Univerzitet u Novom Sadu,
Fakultet tehničkih nauka

SEMINARSKI RAD

Nastavni predmet: Tehnologije i sistemi eUprave
Naziv teme: Univerzitet

Student,
Janko Rakonjac SR33/2021

Mentor,
Profesor Marko Markovic

Jun, 2024

1. Sažetak

U ovom radