:

* Ograničene funkcionalnosti: Nedostatak naprednih alata za upravljanje donacijama, analitiku i prilagodbu sustava.
* Nedostatak skalabilnosti: Ograničene mogućnosti za prilagodbu i proširenje funkcionalnosti mogu postati problem kako organizacija raste.
* Nedostatak napredne podrške: Iako jednostavan za korištenje, podrška je manje sveobuhvatna nego kod skupljih platformi.

### Zaključak analize

Detaljna analiza postojećih platformi za upravljanje neprofitnim organizacijama, uključujući Salesforce Nonprofit Cloud, Blackbaud i Wild Apricot, pokazuje da nijedna od njih ne može u potpunosti zadovoljiti specifične potrebe Udruge.

Salesforce Nonprofit Cloud, iako iznimno moćan i prilagodljiv sustav, nameće izazove vezane uz visoke troškove i složenost implementacije. Sustav zahtijeva značajne tehničke resurse i znanje, što može biti prepreka za organizacije s ograničenim proračunom i resursima. S druge strane, Blackbaud, iako je specijaliziran za neprofitni sektor i pruža napredne alate za prikupljanje sredstava i upravljanje donatorima, također dolazi s visokim financijskim opterećenjem i složenom uporabom. To dodatno otežava implementaciju u manjim organizacijama koje nemaju resurse za opsežnu obuku i prilagodbu sustava. Wild Apricot nudi jednostavnije i pristupačnije rješenje, ali njegove su funkcionalnosti ograničene. Sustav je prilagođen manjim udrugama s jednostavnijim potrebama, dok kompleksne organizacije, koje zahtijevaju napredne alate za upravljanje donacijama, analitiku i prilagodbu, neće pronaći odgovarajuća rješenja u ovom alatu.

S obzirom na sve ove čimbenike, postaje jasno da nijedna od analiziranih platformi ne može u potpunosti zadovoljiti specifične zahtjeve udruge. Nedostatak sveobuhvatnog rješenja, koje bi pokrilo sve aspekte rada unutar Udruge, od praćenja članstva do upravljanja donacijama i organizacije događaja, predstavlja značajan problem. Dodatno, uporaba više različitih platformi mogla bi dovesti do fragmentacije podataka, poteškoća u koordinaciji i povećanja ukupnih troškova.

Uzimajući u obzir navedene slabosti postojećih rješenja, jedini logičan izlaz je razvoj prilagođenog sustava koji će integrirati sve potrebne funkcionalnosti u jednu platformu. Takvo rješenje omogućit će optimalno prilagođavanje specifičnim zahtjevima Udruge, pojednostaviti organizacijske procese, smanjiti operativne troškove i omogućiti članovima jednostavniji i učinkovitiji rad. Razvoj ovoga jedinstvenog sustava omogućit će organizaciji da zadrži fleksibilnost, poboljša transparentnost i optimizira resurse, što će u konačnici povećati uspješnost i zadovoljstvo svih uključenih strana.

## Usklađivanje ideje s pravilnicima Udruge

Tri od šest modula predloženog sustava trebaju biti pažljivo razvijeni kako bi usko pratili pravilnike Udruge. Modul **eAKTIVNOSTI** i modul **eINFO** bit će usklađeni s „Pravilnikom o dodjeli certifikata“. Ovi će moduli omogućiti praćenje aktivnosti koje utječu na ispunjavanje uvjeta za certifikate i prikazivati trenutno stanje uvjeta, koji se razlikuju ovisno o poziciji unutar Udruge.

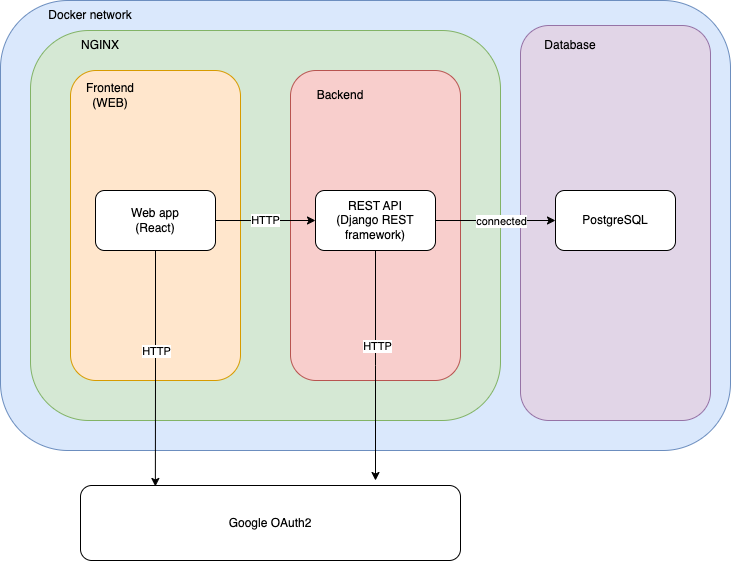
Modul **eIZBORI** bit će razvijen u skladu s pravilnicima o izborima, uključujući „Pravilnik o izboru i radu Nadzornog odbora“ i „Pravilnik o izboru i radu voditelja i Predsjedništva studentske Udruge eSTUDENT“. U ovom modulu, bit će jasno definirano tko ima pravo glasa za koje pozicije; na primjer, za izbor voditelja tima glasaju samo članovi tog tima, dok za Predsjedništvo Udruge pravo glasa imaju samo članovi prisutni na Skupštini.

Svi će moduli poštovati statut Udruge, osiguravajući da su svi aspekti upravljanja usklađeni s pravilima i procedurama Udruge.

# Arhitektura programskog multifunkcionalnog sustava

U ovom će poglavlju biti predstavljena arhitektura programskog multifunkcionalnog sustava, koja detaljno opisuje organizaciju i međusobnu povezanost različitih komponenti unutar sustava. Prikazat će se ključne komponente sustava te način na koji komuniciraju unutar mrežnog okruženja. Poseban naglasak bit će stavljen na korištenje tehnologija, kao i na integraciju s vanjskim servisima, poput onoga za autentifikaciju korisnika.

## Skica i opis arhitekture sustava



Slika 3.1 Arhitektura sustava

Prikazana arhitektura (Slika 3.1) sustava sastoji se od tri glavne komponente: korisničkog sučelja (frontend), poslužiteljskog sloja (backend) i baze podataka, koje sve djeluju unutar Docker mrežnog okruženja. NGINX se koristi kao reverzni proxy poslužitelj, usmjeravajući dolazne HTTP zahtjeve prema odgovarajućim komponentama unutar Docker mreže.

Korisničko sučelje sustava izgrađeno je korištenjem Reacta, popularne JavaScript biblioteke za izradu korisničkih sučelja. Ova je komponenta odgovorna za prikaz sučelja i interakciju s korisnicima, a putem HTTP zahtjeva komunicira s poslužiteljskim slojem. Poslužiteljski sloj je razvijen pomoću Django web-okvira, uz korištenje Django REST Frameworka za izradu RESTful API-ja. Ovaj API povezuje korisničko sučelje s bazom podataka, obrađuje zahtjeve, upravlja poslovnom logikom, autentifikacijom i autorizacijom te omogućava izvođenje CRUD operacija (kreiranje, čitanje, ažuriranje i brisanje) nad podacima.

Baza podataka je upravljana putem PostgreSQL-a, moćnog sustava za upravljanje relacijskim bazama podataka, koji pohranjuje i dohvaća podatke prema zahtjevima poslanim putem API-ja. Sustav je također integriran s Google OAuth2 za autentifikaciju korisnika, omogućujući prijavu putem Google vjerodajnica. Cijeli je sustav modularan, skalabilan i siguran zahvaljujući korištenju Docker kontejnerizacije, NGINX-a za efikasno usmjeravanje zahtjeva te robusnom poslužiteljskom sučelju izgrađenom pomoću Django i PostgreSQL tehnologija.

## Korištene tehnologije

U sljedećem poglavlju opisuju se korištene tehnologije i paradigme pri izradi RESTful API servisa te paradigme pri pripremanju aplikacije za produkciju.

### PostgreSQL

PostgreSQL je relacijska baza otvorenog kôda koja je odgovorna za pohranu podataka REST API servisa. Radi se o besplatnom sustavu za upravljanje relacijskim bazama podataka koji nudi širok raspon značajki, uključujući podršku za napredne SQL funkcionalnosti, skalabilnost i fleksibilnost u prilagodbi potrebama korisnika. Trenutna verzija PostgreSQL-a, koja se koristi u ovom programskom rješenju, jest 16.3.

Prednosti PostgreSQL-a u odnosu na druge baze podataka uključuju njegovu otvorenost, besplatnu licencu, podršku za napredne funkcije, poput rekurzivnih upita, common table expressions, window funkcija i potpore za pohranu različitih tipova podataka (npr. JSON, XML). Također, PostgreSQL je poznat po svojoj robusnosti, pouzdanosti i aktivnoj zajednici koja kontinuirano doprinosi njegovom razvoju.

Mane PostgreSQL-a mogu uključivati nešto veću kompleksnost pri konfiguraciji i održavanju u odnosu na jednostavnije RDBMS-ove, kao što su MySQL ili SQLite. Iako je PostgreSQL vrlo moćan, to može značiti da je potrebna dodatna stručnost za njegovo optimalno korištenje, što može predstavljati izazov za manje iskusne timove.

Pristupačnost PostgreSQL-a je velika s obzirom na to da je dostupan na većini platformi (Linux, Windows, macOS) i podržava razne razvojne okvire i jezike. Njegova jednostavnost korištenja može varirati, ovisno o iskustvu korisnika, ali je generalno priznata kao dobro dokumentirana baza podataka s bogatom dokumentacijom i brojnim resursima za učenje. [4]

Kao open-source rješenje, PostgreSQL je besplatan za korištenje, što ga čini atraktivnim izborom za mnoge organizacije, posebno one koje traže snažno i fleksibilno rješenje, bez visokih troškova licenciranja. Što se tiče podrške za REST API, PostgreSQL se može lako integrirati s raznim alatima i frameworkovima za razvoj API-ja, omogućujući učinkovito dohvaćanje i manipulaciju podacima. [4]

PostgreSQL se trudi uskladiti sa SQL standardom gdje god je to moguće, ali ta usklađenost nije uvijek potpuna zbog zadržavanja tradicionalnih značajki i izbjegavanja loših arhitektonskih odluka. Iako podržava mnoge značajke koje zahtijeva SQL standard, ponekad ih implementira s drugačijom sintaksom ili funkcijom kako bi optimizirao performanse ili pružio dodatne mogućnosti koje standard ne pokriva. SQL standardi su skup pravila koja definiraju kako bi SQL jezik trebao funkcionirati, omogućujući interoperabilnost, dosljednost i portabilnost aplikacija među različitim sustavima baza podataka. Međutim, PostgreSQL, kao i mnogi drugi RDBMS sustavi, ne slijedi ih u potpunosti zbog povijesnih odluka, potrebe za boljim performansama ili pružanja proširenja koja nisu dio standarda. Ovo odstupanje od standarda nije jedinstveno za PostgreSQL, već je uobičajeno među različitim bazama podataka koje balansiraju između usklađenosti sa standardima i specifičnih potreba korisnika. [4]

U sklopu sustava, PostgreSQL se koristi kroz Django REST framework te nema posebnih konfiguracija. Aplikacija se jednostavno povezuje s bazom podataka, a sve operacije nad podacima odvijaju se putem Django REST frameworka. Detaljniji opis nalazi se u poglavlju implementacije sustava. [5]

### Django REST framework

Django REST Framework je moćan i fleksibilan alat za izradu web API-ja unutar Django aplikacija, a koji koristi Python kao programski jezik. Temeljen je na Django web frameworku, koji je također razvijen u Pythonu, što omogućuje lako integriranje i korištenje unutar Django projekata. Django REST Framework koristi tehnologiju temeljenu na modelima (engl. *model-driven*), gdje se podaci definiraju kao Django modeli, koji se potom povezuju s bazom podataka.

Okruženje, koje podržava Django REST Framework, obuhvaća širok spektar alata i tehnologija, uključujući ORM sustav (engl. *Object-Relational Mapping*), koji omogućuje jednostavno mapiranje podataka između Python objekata i tablica u bazi podataka. Django REST Framework pruža podršku za razne formate podataka, kao što su JSON, XML i YAML, što ga čini fleksibilnim za različite vrste aplikacija. Nadalje, Django REST Framework nudi bogate mogućnosti autentifikacije, autorizacije, upravljanja sesijama i paginacije, čime olakšava implementaciju sigurnih i skalabilnih REST API servisa.

Kao dio Python ekosustava, Django REST Framework koristi prednosti velikog broja dostupnih knjižnica i modula, što dodatno proširuje njegove mogućnosti. Razvojno okruženje uključuje upotrebu virtualnih okruženja za upravljanje ovisnostima i alatima za migraciju baza podataka, testiranje i implementaciju aplikacija. Django REST Framework je stoga idealan izbor za razvoj složenih i robusnih web-aplikacija koje trebaju pouzdanu integraciju s relacijskim bazama podataka poput PostgreSQL-a. [5]

U sklopu sustava konfigurirane su dodatne komponente za omogućavanje pravilnog rukovanja CORS (engl. *Cross-Origin Resource Sharing*) zahtjevima, čime se omogućuje korisničkom sučelju aplikacije, koje se nalazi na drugoj domeni, komuniciranje s API-jem. Autorizacija se vrši putem prilagođenog API-ja koji provjerava token, a nakon provjere koristi se Simple JWT za rukovanje JSON Web Token autentifikacijom. JWT autentifikacija predstavlja metodu sigurnog prijenosa informacija između klijenta i servera u obliku JSON objekata, koji su digitalno potpisani, kako bi se osigurala autentičnost i integritet podataka. JSON Web-tokeni sadrže tri dijela: zaglavlje (engl. *header*), korisni teret (engl. *payload*) i potpis (engl. *signature*). Zaglavlje obično specificira tip tokena i algoritam enkripcije, korisni teret uključuje informacije kao što su korisnički ID i dozvole, dok potpis osigurava da se token nije promijenio tijekom prijenosa. JSON Web Zoken je popularan zbog svoje jednostavnosti i skalabilnosti, omogućujući autentifikaciju bez potrebe za čuvanjem sesijskih podataka na serveru, što olakšava rad s distribuiranim sustavima. Postoje još dvije metode autentifikacije koje koristi aplikacija. OAuth2, koji omogućuje pristup resursima na serveru uz korištenje pristupnih tokena, i koji se često koristi za pružanje pristupa resursima trećih strana (npr. prijava putem Google ili Facebook računa). Također, koristi se za autentifikaciju korisnika prije generiranja JSON Web-tokena. Session-based autentifikacija je autentifikacija koja se temelji na sesijama što se pohranjuju na serveru, a korisnici se identificiraju putem kolačića koji sadrže sesijski ID. Iako je session-based autentifikacija pouzdana i jednostavna za implementaciju, manje je skalabilna od JWT autentifikacije u distribuiranim okruženjima. Session-based autentifikacija koristi se prilikom pristupa Django admin stranici [5]

Osim toga, unutar sustava je konfigurirana i Django admin stranica, koja omogućuje administraciju podataka direktno kroz preglednik. Ova je funkcionalnost iznimno korisna za administratore sustava jer olakšava upravljanje podacima pohranjenima u bazi podataka, omogućujući im pregled, uređivanje i brisanje zapisa, bez potrebe za izravnim pristupom bazi podataka.

### React

React je popularna JavaScript biblioteka za izradu korisničkih sučelja, razvijena od strane Facebooka, koja omogućuje izradu komponenti koje se mogu ponovno koristiti i koje olakšavaju izradu složenih korisničkih sučelja. Jedna od ključnih prednosti Reacta je njegov Virtual DOM (engl*. Virtual Document Object Model*), koji omogućuje brže i efikasnije ažuriranje sučelja. Kada dođe do promjene stanja ili podataka, React prvo ažurira virtualni prikaz DOM-a, a zatim učinkovito primjenjuje samo nužne promjene na stvarnom DOM-u, čime se minimizira broj operacija izravno na stvarnom DOM-u i poboljšavaju se performanse aplikacije.

React također podržava deklarativni stil programiranja, gdje programeri definiraju kako bi korisničko sučelje trebalo izgledati u određenom stanju, a React se brine o tome kako će se te promjene provesti. Ovaj pristup olakšava razumijevanje i održavanje kôda, posebno u velikim i složenim aplikacijama.

U ovom projektu, React je korišten za razvoj korisničkog sučelja koje omogućuje intuitivnu i responzivnu interakciju korisnika s aplikacijom. Projekt je inicijalno postavljen korištenjem Vite, modernoga razvojnog alata koji se ističe svojom brzinom i efikasnošću. Vite koristi ESM za brzu obradu i prepakiranje modula, što drastično smanjuje vrijeme kompajliranja i omogućuje gotovo trenutno osvježavanje stranica tijekom razvoja. Ova je značajka posebno korisna za iterativni razvoj jer omogućuje programerima da brzo vide rezultate svojih promjena, što ubrzava cijeli razvojni proces.

TypeScript je korišten kao programski jezik za pisanje aplikacije, što je nadogradnja na JavaScript koja dodaje statičko tipiziranje. Statičko tipiziranje omogućuje otkrivanje pogrešaka već tijekom faze razvoja, umjesto za vrijeme izvršavanja, što značajno poboljšava pouzdanost i stabilnost aplikacije. TypeScript također olakšava rad na većim projektima, pružajući mogućnost definiranja složenih tipova podataka, što rezultira čišćim i održivijim kôdom.

Projekt koristi niz osnovnih biblioteka koje dodatno poboljšavaju funkcionalnost aplikacije. Ant Design (AntD) je popularan React UI framework koji pruža bogat skup gotovih komponenti, kao što su forme, tablice, modalni prozori i razni elementi korisničkog sučelja, što omogućuje bržu izradu dosljednih i vizualno privlačnih sučelja.

Uz to, uključene su i druge važne biblioteke:

* Axios za obavljanje HTTP zahtjeva, što omogućuje jednostavnu komunikaciju između korisničkog sučelja i poslužiteljskog sloja.
* React Router DOM za upravljanje rutama unutar aplikacije, koji omogućuje navigaciju između različitih dijelova aplikacije na jednostavan i modularan način.
* React Query za upravljanje stanjem i dohvaćanje podataka optimizira performanse aplikacije, smanjujući pritom nepotrebne ponovne zahtjeve i pružajući jednostavne alate za rad s asinkronim podacima.
* js-cookie za upravljanje kolačićima omogućuje jednostavno pohranjivanje i dohvaćanje podataka u kolačićima.
* Day.js za rad s datumima i vremenom nudi laganu i fleksibilnu alternativu popularnim bibliotekama poput Moment.js.
* @react-oauth/google za integraciju s Google OAuth2 protokolom omogućuje korisnicima prijavu putem Google računa, čineći proces autentifikacije sigurnijim i jednostavnijim.

Kombinacija ovih alata i tehnologija omogućuje izgradnju robusne, sigurne i visokoučinkovite aplikacije koja pruža bogato korisničko iskustvo i pouzdano upravlja podacima pohranjenim u bazi podataka.

### Docker

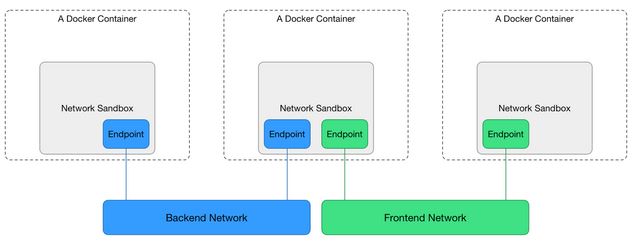
Docker je moćna platforma za izgradnju, distribuciju i pokretanje aplikacija unutar kontejnera, koja omogućava jednostavno upravljanje i skaliranje aplikacija u različitim okruženjima. Kontejneri su lagane, prenosive jedinice koje uključuju sve potrebne komponente aplikacije, kao što su kôd, biblioteke i konfiguracije, osiguravajući dosljednost u radu aplikacije, neovisno o okolini u kojoj se pokreće, bilo da se radi o lokalnom razvojnom okruženju, testiranju ili produkciji. Docker koristi koncept **kontejnerizacije** koji omogućuje izolaciju aplikacija i njihovih ovisnosti u zasebne kontejnerske instance, čime se eliminiraju problemi s kompatibilnošću i konfliktima između različitih verzija softvera.

Docker koristi **Docker Engine** koji se sastoji od:

* **Docker Daemona, koji je** glavni servis koji upravlja Docker objektima, kao što su slike, kontejneri, mreže i volumeni
* **REST API-ja** putem kojeg Docker Daemon komunicira s klijentima
* **Docker CLI-j**a koji omogućuje korisnicima interakciju s Dockerom.

Ključni aspekt Dockera je njegova sposobnost da koristi **Docker slike (engl. *image*)** kao predloške za kreiranje kontejnera. Docker slike su nepromjenjive i sadrže sve što je potrebno za pokretanje aplikacije, a kontejneri su pokrenute instance tih slika.

Docker mreže (engl. *network*) omogućuju međusobnu komunikaciju kontejnera unutar iste mreže, gdje svaki kontejner može komunicirati s drugim koristeći DNS nazive umjesto IP adresa. To olakšava konfiguraciju i skaliranje jer se kontejneri mogu dinamički dodavati ili uklanjati iz mreže, bez utjecaja na druge komponente aplikacije. U sustavima, gdje je potreban load balancing, Docker mreže u kombinaciji s alatima, kao što je **NGINX,** omogućuju upravljanje prometom između korisnika i aplikacije, osiguravajući visoku dostupnost i učinkovitost. Docker također podržava **volume** za trajnu pohranu podataka, što omogućuje da podaci prežive ponovno pokretanje ili ažuriranje kontejnera, čime se osigurava kontinuitet rada aplikacija. [6]



Slika 3.2 [7] Vizualni prikaz docker mreže

Sve ove komponente zajedno omogućuju Dockeru da pojednostavi cijeli životni ciklus razvoja i distribucije aplikacija, omogućujući brže izdavanje verzija, poboljšanu sigurnost, bolju upotrebu resursa i jednostavniju administraciju složenih aplikacija u različitim okruženjima. [6]

Docker je u ovom sustavu posložen na način da glavna *docker-compose.yaml* datoteka orkestrira pokretanje svih kontejnera unutar zajedničke mreže. Docker mreža se automatski kreira i upravlja kontejnerima, a *docker-compose.yaml* pruža mogućnost definiranja različitih mreža prema potrebama aplikacije, poput *bridge* mreže koja izolira kontejnere ili *overlay* mreže, a koja povezuje više Docker hostova za skaliranje aplikacija na više servera.

Nakon definiranja mreže, Docker pokreće NGINX kontejner, koji služi kao reverzni proxy, usmjeravajući promet između korisničkog sučelja i poslužiteljskog sloja. Ovaj pristup omogućuje sigurno i učinkovito upravljanje prometom, pružajući izolaciju i kontrolu nad komunikacijom unutar sustava. Time se osigurava da aplikacija bude dostupna i pouzdana, bez obzira na kompleksnost infrastrukture ili broj kontejnera koji se koriste.

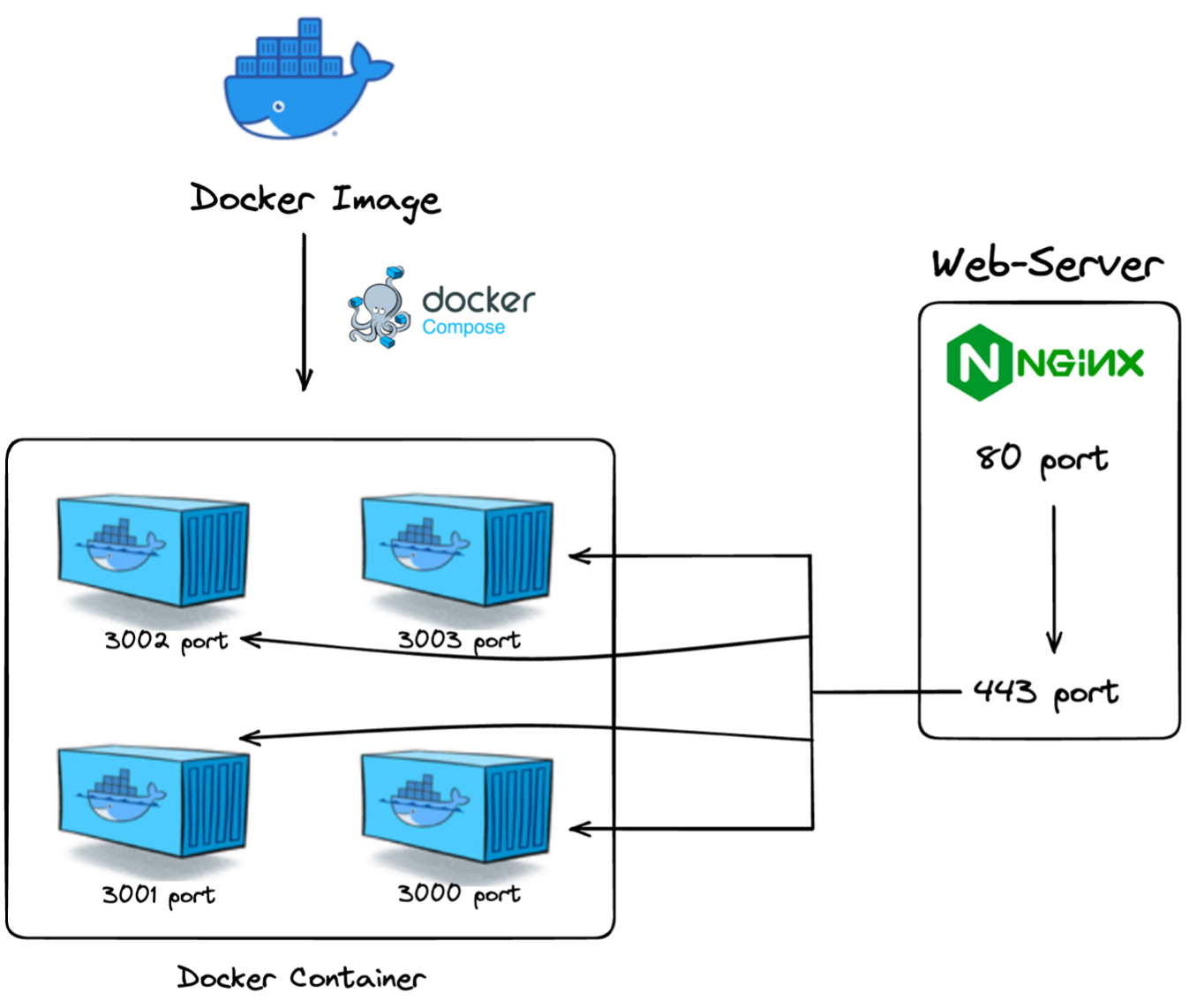
### NGINX

NGINX je snažan web-server i reverzni proxy server otvorenog kôda koji se također može koristiti kao load balancer, HTTP cache i mail proxy. Razvio ga je Igor Sysoev 2002. godine, a zamišljen je tako da ostvari visoku performansu i stabilnost. [7]

Radi na način da za svaki zahtjev koji prima ne stvara novi proces ili thread, već koristi unaprijed definiran broj radnih procesa koji komuniciraju kroz jednostavan sistem upravljanja događajima. To mu omogućava brzu obradu zahtjeva i vrlo dobru skalabilnost na sistemu s velikim opterećenjem.

Među prednostima NGINX-a su visoka efikasnost u radu sa statičkim i dinamičkim sadržajem, smanjenje potrebe za hardverskim resursima, kao i sposobnost upravljanja i optimizacije web-prometa kroz napredne funkcije, kao što su SSL soffloading, caching, kompresija sadržaja i zaštita od DDOS napada. [8]

NGINX je postavljen kao proxy server kojem je glavni cilj da prema dobivenom URIu preusmjeri poziv na specifični docker kontejner. Tako će sve zahtjeve koji počinju s „/api“ preusmjeriti na kontejner u kojem je pokrenut API, a ostatak na korisničko sučelje.



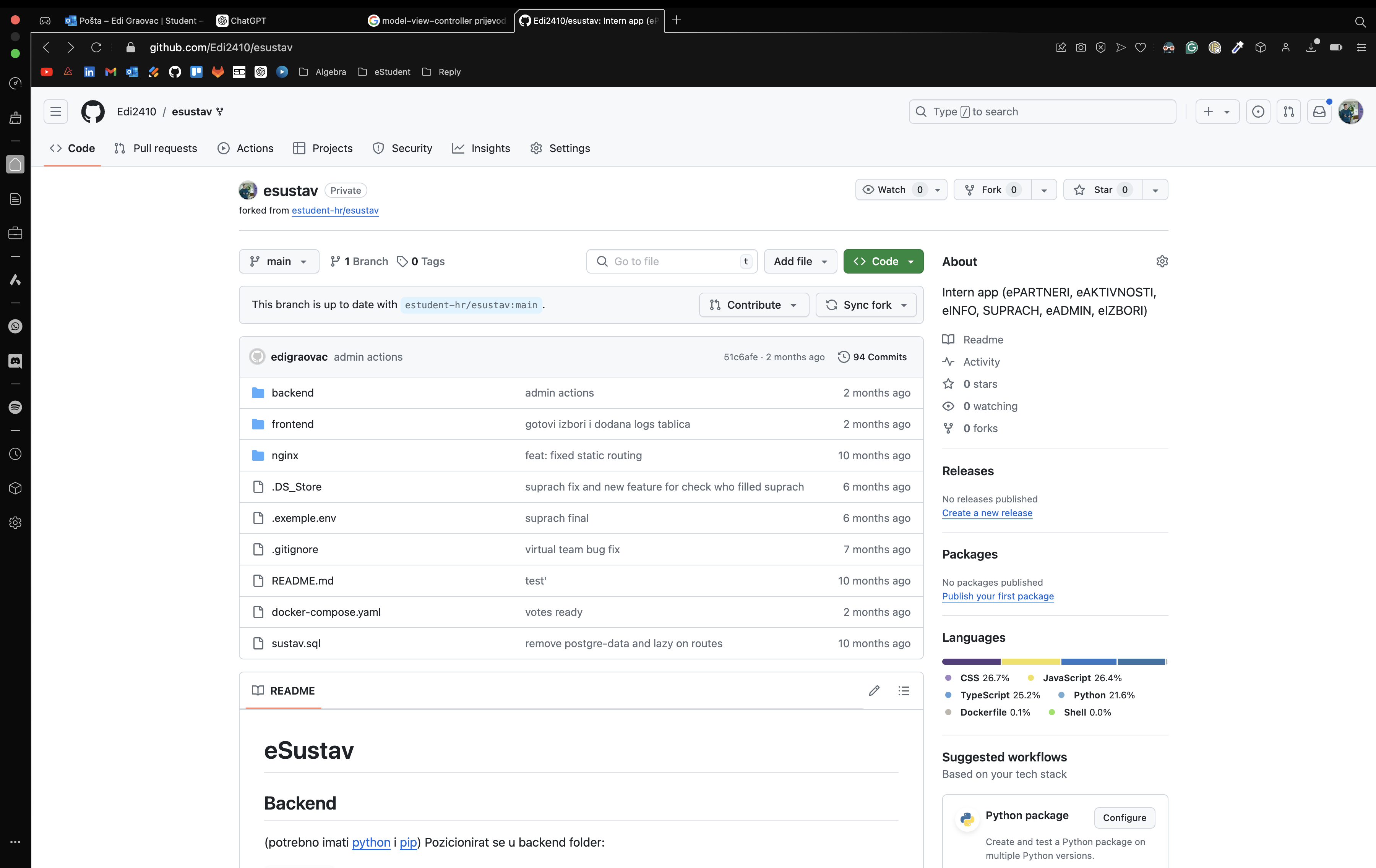
Slika 3.3 [10] Slikoviti prikaz NGINX funkcije

## Verzioniranje kôda

Izvorni kôd upravlja se korištenjem Git alata, distribuiranog sustava za čuvanje i kontrolu verzija kôda. Git omogućava praćenje promjena u bilo kojem skupu datoteka te olakšava koordinaciju među programerima koji zajednički rade na razvoju softvera. Neke od glavnih prednosti Gita uključuju njegovu brzinu, integritet podataka i podršku za distribuirane, nelinearne tijekove rada, što omogućava istovremeni rad na tisućama paralelnih grana na različitim sustavima. [9]

Unutar eSTUDENT GitHub organizacije kreiran je privatni repozitorij pod nazivom *esustav*. Sve promjene kôda kontinuirano su se spremale u ovaj repozitorij. Trenutno, repozitorij sadrži najnoviju verziju kôda na grani pod nazivom *main*. Struktura repozitorija sastoji se od dvije glavne grane: *main* i *development*.

Grana *development* služi kao razvojna grana na kojoj se pohranjuju sve izmjene koje su još u fazi testiranja i provjere. Kada se kôd na ovoj grani testira i verificira te kada se procijeni da više nema potrebe za dodatnim izmjenama, ona se sjedinjuje s glavnom (*main*) granom, gdje se pohranjuje stabilna i provjerena verzija aplikacije. Nakon sjedinjavanja, razvojna grana se briše. Prilikom svake nove izmjene kôda, proces započinje iznova: iz *main* grane se kreira nova razvojna grana na kojoj se objavljuje novi kôd.



Slika 3.4 Verzioniranje kôda

# Implementacija multifunkcionalnog sustava

Ovo poglavlje opisuje implementaciju programskog rješenja, komponente od kojih se sastoji sustav, arhitekturu programskog rješenja i realizaciju programskog rješenja uz pomoć programskih jezika, okvira, obrazaca i paradigmi.

## Izrada i konfiguracija baze podataka

Baza podataka zaslužna je za organiziranu pohranu podataka kako bi ostale komponente mogle komunicirati s bazom podataka i dohvaćati, spremati, mijenjati i brisati podatke koji su pohranjeni u bazi podataka.

Sva izrada baze podataka i konfiguracija baze ide kroz Django ORM (engl. *Object-Relational Mapping*) modul. ORM je tehnika koja omogućuje rad s bazama podataka pomoću objektno orijentiranog pristupa, umjesto izravnog pisanja SQL upita. Django ORM je modul koji nam omogućuje interakciju s bazom podataka, koristeći objektno orijentirane API-je visoke razine. ORM preslikava tablice baze podataka u Python klase i zapise baze podataka u instance tih klasa, dopuštajući nam da radimo sa zapisima baze podataka kao da su obični Python objekti. Na taj način, ORM pojednostavljuje rad s bazom podataka jer, umjesto da pišemo kompleksne SQL upite, možemo koristiti Python metode i atribute za manipulaciju podacima.. [5]

Korištenje Django ORM-a omogućuje uvoz već gotovih pristupnih točaka iz vanjskih biblioteka, kao i korištenje ugrađenih tablica koje Django pruža prilikom instalacije. Ovaj je pristup iskorišten za konfiguraciju autorizacije pomoću Django JWT tokena, gdje se koristi predefinirani Django korisnik za autentifikaciju. Kako bi se izbjeglo nadjačavanje originalne korisničke tablice, primijenjena je vertikalna particionacija. To znači da je postojeća tablica korisnika proširena dodavanjem nove tablice UserDetails, čime su omogućene dodatne informacije o korisniku, bez potrebe za izmjenama postojeće strukture.

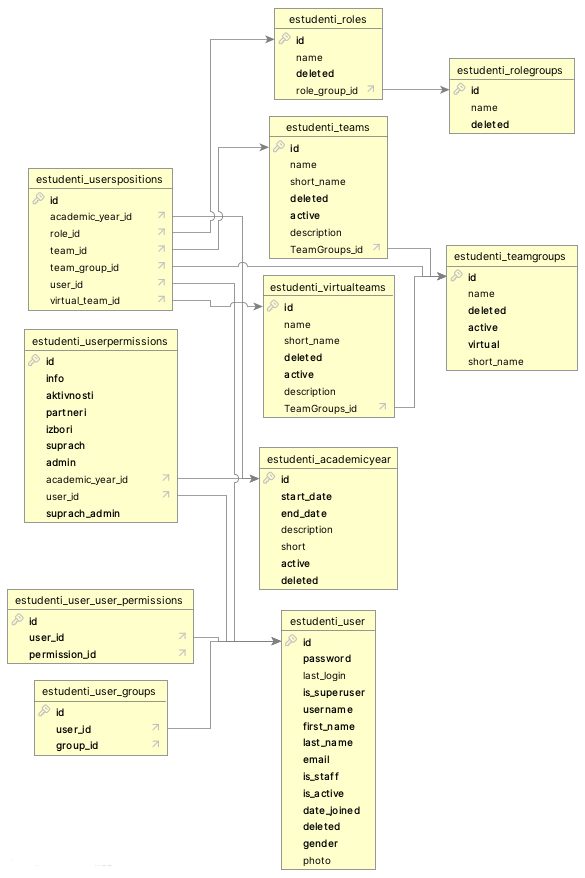
Django ORM koristi specifičan način imenovanja tablica. Prvi dio naziva označava aplikaciju unutar poslužiteljskog sloja u kojoj se nalazi model klase tablice, nakon čega slijedi naziv same klase, odnosno tablice. Ovakav način imenovanja omogućava odvajanje tablica po modulima aplikacije, što doprinosi boljoj preglednosti baze podataka i poslužiteljskog sloja aplikacije. Boljoj preglednosti pridonosi i slijeđenje normalnih formi.

Normalne forme predstavljaju skup ograničenja koje relacijska shema mora zadovoljavati. Najčešće spominjane normalne forme su: prva (1NF), druga (2NF), treća (3NF), Boyce-Coddova (BCNF), četvrta (4NF), peta (5NF) i šesta normalna forma (6NF). U praksi je često dovoljna normalizacija do treće normalne forme.

Da bi tablica bila u prvoj normalnoj formi (1NF), mora imati primarni ključ, jedinstvene vrijednosti i ne smije sadržavati ponavljajuće grupe stupaca. Druga normalna forma (2NF) zahtijeva ispunjavanje uvjeta 1NF i da svaki ne-ključni stupac ovisi o cijelom primarnom ključu. Treća normalna forma (3NF) dodatno zahtijeva da svaki ne-ključni stupac ovisi samo o ključu, a ne o drugim ne-ključnim stupcima.

Za potrebe ovog rada, napravljena je baza podataka prema ER dijagramu koja se nalazi u trećoj normalnoj formi (3NF). Baza podataka je podijeljena u sedam modula.

Glavni modul, estudenti, obuhvaća sve podatke o korisnicima, njihovim timovima, pozicijama i dopuštenjima. Središnja tablica ovog modula je users, koja sadrži podatke za autorizaciju korisnika. Podaci o pozicijama i njihovim grupama pohranjeni su u tablicama roles i rolesgroups, dok se podaci o timovima i njihovim grupama nalaze u tablicama teams i teamgroups. Pojedini timovi mogu biti podijeljeni u podtimove, čiji su podaci pohranjeni u tablici virtualteams. Tablica academicyear sadrži informacije o aktivnoj akademskoj godini te omogućuje njihovo arhiviranje. U tablici userpositions korisnik je povezan sa svojom pozicijom, timom i pod-timom za određenu akademsku godinu.



Slika 4.1 ER dijagram modula *estudneti*

Modul *eaktivnosti* sadrži tablice koje prate sve podatke o aktivnostima i korisnicima koji su sudjelovali u određenim aktivnostima. Tablica *activitytyperequirements* pohranjuje informacije o potrebnom prisustvu korisnika na određenim aktivnostima, dok su tipovi aktivnosti definirani u tablici *activitytypes*. Informacije o provedenim aktivnostima nalaze se u tablici activity*,* a podaci o prisutnosti korisnika na tim aktivnostima u tablici *useractivity*. Certifikat, koji korisnik dobiva na kraju godine, personaliziran je i sadrži pohvalu nadređenog, pohranjenu u tablici *teamleadrecomendations*.

A diagram of a computer

Description automatically generated

Slika 4.2 ER dijagram modula *eaktivnosti*

Modul eizbori sadrži podatke za provedbu tri tipa glasanja: izbora za voditelje, izbora za Predsjedništvo i glasanja za promjene pravilnika. Tablica elections pohranjuje podatke o aktivnim izborima i njihovim tipovima. Kandidati za pojedine izbore evidentirani su u tablici candidates, dok se glasovi korisnika bilježe u tablici votes. Pritom se podatak o tome je li korisnik glasovao bilježi u tablici isvoted, što omogućuje anonimnost glasovanja. Za glasovanje o izmjenama pravilnika, koriste se prijedlozi iz tablice votingdocuments, a glasovi se bilježe u tablici documentsvotes. Prilikom izbora za voditelje, neki korisnici imaju veći broj glasova, a ti su podaci evidentirani u tablici numberofvotesperteam.

A diagram of a computer

Description automatically generated with medium confidence

Slika 4.3 ER dijagram modula *eizbori*

Unutar modula suprach nalaze se tablice potrebne za međusobnu evaluaciju korisnika, sustav poznat kao SUPRACH. Pitanja za evaluaciju nalaze se u tablici question, dok su uvjeti za dodjelu pitanja korisnicima definirani u tablici questionrolerolegroups. Modul omogućuje ocjenjivanje specijalnih osoba ili grupa poput timova, a ti se podaci pohranjuju u tablici specialpersonforgrade. Ocjene i komentari pohranjuju se u tablicama scores i comments, dok se podaci o tome je li korisnik ocijenio određenu osobu nalaze u tablici gradings. Ako se pojedine osobe međusobno ne poznaju, mogu se samo "lajkati", a ti se podaci pohranjuju u tablici likes. Aktivni SUPRACH definiran je u tablici suprach, a pitanja dodijeljena tom SUPRACH-u pohranjena su u tablici suprachquestions.

A screenshot of a computer

Description automatically generated

Slika 4.4 ER dijagram modula *suprach*

Modul epartneri jedini je modul koji nije povezan s korisnicima, a sadrži sve podatke o partnerima s kojima je Udruga surađivala te njihove kontakt osobe, pohranjene u tablicama partners i partnerscontacts. Podaci o projektima na kojima su partneri sudjelovali te svi ostali relevantni podaci nalaze se u tablicama partnernotes i projects.

A screenshot of a computer

Description automatically generated

Slika 4.5 ER dijagram modula *epartneri*

Posljednji i najvažniji modul za otkrivanje pogrešaka je logs, koji sadrži podatke o svakoj pogrešci koja se dogodila na poslužiteljskom sloju te korisniku koji ju je prouzročio, a te se informacije nalaze u tablici errorlogs.

## Implementacija Google OAuth2 autentifikacije

Okvir za autorizaciju OAuth 2.0 omogućuje aplikaciji treće strane dobivanje ograničenog pristupa HTTP usluzi, bilo u ime vlasnika resursa orkestriranjem interakcije odobrenja između vlasnika resursa i HTTP usluge ili dopuštanjem aplikaciji treće strane dobivanje pristupa u svoje ime. [10]

Važno je znati da OAuth 2.0 omogućuje aplikacijama ograničen pristup zaštićenim resursima korisnika (poput bankovnog računa ili drugih osjetljivih informacija), bez potrebe da korisnik otkriva svoje vjerodajnice za prijavu u aplikaciju.[11]

Evo kratkog i jednostavnog pregleda kako funkcionira Google OAuth2 unutar aplikacije.

**Korisnik inicira autentifikaciju** na način da stisne gumb „Login with Google“, aplikacija ga preusmjerava na Google OAuth2 poslužitelj s URL-om za autorizaciju. Taj URL uključuje ključne parametre, kao što su klijent ID, traženi opseg (engl. *scope*) i URL za povratak (engl. redirect URI). Google prikazuje stranicu za prijavu, gdje se korisnik prijavljuje svojim Google računom (ako već nije prijavljen). Nakon prijave, Google traži od korisnika dozvolu za dijeljenje podataka koje je aplikacija zatražila (npr. email, profil). Ako korisnik odobri zahtjev, Google preusmjerava korisnika natrag na URL za povratak (engl. *redirect URI*) te šalje autorizacijski kôd kao parametar u URL-u. Korisničko sučelje dobiva kôd koji šalje prema poslužiteljskom sloju **koji ga potom** šalje Googleu kako bi ga dekodirao i pronašao podatke o korisniku. Nakon što dobije podatke o korisniku, provjerava postoji li korisnik unutar aplikacije i vraća JSON web-token s korisnikovim podacima. U daljnjoj komunikacija koristi se JSON web-token, dok će se Google OAuth2 autentifikacija ponoviti tek nakon što JSON web-token istekne.

credential = request.query\_params.get("credential")

if not credential:

return Response(

{"error": "Credential not provided"},

status=status.HTTP\_400\_BAD\_REQUEST,

)

decoded\_token = id\_token.verify\_oauth2\_token(

credential,

requests.Request(),

GOOGLE\_SCOPE\_ID)

email = decoded\_token.get("email")

if email and email.split("@")[1] == "estudent.hr":

try:

user = User.objects.get(username=email, deleted=0)

picture = decoded\_token.get("picture")

if not (user.photo == picture):

user.photo = picture

user.save()

refresh = RefreshToken.for\_user(user)

return Response(

{"accessToken": str(refresh.access\_token)},

status=status.HTTP\_200\_OK,)

except User.DoesNotExist:

return Response(

{"error": "User does not exist"},

status=status.HTTP\_404\_NOT\_FOUND,)

else:

return Response(

{"error": "Invalid email domain"},

status=status.HTTP\_400\_BAD\_REQUEST,)

Kôd 4.1 Dekodiranje Googleovog kôda i autorizacija korisnika

## Izrada poslužiteljskog sloja

Poslužiteljski sloj ima ključnu ulogu u provođenju poslovne logike i omogućuje komunikaciju između korisničkog i podatkovnog sloja. Kao veza između korisnika i podataka, ovaj sloj obrađuje korisničke zahtjeve, provjerava i obrađuje podatke te obavlja sve potrebne operacije kako bi osigurao ispravnost podataka. [12]

Uz poslovnu logiku, poslužiteljski sloj također upravlja autentifikacijom i autorizacijom korisnika. Tijekom autentifikacije provjerava se identitet korisnika kako bi se utvrdilo hoće li mu se omogućiti pristup sustavu. Nakon što korisnik uspješno prođe autentifikaciju, poslužiteljski sloj određuje njegovu razinu pristupa i omogućava mu pristup određenim funkcionalnostima.

Poslužiteljski sloj sustava za pregled novogradnje temelji se na REST API servisima. REST, kao arhitektonski stil, postavlja standarde za komunikaciju između računalnih sustava putem weba. REST sustavi, koji se odlikuju jasnom podjelom odgovornosti između klijenta i poslužitelja, koriste HTTP protokol za komunikaciju između poslužitelja i klijenta pri korištenju mrežnih resursa. [12]

HTTP standard definira pet skupina statusnih kôdova, klasificiranih prema prvoj znamenki troznamenkastog broja:

* 1xx: Informacija (engl. *Information*) – Zahtjev je zaprimljen i procesira se.
* 2xx: Uspjeh (engl. *Successful*) – Zahtjev je uspješno obavljen.
* 3xx: Redirekcija (engl .*Redirection*) – Zahtjev se odrađuje uz dodatnu akciju.
* 4xx: Greška klijenta (engl. *Client error*) – Zahtjev nije ispravno postavljen ili se ne može ispuniti prema poslanim podacima.
* 5xx: Interna greška poslužitelja (engl. *Internal server error*) – Zahtjev je valjan, ali se dogodila greška na poslužitelju.

U REST arhitekturi, klijenti i poslužitelji su neovisni, što znači da se mogu razvijati zasebno, bez detaljnog poznavanja implementacije druge strane. Sve dok oba dijela sustava razumiju format poruka za međusobnu komunikaciju, mogu raditi neovisno, što je ilustrirano na slici (Slika 4.1).

Diagram

Description automatically generated

Slika 4.6 Prikaz komunikacije različitih tehnologija s REST servisom[[1]](#footnote-1)

Neke od ključnih karakteristika REST servisa su:

* bezstanjnost (engl. *stateless*)
* mogućnost keširanja (engl. *cacheable*)
* jedinstveno sučelje (engl. *uniform interface*)
* dosljedno korištenje HTTP metoda
* korištenje XML i/ili JSON formata za razmjenu poruka.

Komunikacija s REST API-jem odvija se putem CRUD funkcija (engl. *Create, Read, Update, Delete*), koje predstavljaju osnovne operacije nad podacima. Ove operacije omogućuju korisnicima i sustavima manipulaciju resursima putem HTTP metoda na određenim URL-ovima. Svaka CRUD funkcija odgovara specifičnoj HTTP metodi i omogućava različite vrste interakcija s resursima. [12]

### Postavljanje poslužiteljskog sloja

Za implementaciju REST API servisa koristi se Django REST framework, koji se nakon početnog postavljanja dodatno konfigurira u datoteci settings.py. Ovdje se postavke odnose na CORS (Cross-Origin Resource Sharing), CSRF (Cross-Site Request Forgery), bazu podataka, JSON web-tokene, postavljanje Swaggera te kreiranje podaplikacija za pojedine module.

CORS je mehanizam temeljen na HTTP zaglavljima koji omogućuju poslužitelju da specificira bilo koje izvorište (domenu, shemu ili priključak) osim vlastitog, s kojeg preglednik smije dopustiti učitavanje resursa. Za postavljanje CORS-a potrebno je instalirati biblioteku *django-cors-headers* i dodati je u INSTALLED\_APPS Kôd 4.2 prikazuje dopuštena izvorišta koja se postavljaju unutar varijable CORS\_ALLOWED\_ORIGINS.

CORS\_ALLOWED\_ORIGINS = [

"https://esustav.estudent.hr",

"http://localhost:8700",

"http://localhost:3000",

"http://localhost:5173",

"http://127.0.0.1:3000",

]

Kôd 4.2 Specificirani izvori koji imaju pristup CORS-a

**CSRF** je sigurnosna ranjivost na webu koja omogućuje napadaču da navede korisnike na izvršenje radnji koje nisu namjeravali poduzeti. To djelomično zaobilazi politiku istog podrijetla, osmišljenu da spriječi međusobno ometanje različitih web-stranica. Za konfiguraciju CSRF zaštite nije potrebno instalirati vanjske alate jer Django REST framework već uključuje ovu funkcionalnost. Kôd 4.3 prikazuje dopuštena izvorišta za CSRF koji se postavljaju na identičan način kao i za CORS:

CSRF\_TRUSTED\_ORIGINS = [

"https://esustav.estudent.hr",

"http://localhost:8700",

"http://localhost:3000",

"http://localhost:5173",

"http://127.0.0.1:3000",

Kôd 4.3 Specificirani izvori koji imaju pristup CSRF-a

Nakon što je korisnicima omogućen pristup poslužiteljskom sloju, sljedeći je korak povezivanje tog sloja s bazom podataka. Konfiguracija baze podataka predstavlja ključni dio postavki za svaku Django aplikaciju jer određuje način na koji će aplikacija komunicirati s bazom podataka te kako će pohranjivati i dohvaćati podatke.

U Django projektu, postavke baze podataka definirane su u datoteci settings.py unutar varijable *DATABASES koju prikazuje Kôd 4.3*. Ove postavke određuju koji se tip baze podataka koristi (npr. PostgreSQL, MySQL, SQLite itd.) te potrebne parametre za povezivanje s tom bazom.

DATABASES = {

# docker POSTGRESs

"default": {

"ENGINE": "django.db.backends.postgresql",

"NAME": env("DATABASE\_NAME"),

"USER": env("DATABASE\_USER"),

"PASSWORD": env("DATABASE\_PASSWORD"),

"HOST": env("DATABASE\_HOST"),

"PORT": env("DATABASE\_PORT"),

}

# local MYSQLITE

# "default": {

# "ENGINE": "django.db.backends.sqlite3",

# "NAME": BASE\_DIR / "db.sqlite3",

# }

}

Kôd 4.4 Postavke baze podataka na poslužiteljskom sloju

Swagger je alat koji omogućuje detaljan opis strukture sustava API-ja na način da ga stroj može čitati. Ova sposobnost dokumentiranja API-ja predstavlja srž svega što Swagger čini izvanrednim. Kada Swagger očita strukturu API-ja, omogućuje automatsko generiranje atraktivne i interaktivne dokumentacije za API. Kao rezultat toga, stvara se web-stranica koja dokumentira svaku pristupnu točku aplikacije. Prikaz stranice se nalazi na Slici 4.2

A screenshot of a computer

Description automatically generated

Slika 4.7 Prikaz Swagger stranice

Postavljanje Swaggera odvija se uz pomoć vanjske biblioteke *drf\_cpectacular* koju je potrebno instalirati i dodati u *INSTALLED\_APPS* varijablu. Kako bi mogli pristupiti Swagger stranici, potrebno ju je dodijeliti na jednu pristupnu točku unutar *urls.py* datoteke.

Postavljanje Swagger postavki odvija se modificiranjem varijabli *SPECTACULAR\_SETTINGS i SWAGGER\_SETTINGS*  koje prikazuje Kôd 4.5

SPECTACULAR\_SETTINGS = {

"TITLE": "eSustav",

"DESCRIPTION": "Aplikacije za upravljanje sustavom eStudent",

"VERSION": "1.0.0",

"SERVE\_INCLUDE\_SCHEMA": False,

}

SWAGGER\_SETTINGS = {

"SECURITY\_DEFINITIONS": {"basic": {"type": "basic"}},

}

Kôd 4.5 Swagger postavke

Nakon što je projekt postavljen i povezan s bazom podataka, potrebno je pripremiti poslužiteljski sloj za logički dio projekta. Baš kao što je baza podataka podijeljena u sedam modula, i poslužiteljski će sloj biti podijeljen na sedam modula. Uz tih sedam modula, bit će uspostavljen direktorij koji će sadržavati sve enume, kao i direktorij posvećen autentifikaciji. Moduli se kreiranju kao Django pod-aplikacije i moraju biti dodane u *INSTALLED\_APPS.* Kôd 4.6 prikazuje finalnu konfiguraciju *INSTALLED\_APPS* varijable, a prikaz poslužiteljskog sloja može se vidjeti na Slici 4.3

INSTALLED\_APPS = [

"django.contrib.admin",

"django.contrib.auth",

"django.contrib.contenttypes",

"django.contrib.sessions",

"django.contrib.messages",

"django.contrib.staticfiles",

"django\_filters",

"rest\_framework",

"rest\_framework\_simplejwt",

"drf\_spectacular",

"corsheaders",

"estudenti",

"eizbori",

"eaktivnosti",

"einfo",

"epartneri",

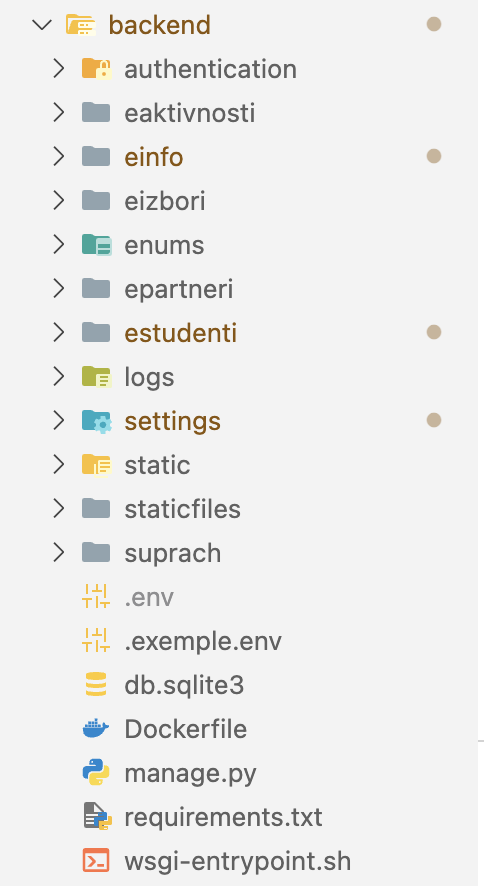
"suprach",

"logs",

"authentication",

]

Kôd 4.6 INSTALLED\_APPS konfiguracija



Slika 4.8 Direktoriji i njihova struktura unutar poslužiteljskog sloja aplikacije

### Podatkovni sloj unutar poslužiteljskog sloja

Svaki modul unutar poslužiteljskog sloja sadrži definirane ORM klase koje predstavljaju modele tablica u bazi podataka. Tablice u bazi podataka neće se kreirati sve dok se ne izvrše migracije koje prilagođavaju bazu podataka definiranim modelima.

Kôd 4.7 prikazuje jednu ORM klasu, pod imenom *AcademicYear* koja se trenutno nalazi u *estudenti* modulu. Nakon izvršenih migracija Klasa *AcdemicYear,* u bazi podataka kreirat će se tablica s nazivom *estudenti\_academicyear*. Iako će tablica nositi ovakav naziv, pristup podacima u aplikaciji odvijat će se putem naziva klase, što omogućuje intuitivniji i čišći rad s podacima.

class AcademicYear(models.Model):

start\_date = models.DateField()

end\_date = models.DateField()

description = models.CharField(max\_length=50, blank=True, null=True)

short = models.CharField(max\_length=50, blank=True, null=True)

active = models.BooleanField(default=False)

deleted = models.BooleanField(default=False)

def \_\_str\_\_(self) -> str:

return f"{self.description}"

Kôd 4.7 ORM klasa AcademicYear

Django ORM modul pruža ključnu prednost u vidu validacije podataka temeljenih na definiranim ORM klasama, a ova validacija posebno dolazi do izražaja unutar Django REST frameworka pomoću serijalizatora. Serijalizator je komponenta unutar Django REST Frameworka koja omogućuje pretvaranje složenih podataka u formate koji su lako razumljivi i korišteni izvan same aplikacije, poput JSON-a ili XML-a. Njegova je glavna funkcija serijalizacija, odnosno pretvaranje kompleksnih objekata, kao što su skupovi upita (engl. *querysets*) i instance modela, u jednostavne Python tipove podataka. To omogućuje njihovu lakoću prikazivanja i razmjenu između različitih sustava ili API-ja.

Osim serijalizacije, serijalizatori također omogućuju deserijalizaciju, proces u kojem se podaci iz jednostavnih formata, kao što su JSON ili XML, ponovno pretvaraju u složene tipove podataka, poput ORM instanci modela. Ovaj je proces ključan jer omogućava REST API-ju da prima podatke od vanjskih klijenata u formatu, kao što je JSON, a zatim ih vraća u oblik koji Django aplikacija može koristiti za daljnju obradu ili pohranu u bazu podataka.

Uz ove funkcije, serijalizatori također pružaju robustan mehanizam za validaciju podataka, osiguravajući da su svi podaci ispravni i u skladu s pravilima definiranim unutar ORM klasa prije nego što se pohrane u bazu podataka. Ova kombinacija funkcionalnosti čini serijalizatore svestranim alatom koji omogućuje pouzdan rad aplikacije s ispravnim i validnim podacima, osiguravajući da svi podaci koji prolaze kroz API budu adekvatno provjereni i formatirani prije daljnje upotrebe. [13]

Kôd 4.8 prikazuje serijalizator za *AcademicYear* tablicu koji će od dobivenih podataka vratiti samo podatke koji odgovaraju *fields* parametrima. Parametri se moraju nalaziti unutar ORM klase inače će se dogoditi pogreška.

class AcademicYearSerializer(serializers.ModelSerializer):

class Meta:

model = AcademicYear

fields = "\_\_all\_\_"

Kôd 4.8 AcademicYear serializator

Ako želimo da naš serijalizator uključi povezane podatke, a ne samo ID, onda moramo povezati njegov serijalizator s parametrom u tablici. Jedan takav primjer prikazuje Kôd 4.9

class RolesSerializer(serializers.ModelSerializer):

role\_group = RoleGroupsSerializer()

class Meta:

model = Roles

fields = "\_\_all\_\_"

Kôd 4.9 Serijalizator s povezanim serijalizatorom na parametru role\_group

### Izrada poslovne logike unutar poslužiteljskog sloja

Sloj poslovne logike je komponenta softverske arhitekture koja je odgovorna za implementaciju poslovne logike aplikacije. Nalazi se između prezentacijskog sloja (npr. korisničkog sučelja) i sloja za pristup podacima (npr. baze podataka) te je odgovoran za obradu i manipuliranje podacima prije nego što se prezentiraju korisniku ili pohrane u bazi podataka. Glavne funkcije logičnog sloja su validacija podataka i poslovnih pravila, transformacija podataka, integracija s vanjskim sustavima i implementacija poslovnih procesa. [14]

Kako bi poslužiteljski sloj i njegovi klijenti mogli učinkovito komunicirati i razmjenjivati podatke prilikom slanja HTTP zahtjeva određenoj pristupnoj točki, koristi se JSON format. JSON je format razmjene podataka koji je čitljiv ljudima i opisuje entitet u obliku parova atribut – vrijednost. Atribut označava specifično svojstvo entiteta, dok vrijednost predstavlja vrijednost tog svojstva. Kada klijent želi dohvatiti podatke putem REST API servisa, šalje GET metodu na željenu pristupnu točku. Poslužiteljski sloj potom dohvaća podatke iz baze podataka i serijalizira ih u JSON format. Nakon što klijent zaprimi odgovor, deserijalizira JSON podatke u objekte koje može dalje procesirati. Ako klijent želi stvoriti ili ažurirati postojeći entitet putem REST API servisa, serijalizira željene podatke u JSON format i koristi POST ili PUT metodu za slanje HTTP zahtjeva koji sadržava JSON format u tijelu zahtjeva. Servis tada deserijalizira primljene podatke u objekte koje može procesirati, omogućujući daljnju komunikaciju s bazom podataka. [13]

U sustavima temeljenim na MVC arhitekturi (engl. *Model-View-Controller*), svaki od tri glavna elementa ima svoju specifičnu ulogu. *Model* predstavlja sloj koji upravlja podacima, njihovom strukturom i poslovnim pravilima koja definiraju kako se podaci trebaju ponašati. *View* je sloj koji prikazuje podatke korisniku, odnosno odgovoran je za prezentaciju podataka i interakciju s korisnikom. *Controller* služi kao posrednik između *Modela* i *Viewa*, upravlja logikom aplikacije i definira način na koji se podaci prikazani u *Viewu* trebaju obrađivati.

U ovom kontekstu, sustav koristi poglede, koji u MVC arhitekturi igraju ključnu ulogu u implementaciji poslovne logike i definiranju pristupnih točaka REST API-ja za CRUD operacije na modelima podataka. Pritom se koriste *ModelViewSet* i *GenericViewSet* pogledi, koji omogućuju jednostavniju implementaciju i ponovnu upotrebu kôda, olakšavajući razvoj funkcionalnosti sustava u skladu s poslovnim pravilima.

*ModelViewSet* u Django REST frameworku automatski generira pristupne točke za sve CRUD operacije, uključujući stvaranje, dohvaćanje, ažuriranje i brisanje objekata. Time se olakšava rad s modelima jer nije potrebno dodatno pisanje kôda za svaku od tih funkcionalnosti. S druge strane, GenericViewSet pruža osnovne metode za dohvaćanje podataka, ali ne generira automatski pristupne točke za CRUD operacije. Unutar svakog pogleda postoje funkcije koje predstavljaju određene CRUD operacije, kao što je prikazano u Tablici 4.1. Kako bi se omogućila određena operacija nad pristupnom točkom u GenericViewSetu, potrebno je nadjačati funkciju drugom. Nadjačavanje funkcija također je moguće u ModelViewSetu.

|  |  |
| --- | --- |
| CRUD operacija | Naziv funkcije unutar pogleda |
| GET all | list |
| POST | create |
| GET by id | retrieve |
| PUT | update |
| PATCH | partial\_update |
| DELETE | destroy |

Tablica 4.1 Funkcije koje predstavljaju CRUD operacije unutar pogleda

class TeamsView(viewsets.ModelViewSet):

queryset = Teams.objects.all()

serializer\_class = TeamsSerializer

permission\_classes = [IsAuthenticated]

Kôd 4.10 ModelViewSet pogled

Određene funkcije mogu se proširiti posebnim funkcijama koje se uz @action anotaciju pretvaraju u proširene pristupne točke. Ova anotacija služi za definiranje naziva pristupne točke i tip metode koju će ta točka podržavati.

class TeamsView(viewsets.ModelViewSet):

queryset = Teams.objects.all()

serializer\_class = TeamsSerializer

permission\_classes = [IsAuthenticated]

@action(

methods=["GET"],

detail=False,

url\_path="get-my-team",

)

def get\_my\_team(self, request):

try:

academic\_year = AcademicYear.objects.get(active=True)

user\_position = UsersPositions.objects.get(

user=self.request.user, academic\_year=academic\_year

)

queryset = UsersPositions.objects.filter(

team=user\_position.team,

role=Roles.objects.get(name=RolesEnum.CLAN.value),

)

serializer = UserPositionSerializer(queryset, many=True)

return Response(serializer.data, status=status.HTTP\_200\_OK)

except Exception as e:

ErrorLogs.objects.create(error=str(e), user=self.request.user)

return Response(status=status.HTTP\_400\_BAD\_REQUEST)

Kôd 4.11 Proširivanje pogleda

U određenim modulima datoteka s pogledima je s vremenom postala prevelika. Takvi moduli poput *einfo*, *eizbori* i *suprach* dobili su zaseban direktorij naziva *views* u kojoj se nalaze svi pogledi na tom modulu. Na taj način pristup logici pojedine pristupne točke je puno brži i jednostavniji jer se datoteke nazivaju prema nazivu pogleda.

Unutar svakog pogleda obavlja se manipulacija podataka pomoću **QuerySet** sistema. **QuerySet predstavlja kolekciju objekata iz naše baze podataka. Može imati nula, jedan ili više filtera. Filtri sužavaju rezultate upita na temelju zadanih parametara. U terminima SQL-a, QuerySet izjednačuje se s SELECT naredbom, a filter je produžetak SELECT-a s WHERE ili LIMIT.** Svaki QuerySet je lijen, što znači da se upiti prema bazi podataka izvršavaju tek kada se rezultati QuerySet-a eksplicitno zatraže. Dohvaćanje podataka iz QuerySeta je vrlo jednostavan jer se on ponašao kao iterativni objekt. Primjer manipulacije podataka prikazuje Kôd 4.12, gdje dohvaćamo sve partnere i kratke nazive projekata na kojima su surađivali. Podaci se pohranjuju u novu listu koja se zatim šalje klijentu koji je zatražio podatke.

class PartnersView(viewsets.ModelViewSet):

queryset = Partners.objects.all()

serializer\_class = PartnersSerializers

permission\_classes = [IsAuthenticated]

def list(self, request):

try:

return\_data = []

partners = Partners.objects.all()

for partner in partners:

project\_data = (

PartnerNotes.objects.filter(

partner=partner,

project\_\_is\_deleted=False

)

.values\_list(

"project\_\_short\_project\_name")

.distinct()

)

partner\_serializer =

PartnersSerializers(partner)

return\_data.append(

{

"partner": partner\_serializer.data,

"projects": project\_data

}

)

return Response(

return\_data,

status=status.HTTP\_200\_OK

)

except Exception as e:

ErrorLogs.objects.create(

error=str(e),

user=self.request.user)

return Response(

status=status.HTTP\_400\_BAD\_REQUEST

)

Kôd 4.12 PartnersView pristupna točka

Da bi klijent mogao poslati HTTP zahtjev prema serveru, potrebno je povezati pogled s odgovarajućom pristupnom točkom. Pristupne točke obično slijede određenu strukturu URL-a koja se sastoji od prefiksa i imenice koja jednoznačno identificira entitet koji se koristi u komunikaciji. Uobičajena praksa je da se prefiks */api/v1/* koristi kao početni dio URL-a, nakon kojeg slijedi naziv entiteta ili modula, npr. */api/v1/users/. Ovaj* pristup omogućava jasnu identifikaciju da klijent komunicira s API servisom, a verzioniranje omogućava fleksibilnost u razvoju.

Budući da je naš server podijeljen u module, prefiksi pristupnih točaka također će odražavati tu strukturu pa će URL izgledati ovako: */api/v1/{naziv\_modula}/.* Verzija API-ja uključena je u URL kako bi se klijentima jasno naznačilo s kojom verzijom API-ja komuniciraju. Verzioniranje je korisno jer omogućava postojanje više verzija API-ja s istim ili sličnim funkcionalnostima. Na primjer, ako se implementira nova verzija API-ja s poboljšanjima ili promjenama, koristit će prefiks */api/v2/{naziv\_modula}/.* Klijenti, koji žele koristiti novu verziju API-ja, jednostavno će promijeniti verzijski broj u URL-u, što predstavlja minimalnu promjenu na strani klijenta.

Svaki modul definira svoje pristupne točke u vlastitoj *urls.py* datoteci, gdje povezuje odgovarajuće poglede s URL-ovima. Primjer povezivanja prikazan je u sljedećem kôdu (Kôd 4.13)

router = routers.DefaultRouter()

router.register("users", views.UserView, basename="users")

router.register("teams", views.TeamsView, basename="teams")

router.register("teamsgroup", views.TeamsGroupView, basename="teamsgroup")

router.register("virtualteam", views.VirtualTeamsView, basename="virtualteam")

router.register("academic-year", views.AcademicYearView, basename="academic-year")

urlpatterns = [

path("", include(router.urls)),

]

Kôd 4.13 Povezivanje pogleda i pristupnih točaka

Ovaj kôd povezuje različite poglede s pripadajućim URL-ovima. Na taj način, klijentima su omogućene različite CRUD operacije nad entitetima putem REST API-ja.

Međutim, klijent ne može pristupiti ovim HTTP zahtjevima sve dok se modulske *urls.py* datoteke ne povežu s glavnim URL-om aplikacije. Ova povezanost postiže se u glavnoj *urls.py* datoteci unutar modula *settings*. Tamo se definira glavni prefiks */api/v1/{naziv\_modula}/* na način prikazan u sljedećem primjeru kôda (Kôd 4.14)

urlpatterns = [

path("api/v1/admin/", admin.site.urls),

path("api/v1/auth/", include("authentication.urls")),

path("api/v1/estudenti/", include("estudenti.urls")),

path("api/v1/eizbori/", include("eizbori.urls")),

path("api/v1/eaktivnosti/", include("eaktivnosti.urls")),

path("api/v1/einfo/", include("einfo.urls")),

path("api/v1/epartneri/", include("epartneri.urls")),

path("api/v1/suprach/", include("suprach.urls")),

path("api/v1/schema/", SpectacularAPIView.as\_view(), name="schema"),

path(

"api/v1/docs/",

SpectacularSwaggerView.as\_view(url\_name="schema"),

name="swagger-ui",

),

]

Kôd 4.14 Povezivanje modula s glavnim prefiksom

Ovim načinom osigurava se da svaki modul u aplikaciji ima vlastiti prefiks unutar API-ja, čime se olakšava organizacija i verzioniranje API-ja. Svi zahtjevi koji dolaze na */api/v1/{naziv\_modula}/* bit će proslijeđeni odgovarajućem modulu i obradi putem pripadajućih pogleda, čime je cijeli proces komunikacije između klijenta i servera u potpunosti zaokružen i funkcionalan.

### Postavljanje Django administratorske stranice unutar poslužiteljskog sloja

Django administrativna aplikacija omogućava automatsko generiranje administracijskog sučelja za rad s našim modelima. Ova značajka uvelike olakšava stvaranje, pregled, ažuriranje i brisanje zapisa tijekom razvoja, čime štedi vrijeme i pojednostavljuje testiranje modela. Kroz administrativno sučelje, lako možemo dobiti uvid u podatke i provjeriti jesu li ispravno strukturirani. Iako je ova aplikacija izuzetno korisna tijekom razvoja, može se koristiti i za upravljanje podacima u produkciji, ovisno o specifičnostima web-stranice. Međutim, Django preporučuje korištenje administrativnog sučelja samo za interno upravljanje podacima (tj. za administratore ili članove organizacije) budući da pristup usmjeren na model može izložiti previše detalja koji nisu potrebni krajnjim korisnicima. [13]

A screenshot of a computer

Description automatically generated

Kôd 4.15 Django administratorsko sučelje

Prilikom izrade projekta, Django automatski konfigurira sve potrebne komponente za uključivanje administrativne aplikacije u sustav. Da bi se modeli učinili dostupnima u administrativnom sučelju, potrebno ih je registrirati unutar odgovarajuće datoteke admin.py, koja se nalazi u svakom modulu. Registracijom modela omogućuje se njihovo prikazivanje i manipulacija unutar administrativnog sučelja.

Prema zadanim postavkama, Django prikazuje samo ID zapisa u administracijskom sučelju osim ako se ne otvori detaljan prikaz zapisa. Kako bi se poboljšala preglednost, nadjačava se funkcija *\_\_str\_\_* unutar svakog modela, čime se određuje kako će se zapisi prikazivati. Još bolja opcija je prikaz zapisa u obliku tablice s definiranim stupcima, što se postiže definiranjem klase *ModelAdmin* unutar datoteke *admin.py* za svaki model. Ova klasa omogućuje prikupljanje i prikaz ključnih elemenata modela u tabličnom formatu, čime se olakšava pregled i upravljanje podacima.

Osim prilagodbe prikaza, moguće je definirati funkcije koje će se izvršavati nad označenim zapisima. Sustav uključuje funkciju koja služi za dodjelu prava glasa na Skupštini ako korisnik ima pravo glasa na izborima. Primjer spomenutih modifikacija prikazan je u kôdu (Kôd 4.15).

@admin.action(description="Set Can Vote on True/False")

def set\_can\_vote\_on\_true(modeladmin, request, queryset):

for obj in queryset:

obj.can\_vote = not obj.can\_vote

obj.save()

class UserPermissionsAdmin(admin.ModelAdmin):

list\_display = (

"user",

"academic\_year",

"info",

"aktivnosti",

"partneri",

"izbori",

"suprach",

"suprach\_admin",

"can\_vote",

)

actions = [set\_can\_vote\_on\_true]

admin.site.register(UserPermissions, UserPermissionsAdmin)

Kôd 4.16 Registracija UserPermissions modela i optimizacija prikaza

## Izrada korisničkog sučelja sustava

Korisničko sučelje predloženog sustava razvijeno je koristeći React, uz pomoć Vite lokalnog razvojnog poslužitelja, koji omogućuje brzu i efikasnu izradu aplikacija. Kao temeljni alat za dizajn komponenti, koristi se Ant Design, popularna biblioteka za dizajn korisničkog sučelja koja je posebno razvijena za rad s Reactom. Ant Design pruža širok spektar predefiniranih elemenata korisničkog sučelja, uključujući dugmad, obrazaca, izbornika, tablica i mnogih drugih komponenti. Ove su komponente pažljivo stilizirane i optimizirane kako bi se osigurala dosljednost u dizajnu, visoka razina korisničkog iskustva i jednostavnost korištenja. Ant Design također omogućuje prilagodbu komponenti prema specifičnim potrebama projekta kroz fleksibilne opcije konfiguracije i stiliziranja, što pomaže u izradi profesionalnog i estetski privlačnog sučelja. Korištenje ove biblioteke značajno ubrzava razvoj, smanjuje potrebu za pisanjem vlastitog izgleda i osigurava usklađenost sa suvremenim dizajnerskim standardima.

### Postavljanje korisničkog sučelja sustava

Korisničko sučelje je izgrađeno koristeći TypeScript kao osnovni programski jezik. TypeScript, kao statički tipizirani nad-skup JavaScripta, omogućuje bolju kontrolu nad tipovima podataka i smanjuje mogućnost pogrešaka u kôdu. Usto, koristi se ESLint, alat za statičku analizu kôda, koji pomaže u pronalaženju problematičnih obrazaca u JavaScriptu, poput mogućih grešaka ili nepravilnosti u stilu kodiranja. Ova konfiguracija osigurava visoku kvalitetu kôda i održavanje najboljih praksi u razvoju.

Za razliku od složenijeg poslužiteljskoga sučelja, korisničko sučelje sastoji se od samo šest glavnih modula: *einfo*, *eaktivnosti*, *eizbori*, *epartneri*, *suprach* i *eadmin*. Ovaj modularni pristup omogućuje bolju organizaciju kôda i olakšava održavanje aplikacije jer se svaki dio aplikacije razvija i testira zasebno.

Projekt je strukturiran na način da se sve stranice, osim glavnih (kao što su *main.tsx* i *App*.*tsx*), nalaze unutar direktorija *pages*. Unutar ovog direktorija, stranice su organizirane prema modulima u šest pod-direktorija. Svaki pod-direktorij sadrži stranice povezane s odgovarajućim modulom. Izvan ovih modula nalaze se početna stranica i stranica za prijavu korisnika.

Komponente, koje se koriste unutar stranica, smještene su unutar direktorija *components*. Taj direktorij je također organiziran u pod-direktorije prema modulima, s ciljem grupiranja sličnih komponenti i olakšavanja navigacije kroz kôd. Sve su komponente opremljene mehanizmom za *lazy* učitavanje, što znači da se komponente učitavaju samo kada su potrebne, čime se poboljšava performansa aplikacije.

Sve stranice i komponente renderiraju se unutar glavnog layouta, koji se nalazi u direktoriju *layouts*. Glavni layout pruža osnovni okvir aplikacije, u kojem se dinamički prikazuju različite stranice i komponente, ovisno o trenutnom stanju aplikacije.

Konfiguracije za pojedine komponente, stranice i rute nalaze se u direktoriju *configurations*. Ove konfiguracije su izuzetno važne, posebno za Ant Design, koji se uvelike oslanja na napredne konfiguracije u obliku JSON varijabli. Iz tog razloga, sve važne konfiguracije su izdvojene u zasebni direktorij kako bi se osigurala njihova preglednost i lakoća pristupa.

Stilovi i tipovi podataka su organizirani u zasebnim direktorijima, pri čemu svaki modul ima svoju odgovarajuću datoteku. Iako stilovi i tipovi nisu podijeljeni u pod-direktorije, njihova struktura osigurava konzistentnost i jasnoću u organizaciji kôda, što je ključno za održavanje velikih projekata.

Korisničko sučelje koristi vlastiti context za čuvanje podataka o korisniku, što omogućuje centralizirano upravljanje korisničkim informacijama i olakšava pristup tim podacima kroz različite dijelove aplikacije. Također, postoji i Axios context, koji služi za postavljanje glavnih postavki za komunikaciju s poslužiteljskim slojem. Axios, kao popularna biblioteka za HTTP zahtjeve, koristi se za dohvaćanje podataka s poslužitelja, a svi API pozivi su organizirani u obliku React hookova. Svi hookovi koji se koriste za dohvaćanje podataka nalaze se unutar direktorija *hooks*, što dodatno doprinosi organizaciji i ponovnoj upotrebljivosti kôda.

### Dozvole korisnika kroz korisničko sučelje

Jedan od glavnih problema u sustavu bio je precizno postavljanje i upravljanje dozvolama korisnika na temelju njihovih pozicija i timova unutar Udruge. Sustav uključuje različite module, a svaki korisnik, ovisno o svojoj ulozi kao član, voditelj ili koordinator, treba imati specifičan pristup samo određenim funkcionalnostima unutar tih modula ako uopće ima pristup modulu. Dok je član tima, na primjer, ograničen na pregled vlastitih aktivnosti, voditelj može imati potrebu za upravljanjem i pregledom aktivnosti cijelog tima. Ova kompleksnost u definiranju i implementaciji pravila pristupa dovela je do izazova u održavanju pravilnih dozvola, što može rezultirati neautoriziranim pristupom ili neadekvatnim pravima, čime se ugrožava sigurnost i integritet podataka. Također, promjene u strukturi tima, kao što su novi članovi ili promjene u ulozi postojećih korisnika, zahtijevaju stalno ažuriranje prava pristupa, što dodatno komplicira upravljanje dozvolama i zahtijeva pažljivo praćenje i prilagodbu sustava. Tablica 4.2 prikazuje uvjete po modulu i poziciji.

|  |  |  |
| --- | --- | --- |
| Modul | Pozicija | Treba li korisnik imati pristup? |
| eAKTIVNOSTI | Član | Ne, osim članovima evaluacijskog tima koji smiju samo pregledavati aktivnosti. |
| eAKTIVNOSTI | Voditelj | Da, smije uređivati samo aktivnosti svog tima i ima mogućnost unošenja pozitivne ocjene rada članovima tima. |
| eAKTIVNOSTI | Koordinator | Da, smije uređivati samo aktivnosti svoje skupine i ima mogućnost unošenja pozitivne ocjene rada voditeljima skupine. |
| eINFO | Član | Da, pregled samo svojih aktivnosti. |
| eINFO | Voditelj | Da, pregled svojih aktivnosti i aktivnosti svojih članova. |
| eINFO | Koordinator | Da, pregled svojih aktivnosti i aktivnosti svojih voditelja. |
| eINFO | Admin | Da, može postavljati uvjete za certifikat. |
| ePARTNERI | Član | Ne, nema pristup modulu. |
| ePARTNERI | Voditelj, koordinator | Da, puni pristup. |
| eIZBORI | Član | Da, pravo glava ovisi o tipu izbora. Pristup glasovanju ako se bira voditelj njegovog tima.  Pristup glasovanju ako je fizički prisutan na Skupštini. |
| eIZBORI | Voditelj | Da, pravo glava ovisi o tipu izbora. Pristup glasovanju ako se bira voditelj njegovog tima.  Pristup glasovanju ako je fizički prisutan na Skupštini. |
| eIZBORI | Koordinator | Da, pravo glava ovisi o tipu izbora. Pristup glasovanju ako se bira voditelj njegove skupine (skupina može imati više timova).  Pristup glasovanju ako je fizički prisutan na Skupštini. |
| eIZBORI | Admin | Da, pravo da vidi rezultate izbora. |
| SUPRACH | Član | Da, ocjenjuje svoj tim, voditelje i specijalne grupe. |
| SUPRACH | Voditelj | Da, ocjenjuje svoj tim, voditelje, koordinatore i specijalne grupe. |
| SUPRACH | Koordinator | Da, ocjenjuje svoje voditelje u skupini, Predsjedništvo i specijalne grupe. |
| SUPRACH | Admin | Da, uvid u podatke tko nije ispunio SUPRACH. |
| eADMIN | Član, voditelj, koordinator | Ne. |
| eADMIN | Admin | Da. |

Tablica 4.2 Popis dozvola korisnika unutar korisničkog sučelja

Dozvole korisnika u sustavu funkcioniraju tako da se nakon autentifikacije putem Google OAuth2 dohvaćaju podaci o korisniku koji određuju njegova prava pristupa unutar aplikacije. Dok se podaci ne preuzmu, korisnik nema pristup aplikaciji niti se stranice konfiguriraju. Nakon uspješnog preuzimanja podataka, učitava se *MainLayout*, koji prema korisničkim dozvolama konfigurira dostupne putanje i module na početnoj stranici, omogućujući korisnicima pristup samo onim dijelovima aplikacije za koje su ovlašteni.

A screenshot of a computer

Description automatically generated

Slika 4.9 Početna stranica sustava

Svaka stranica prilikom učitavanja dodatno provjerava dozvole pristupa kako bi se osiguralo da se slučajno putanje nisu krivo konfigurirale. Uvjeti prikaza po modulu postavljeni su na početnoj stranici modula. Kako to izgleda unutar pojedinog modula, prikazuje Kôd 4.16.

<div className="footerSize">

{user?.role.name === RoleEnum.Clan ? (

<Suspense fallback={<Spin size="small" />}>

<ActivityList />

</Suspense>

) : (

<>

<Segmented

className="segmented"

options={[

{ label: "Unos aktivnosti", value: "activity" },

{ label: "Pozitivne ocjene rada", value: "recommendations" },

]}

onChange={(value) => setPage(value)}

/>

{page === "activity" ? (

<Suspense fallback={<Spin size="small" />}>

<ActivityList />

</Suspense>

) : (

<Suspense fallback={<Spin size="small" />}>

<Recommendations />

</Suspense>

)}

</>

)}

</div>

Kôd 4.17 Dozvole unutar modula eAKTIVNOSTI

Pojedine radnje odvijaju se unutar nekih komponenti pa su radi toga pojedine dozvole definirane unutar konfiguracija komponenti, poput dozvole smije li korisnik uređivati aktivnost koja je definirana unutar konfiguracije za tablicu aktivnosti.

{

title: "",

dataIndex: "",

render: (\_, record) => (

<>

{(user?.role.name === RoleEnum.Voditelj ||

user?.role.name === RoleEnum.Koordinator) &&

(user.team.id === record.team.id ||

user?.team\_group.name === "Informacijske tehnologije" ||

(user?.role.name === RoleEnum.Koordinator &&

user.team\_group.id === record.team.TeamGroups.id)) ? (

<div onClick={(e) => e.stopPropagation()}>

<QrCodeModal activity={record} />

<EditActivity activity={record} />

<DeleteActivity activityID={record.id} />

</div>

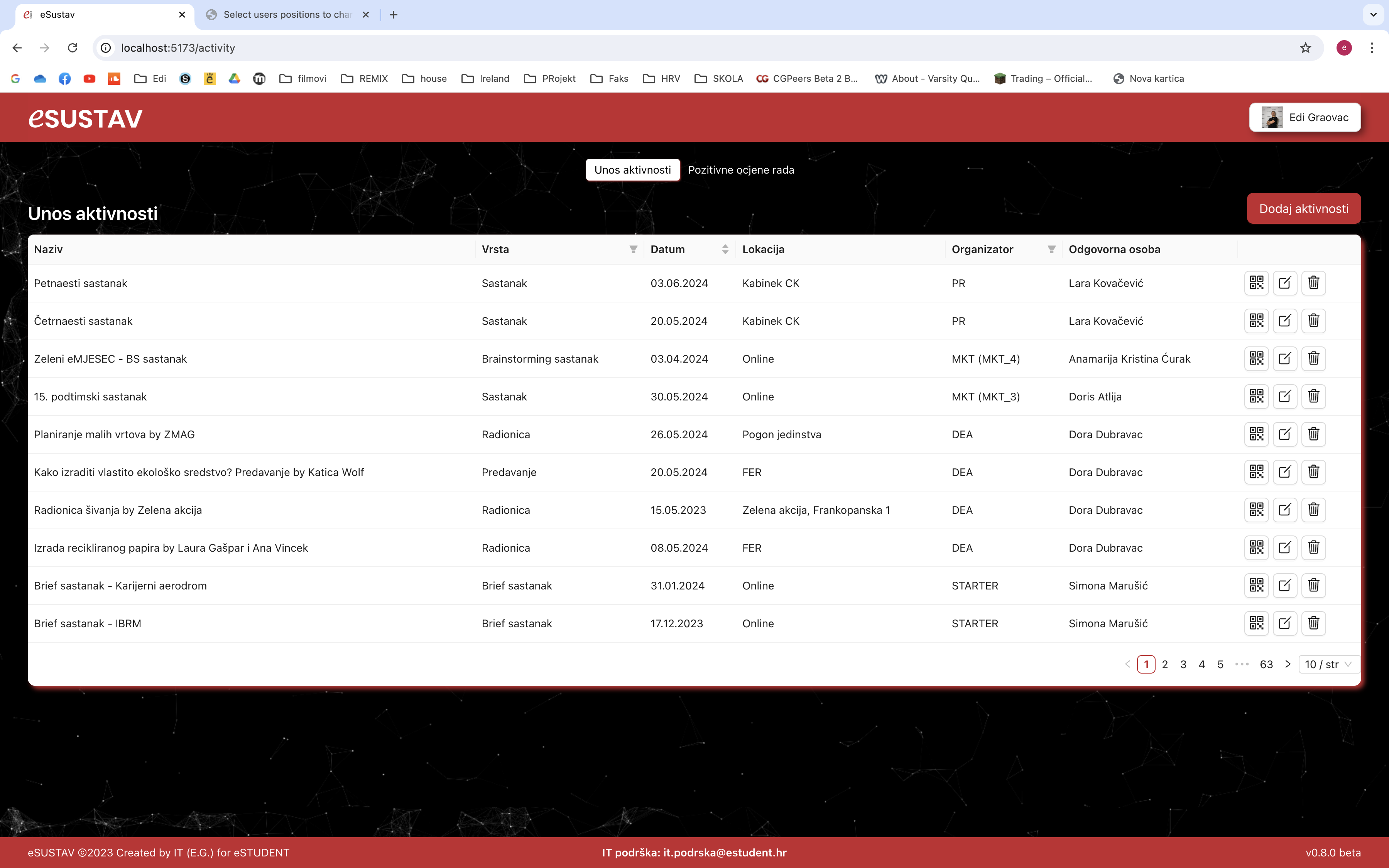
) : null}

</>

),

},

Kôd 4.18 Dozvole unutar tablice aktivnosti



Kôd 4.19 Tablica aktivnosti unutar modula eAKTIVNOSTI

### Konfiguracija komunikacije korisničkog sučelja i poslužiteljskog sloja

U sustavu, komunikacija između klijenta i poslužitelja upravlja se putem Axios modula, koji olakšava slanje HTTP zahtjeva i obradu odgovora. Kako bismo unaprijedili organizaciju kôda i osigurali dosljednost u rukovanju zahtjevima i odgovorima, implementirali smo Axios Context. [5]

Unutar Axios Contexta smo konfigurirali da se na svaki HTTP zahtjev automatski dodaje autorizacijski token u zaglavlje. Ovo omogućava provjeru identiteta korisnika i pristup samo ovlaštenim resursima. Axios Context također obrađuje odgovore s poslužitelja. Ako se u odgovoru detektira status kôd 401, što označava neovlašten pristup, korisnik se automatski preusmjerava na stranicu za prijavu. Ovo osigurava da korisnici bez valjanih vjerodajnica ne mogu pristupiti aplikaciji. Konfiguracija Axios Contexta može se vidjeti u kôdu (Kôd 4.20)

const auth = useMemo(() => {

// const axios = Axios.create({ baseURL: "https://esustav.estudent.hr/api/" }); //deployed

const axios = Axios.create({ baseURL: "http://127.0.0.1:8000/api/" }); //local

// const axios = Axios.create({ baseURL: "http://127.0.0.1:8700/api/" }); //docker

let tokenData: string | null = sessionStorage.getItem("accessToken");

axios.interceptors.request.use((config) => {

if (tokenData) {

config.headers.Authorization = `Bearer ${tokenData}`;

}

return config;

});

axios.interceptors.response.use(

(response) => {

return response;

},

(error) => {

if (error.response?.status === 401) {

sessionStorage.removeItem("accessToken");

if (window.location.pathname !== "/") {

window.location.reload();

}

console.log(error);

}

}

);

return axios;

}, [navigate]);

Kôd 4.20 Konfiguracija Axios Contexta

Korištenjem Axios Contexta smanjuje se potreba za ponavljanjem kôda i centralizira se logika autentifikacije i upravljanja pogreškama, čime se poboljšava efikasnost i sigurnost aplikacije.

U našem sustavu, komunikacija s poslužiteljem odvija se putem Axios modula koji je integriran u React hookove kako bi se upravljalo HTTP zahtjevima i odgovorima. Ovi hookovi omogućuju jednostavno dohvaćanje i slanje podataka te upravljanje statusom zahtjeva unutar aplikacije. U sustavu razlikujemo dvije vrste hookova za komunikaciju. Za dohvaćanje podataka i za slanje podataka, dok ta dva hooka potom unutar sebe razlikuju zahtjeve. *Hook* za dohvaćanje podataka ima mogućnost GET i DELETE zahtjeva, dok hook za slanje podataka može odraditi sve druge zahtjeve PUT; PATCH i POST.

Za dohvaćanje podataka s poslužitelja koristimo *useQuery* iz *react-query*. Ovaj hook koristi funkciju za slanje GET i DELETE zahtjeva putem Axios instance. Za primjer, možemo uzeti Kôd 4.21 gdje useGetAllTeams dohvaća sve timove s poslužitelja, provjerava valjanost autorizacijskog tokena prije slanja zahtjeva i upravlja pogreškama ako dođe do problema prilikom dohvaćanja podataka. useQuery također omogućuje upravljanje pred memorijom i automatsko osvježavanje podataka kada se promijeni valjanost querya.

export const useGetAllTeams = () => {

const axios = useAxios();

const fetchTeamsAllData = async () => {

if (sessionStorage.getItem("accessToken") !== null) {

try {

const { data } = await axios.get("estudenti/teams/");

return data;

} catch (error) {

throw new Error("Nije moguće dohvatiti podatke");

}

}

return null;

};

return useQuery("teams\_data", fetchTeamsAllData, {

onError: (error) => console.log("error", error),

staleTime: Infinity,

});

};

Kôd 4.21 Primjer hooka za dohvaćanje podataka

Unutar hooka za slanje podataka na poslužitelj koristimo *useMutation* iz *react-query*. Ovaj hook omogućava slanje POST, PUT i PATCH zahtjeva putem Axios instance i upravljanje odgovorima. Kao primjer, možemo uzeti Kôd 4.22, gdje useLikeUser koristi POST zahtjev za označavanje korisnika kao da je ocijenjen. Nakon uspješnog slanja podataka, hook automatski ažurira relevantne podatke u kešu pomoću queryClient.invalidateQueries i obavještava korisnika o uspjehu putem poruka.

export const useLikeUser = () => {

const axios = useAxios();

const queryClient = useQueryClient();

const likeUser = async (LikesData: LikesType) => {

try {

const response = await axios.post("suprach/likes/", LikesData);

return response.data;

} catch (error) {

throw new Error("Greška prilikom lajkanja osobe");

}

};

return useMutation(likeUser, {

onSuccess: () => {

queryClient.invalidateQueries({ queryKey: ["user\_to\_grade"] });

message.success("Osoba je lajkana");

},

});

};

Kôd 4.22 Primjer hooka za slanje podataka

### Vizualni identitet korisničkog sučelja

Stilski vodič (engl. Brand guide) Udruge postavlja jasne smjernice za vizualni identitet i komunikaciju, osiguravajući dosljednost u svim aspektima sustava. Ovaj vodič obuhvaća niz elemenata koji definiraju kako se brend Udruge treba percipirati, uključujući vizualne i verbalne komponente koje zajedno stvaraju prepoznatljivost i jačaju identitet brenda. Zapravo, stilski vodič odražava ključne poruke i vrijednosti koje Udruga želi komunicirati, pomažući u stvaranju povezane i inspirativne slike o brendu.

Prema ovom vodiču, aplikacija je dizajnirana s dominantnim korištenjem crvene i bijele boje, koje su pažljivo odabrane kako bi prenijele energiju, odlučnost te čistoću i jednostavnost. Točne vrijednosti ovih boja definirane su u stilskom vodiču, a njihovu primjenu možemo vidjeti kroz cijeli korisnički interfejs, osiguravajući vizualnu dosljednost i sklad. Slika 4.5 prikazuje specifične nijanse crvene, crne i bijele boje, pružajući referencu za sve vizualne materijale povezane s brendom.

A close-up of a color chart

Description automatically generated

Slika 4.10 Paleta eSTUDENT boja

Jedan od ključnih elemenata vizualnog identiteta Udruge je i pozadina aplikacije, koja se temelji na glavnom sloganu Udruge: "Network of Excellence". Ovaj slogan simbolizira povezanost, zajedništvo i vrhunsku kvalitetu, a vizualno je predstavljen kroz pozadinsku sliku zvijezda na noćnom nebu, koje su međusobno povezane linijama. Slika 4.6 detaljno prikazuje ovu pozadinu, koja ne samo što obogaćuje estetski izgled aplikacije već i dodatno naglašava poruke povezivanja i izvrsnosti koje Udruga želi prenijeti.

A black background with white dots and lines

Description automatically generated

Slika 4.11 Pozadinska slika korisničkog sučelja

Konačno, font koji se koristi kroz cijelu aplikaciju je Poppins, moderan i čist sans-serif font koji doprinosi čitljivosti i vizualnoj privlačnosti. Poppins je odabran zbog svoje jednostavnosti i elegancije, savršeno se uklapajući u suvremeni dizajn aplikacije te omogućava dosljedan tipografski identitet kroz sve tekstualne elemente. Korištenje ovog fonta osigurava da tekst bude jasno prikazan, a istovremeno doprinosi ukupnom vizualnom identitetu Udruge, usklađenom s njenim stilskim vodičem.

## Korištenje Dockera za virtualizaciju modula sustava

Docker je alat za virtualizaciju aplikacija putem kontejnerizacija. Dok tradicionalna virtualizacija koristi hipervizore za kreiranje i upravljanje virtualnim mašinama koje uključuju cijele operativne sustave, Docker koristi kontejnerizaciju koja izolira samo aplikaciju i njene zavisnosti unutar istoga operativnog sustava u obliku kontejnera.

Pomoću Dockera, sustav je virtualiziran u četiri kontejnera koji se nalaze unutar zajedničkog mrežnog okruženja (engl. *Network*). Ova mreža omogućuje međusobnu komunikaciju svih kontejnera, a pokreće ih jedna glavna skripta smještena u *docker-compose.yaml* datoteci. U ovoj su datoteci konfigurirana četiri ključna kontejnera koja predstavljaju korisničko sučelje, poslužiteljski sloj, bazu podataka i Nginx proxy server. Konfiguracija docker-compose datoteke prikazana je u kôdu (Kôd 4.23)

version: "3.7"

services:

esustav\_ngix:

build: ./

container\_name: esustav\_nginx

restart: unless-stopped

ports:

- 8700:80

volumes:

- ./backend/static:/static:ro

depends\_on:

- esustav\_fe

- esustav\_db

esustav\_api:

build: ./backend

container\_name: esustav\_api

restart: unless-stopped

volumes:

- ./backend:/app/backend

depends\_on:

- esustav\_db

esustav\_fe:

restart: unless-stopped

container\_name: esustav\_fe

build:

context: frontend

dockerfile: Dockerfile

volumes:

- "./frontend:/app/backend"

esustav\_db:

container\_name: ${POSTGRES\_DB\_HOST}

image: postgres:latest

restart: unless-stopped

env\_file:

- .env

environment:

POSTGRES\_DB: ${POSTGRES\_DB}

POSTGRES\_USER: ${POSTGRES\_USER}

POSTGRES\_PASSWORD: ${POSTGRES\_PASSWORD}

PGDATA: /var/lib/postgresql/data

healthcheck:

test: ["CMD-SHELL", "sh -c 'pg\_isready -U ${POSTGRES\_USER} -d ${POSTGRES\_DB}'"]

interval: 15s

timeout: 3s

retries: 5

volumes:

- "./postgres-data:/var/lib/postgresql/data"

networks:

default:

Kôd 4.23 docker-compose.yaml

Na najnižem sloju aplikacije nalazi se baza podataka koja se prva pokreće prilikom inicijalizacije sustava. Njeni parametri, poput korisničkog imena, lozinke i naziva baze, povlače se iz .env datoteke kako bi se osigurala fleksibilnost i sigurnost prilikom postavljanja. Unutar kontejnera, koristi se Postgres Docker image, koji pokreće PostgreSQL bazu podataka. Da bi se osigurala pouzdanost, nad ovim kontejnerom je implementirana provjera "zdravlja" baze. Ova provjera vrši se pet puta u intervalima od petnaest sekundi, čime se osigurava da je baza spremna prije nego što ostali dijelovi sustava počnu koristiti njene resurse. Jedan od najvažnijih aspekata konfiguracije baze podataka je korištenje volumena. Volumeni definiraju gdje će se podaci iz kontejnera pohraniti na lokalnom sustavu. Ovo omogućuje trajnu pohranu podataka, što znači da se podaci neće izgubiti čak i ako se kontejner ponovno pokrene.

Poslužiteljski sloj (esustav\_api) predstavlja sustav gdje su implementirani svi ključni poslovni procesi. Ovaj kontejner se gradi iz *backend* direktorija, gdje se koristi Dockerfile za postavljanje Python 3.10 okruženja i instalacija potrebnih ovisnosti, poput migracija nad bazom i generiranje statičkih datoteka. Poslužiteljski sloj je odgovoran za rukovanje zahtjevima koje mu proslijedi Nginx i za komunikaciju bazom podataka. Konfiguriran je tako da automatski povlači promjene iz aplikacijskog kôda kroz volumene, čime se osigurava da sve promjene u kôdu budu odmah vidljive unutar kontejnera. Ovaj kontejner također ovisi o tome da baza podataka bude pokrenuta, što je definirano u *depends\_on* sekciji *docker-compose.yaml* datoteke.

Korisničko sučelje (esustav\_fe) je odgovorno za prikaz korisničkog sučelja aplikacije. Ovaj kontejner gradi se iz *frontend* direktorija, koristeći Dockerfile koji postavlja razvojno okruženje za React aplikaciju. Volumeni su konfigurirani tako da promjene u kôdu na lokalnom stroju budu automatski reflektirane unutar kontejnera, omogućujući brzu iteraciju i razvoj. Kontejner esustav\_fe ne ovisi o ostalim servisima da bi se pokrenuo, ali je važan za cjelokupnu funkcionalnost sustava jer omogućuje korisnicima interakciju s poslužiteljskim slojem aplikacije putem web-preglednika. Pristup ovom kontejneru je također upravljan kroz Nginx proxy server.

Nginx proxy server kontejner (esustav\_nginx) djeluje kao ulazna točka za sav vanjski promet prema aplikaciji. Ovaj kontejner gradi se pomoću prilagođene Nginx konfiguracije smještene u *nginx* direktoriju. Nakon što se pokrene, Nginx preusmjerava zahtjeve prema glavnom URL-u na korisničko sučelje aplikacije, dok zahtjeve prema */api* ruti preusmjerava na poslužiteljski sloj. Statički sadržaji, poput slika i CSS datoteka, poslužuju se izravno iz direktorija definiranog za te potrebe. Ova konfiguracija omogućuje optimizaciju prometa i povećava sigurnost, dok bi Nginx također mogao služiti za SSL/TLS terminaciju, čime bi se dodatno osigurala komunikacija putem HTTPS protokola. [7] Ovakva upotreba Nginxa ključna je za pouzdano i skalabilno funkcioniranje sustava. Unutar docker-compose.yaml datoteke, ovaj kontejner ovisi o tome da se prvo pokrenu korisničko sučelje i baza podataka, čime se osigurava da svi ključni servisi budu dostupni prije nego Nginx počne preusmjeravati promet. Također, ovaj kontejner mapira vanjski port 8700 na unutarnji port 80, omogućavajući pristup aplikaciji putem preglednika na lokalnom stroju.

Svaki od ovih kontejnera igra ključnu ulogu u cjelokupnom sustavu, od upravljanja bazom podataka do prikaza korisničkog sučelja. Korištenjem Dockera i *docker-compose.yaml* datoteke, omogućeno je jednostavno upravljanje svim komponentama, automatsko pokretanje potrebnih servisa i osiguravanje da su svi dijelovi sustava dostupni i funkcionalni. Na ovaj način ne ovisimo o operacijskom sustavu koji će pokretati sustav.

# Provođenje ankete kao analiza uspješnosti programskog rješenja

Da bi se procijenio stvarni utjecaj programskog rješenja na rad članova Udruge, provedeno je istraživanje putem ankete. Cilj je istraživanja bio ispitati razinu zadovoljstva korisnika unutar četiri ključna poglavlja: opće zadovoljstvo, intuitivnost sučelja, korisnost funkcionalnosti te brzina rada programskog rješenja. Anketa je koristila sustav ocjenjivanja od jedan do pet, pri čemu je ocjena jedan označavala nezadovoljstvo ili neispunjenost očekivanja, dok je ocjena pet označavala potpunu ispunjenost očekivanja.

Anketu je ispunilo 42 člana Udruge, od kojih je 12 članova Predsjedništva, 9 voditelja i 21 član tima. Rezultati su pokazali da je 57,1% ispitanika bilo upoznato s aplikacijama koje su se koristile prije implementacije novoga programskog rješenja, što ukazuje na to da je većina članova imala prethodno iskustvo s ovakvim sustavima (Slika 5.1). Ukupno zadovoljstvo članova programskim rješenjem ocijenjeno je vrlo visoko – čak 64,3% ispitanika dalo je ocjenu 5, dok je preostali dio ispitanika dao ocjenu 4. Ovo sugerira da je većina korisnika izrazito zadovoljna rješenjem.

A blue and red pie chart

Description automatically generated

Slika 5.1 Grafički prikaz odgovora na pitanje "Jesi li bio/la upoznat/a s aplikacijama koje su se koristile prije eSUSTAVa"

Ocjene o ispunjenim očekivanjima blago su varirale, pri čemu je 7,1% ispitanika dalo ocjenu 3, dok su svi ostali ispitanici dali ocjene 4 (42,9%) i 5 (50%). Ovi rezultati pokazuju da postoji određeni broj korisnika čija su očekivanja djelomično ispunjena, dok je većina korisnika izrazito zadovoljna postignutim rezultatima.

Na pitanja „Koliko su vam jasna i intuitivna sučelja eSUSTAVa?“ te „Koliko su vam dostupne funkcionalnosti korisne za obavljanje vaših zadataka?“ korisnici su odgovorili gotovo identično. Čak 78,6% ispitanika ocijenilo je sučelje i funkcionalnosti ocjenom 5, 14,3% dalo je ocjenu 4, dok je 7,1% ispitanika dalo ocjenu 3. Ovi rezultati pokazuju da većina korisnika smatra sučelje i funkcionalnosti intuitivnima i korisnima, no postoji i mali broj korisnika koji smatraju da postoje određena područja za poboljšanje.

Brzina rada sustava bila je aspekt s najnižim ocjenama. Ocjenu 3 dalo je 35,7% ispitanika, ocjenu 4 dalo je 28,7%, a 35,7% ispitanika ocijenilo je brzinu rada ocjenom 5. Kao ključne probleme, istaknuli su poteškoće s pristupom putem Google OAuth2 autentifikacije, sporije učitavanje podataka i trajanje JWT tokena. Iako su određeni problemi već bili primijećeni i djelomično popravljeni, ovo je jasno područje za daljnje unaprjeđenje.

Zaključno, istraživanje je pokazalo da su članovi Udruge općenito vrlo zadovoljni novim programskim rješenjem, posebno u pogledu intuitivnosti sučelja i funkcionalnosti. Međutim, brzina rada sustava prepoznata je kao glavni izazov koji zahtijeva daljnje poboljšanje. Rješenje je u velikoj mjeri omogućilo brži i efikasniji rad, a povratne informacije korisnika bit će ključne za daljnje optimizacije.

# Zaključak

Kako bi se osigurao visok stupanj organizacije podataka i ubrzali ključni operativni procesi unutar Udruge, donesena je strateška odluka o primjeni sofisticiranog centraliziranog sustava kao temelja svakodnevnog rada članova. Implementacija ovog sustava nije samo pojednostavila pristup informacijama već je drastično poboljšala učinkovitost i transparentnost cijeloga organizacijskog sustava.

Centralizacijom podataka i integracijom svih ključnih aplikacija u jedan kohezivan sustav, Udruga je ostvarila značajne uštede u vremenu i resursima. Na primjer, procesi kao što su provođenje izbora, upiti nadređenih u vezi s certifikatima te upravljanje SUPRACH-om sada su značajno brži i jednostavniji. Sustav omogućuje trenutačni pristup relevantnim podacima, smanjujući potrebu za ručnim unosom i provjerom informacija, što je rezultiralo značajnim smanjenjem administrativnog opterećenja članova.

Povećana je i sigurnost podataka, što je od kritične važnosti za očuvanje integriteta i povjerljivosti unutar organizacije. Centralizacijom je omogućena dosljedna organizacija podataka, a sigurnosne mjere, poput enkripcije i kontroliranog pristupa, dodatno su osnažene. Zahvaljujući ovakvom pristupu, Udruga je eliminirala potrebu za aktivnom upotrebom zastarjelih aplikacija i datoteka, čime je postignuta ušteda od 30% na poslužiteljskim resursima. Ovo je ne samo smanjilo troškove već i omogućilo brže performanse i bolju iskoristivost postojećih resursa.

Predloženo programsko rješenje razvijeno je u skladu s najboljim praksama u programiranju i arhitekturi sustava, što osigurava visoku modularnost, skalabilnost i jednostavnost održavanja. Sustav je dizajniran s tri glavna sloja: bazom podataka, poslužiteljskim slojem i klijentskim slojem, što omogućuje jasno razdvajanje odgovornosti i neovisnu implementaciju svakog sloja. Korištenje modernih tehnologija, poput Pythona i JavaScripta, omogućilo je brzu implementaciju kôda uz visoku kvalitetu i pouzdanost. Primjena naprednih paradigmi, poput REST-a, obrasca izdavač – pretplatnik te MVC arhitekture, osigurala je fleksibilnost sustava i njegovu jednostavnu prilagodbu budućim potrebama.

Uz sve navedeno, cijeli je sustav virtualiziran korištenjem Dockera, čime je postignuta visoka prenosivost i jednostavnost prilagodbe različitim radnim okruženjima. Ovaj aspekt dodatno osigurava dugoročnu održivost sustava, omogućujući lako skaliranje i upravljanje infrastrukturom, bez zastoja ili komplikacija.

Implementirano rješenje u potpunosti je usklađeno s pravilnicima Udruge i njezinim vizualnim identitetom, što znači da zadovoljava sve trenutne, ali i buduće potrebe u pogledu prikupljanja, organizacije i pregleda podataka. Ovaj je centralizirani sustav značajno unaprijedio operativne i organizacijske kapacitete Udruge, osiguravajući stabilnost, učinkovitost i održivost rada u dugoročnom smislu.

# Popis kratica

API *Application program interface* Aplikacijsko programsko sučelje

CLI Command Line Interface Sučelje naredbenog retka

CORS *Cross-origin resource sharing* Dijeljenje resursa s različitim izvorima

CRUD *Create, Read, Update, and Delete* Stvaranje, čitanje, ažuriranje i brisanje

CSRF *Cross-site request forgery* Krivotvorenje zahtjeva između stranica

DDOS *Denial-of-service attack* Napadi uskraćivanjem resursa

DOM Document Object Model Objektni model dokumenta

ER  *Entity relationship* Odnos entiteta

HTTP *Hypertext Transfer Protocol* Protokol za prijenos hiperteksta

JWT *JSON Web Token* JSON Web Token

MVC *Model-View-Controller* Model – pogled – kontroler

ORM *Object-Relational Mapping* Objektno-relacijsko preslikavanje

RDBMS Relational database management system Sustav upravljanja relacijskim bazama podataka

REST *Representational state transfer* Reprezentativni prijenos stanja

SSL  *Secure Sockets Layer* Sloj sigurnih utičnica

URI *Uniform resource identifier* Jedinstveni identifikator resursa

URL *Uniform resource locator* Jedinstveni lokator sadržaja

XML Extensible Markup Language Proširivi jezik za označavanje podataka

# Popis slika

[Slika 3.1 Arhitektura sustava 10](#_Toc174875992)

[Slika 3.2 Vizualni prikaz docker mreže 17](#_Toc174875993)

[Slika 3.3 Slikoviti prikaz NGINX funkcije 18](#_Toc174875994)

[Slika 3.4 Verzioniranje kôda 19](#_Toc174875995)

[Slika 4.1 ER dijagram modula *estudneti* 22](#_Toc174875996)

[Slika 4.2 ER dijagram modula *eaktivnosti* 23](#_Toc174875997)

[Slika 4.3 ER dijagram modula *eizbori* 24](#_Toc174875998)

[Slika 4.4 ER dijagram modula *suprach* 25](#_Toc174875999)

[Slika 4.5 ER dijagram modula *epartneri* 26](#_Toc174876000)

[Slika 4.6 Prikaz komunikacije različitih tehnologija s REST servisom 29](#_Toc174876001)

[Slika 4.7 Prikaz Swagger stranice 32](#_Toc174876002)

[Slika 4.8 Direktoriji i njihova struktura unutar poslužiteljskog sloja aplikacije 34](#_Toc174876003)

[Slika 4.9 Početna stranica sustava 49](#_Toc174876004)

[Slika 4.10 Paleta eSTUDENT boja 55](#_Toc174876005)

[Slika 4.11 Pozadinska slika korisničkog sučelja 55](#_Toc174876006)

[Slika 5.1 Grafički prikaz odgovora na pitanje "Jesi li bio/la upoznat/a s aplikacijama koje su se koristile prije eSUSTAVa?" 60](#_Toc174876007)

# Popis tablica

[Tablica 4.1 Funkcije koje predstavljaju CRUD operacije unutar pogleda 38](#_Toc174875990)

[Tablica 4.2 Popis dozvola korisnika unutar korisničkog sučelja 48](#_Toc174875991)

# Popis kôdova

[Kôd 4.1 Dekodiranje Googleovog kôda i autorizacija korisnika 28](#_Toc174875966)

[Kôd 4.2 Specificirani izvori koji imaju pristup CORS-a 30](#_Toc174875967)

[Kôd 4.3 Specificirani izvori koji imaju pristup CSRF-a 31](#_Toc174875968)

[Kôd 4.4 Postavke baze podataka na poslužiteljskom sloju 31](#_Toc174875969)

[Kôd 4.5 Swagger postavke 32](#_Toc174875970)

[Kôd 4.6 INSTALLED\_APPS konfiguracija 33](#_Toc174875971)

[Kôd 4.7 ORM klasa AcademicYear 35](#_Toc174875972)

[Kôd 4.8 AcademicYear serializator 36](#_Toc174875973)

[Kôd 4.9 Serijalizator s povezanim serijalizatorom na parametru role\_group 36](#_Toc174875974)

[Kôd 4.10 ModelViewSet pogled 38](#_Toc174875975)

[Kôd 4.11 Proširivanje pogleda 39](#_Toc174875976)

[Kôd 4.12 PartnersView pristupna točka 40](#_Toc174875977)

[Kôd 4.13 Povezivanje pogleda i pristupnih točaka 41](#_Toc174875978)

[Kôd 4.14 Povezivanje modula s glavnim prefiksom 42](#_Toc174875979)

[Kôd 4.15 Django administratorsko sučelje 43](#_Toc174875980)

[Kôd 4.16 Registracija UserPermissions modela i optimizacija prikaza 44](#_Toc174875981)

[Kôd 4.17 Dozvole unutar modula eAKTIVNOSTI 50](#_Toc174875982)

[Kôd 4.18 Dozvole unutar tablice aktivnosti 50](#_Toc174875983)

[Kôd 4.19 Tablica aktivnosti unutar modula eAKTIVNOSTI 51](#_Toc174875984)

[Kôd 4.20 Konfiguracija Axios Contexta 52](#_Toc174875985)

[Kôd 4.21 Primjer hooka za dohvaćanje podataka 53](#_Toc174875986)

[Kôd 4.22 Primjer hooka za slanje podataka 54](#_Toc174875987)

[Kôd 4.23 docker-compose.yaml 57](#_Toc174875988)

# Literatura

|  |  |
| --- | --- |
| [1] | »Salesforce,« [Mrežno]. Available: https://www.salesforce.com/nonprofit/. |
| [2] | »Blackbaud,« [Mrežno]. Available: https://www.blackbaud.com. |
| [3] | »WildApricot,« [Mrežno]. Available: https://www.wildapricot.com. |
| [4] | »About PostgreSQL,« [Mrežno]. Available: https://www.postgresql.org/about/. |
| [5] | K. Mangabo, Full Stack Django and React: Get hands-on experience in full-stack web development with Python, React, and AWS, Packt Publishing, 2023. |
| [6] | J. N. a. S. Kuenzli, Docker in Action, Second Edition, Manning Publications, 2019. |
| [7] | »NGINX Product Documentation,« F5, [Mrežno]. Available: https://docs.nginx.com. [Pokušaj pristupa 8 April 2024]. |
| [8] | M. F. a. C. Nedelcu, Nginx HTTP Server - Fourth Edition, Packt Publishing, 2018. |
| [9] | GIT, »Git Documentations,« [Mrežno]. Available: https://git-scm.com/doc. [Pokušaj pristupa 5 August 2024]. |
| [10] | »Using OAuth 2.0 to Access Google APIs,« Google, 1 January 2024. [Mrežno]. Available: https://developers.google.com/identity/protocols/oauth2. [Pokušaj pristupa 8 August 2024]. |
| [11] | »Introduction to OAuth 2.0,« Google, 2 June 2024. [Mrežno]. Available: https://cloud.google.com/apigee/docs/api-platform/security/oauth/oauth-introduction. [Pokušaj pristupa 8 August 2024]. |
| [12] | R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, 2008. |
| [13] | W. S. Vincen, Django for APIs: Build web APIs with Python & Django, Independently published, 2020. |
| [14] | https://www.geeksforgeeks.org/business-logic-layer/, »Geeks for Geeks (Business-Logic Layer),« 24 Jan 2023. [Mrežno]. Available: https://www.geeksforgeeks.org/business-logic-layer/. [Pokušaj pristupa 11 Aug 2024]. |

1. https://happycoding.io/tutorials/java-server/images/rest-api-1.png [↑](#footnote-ref-1)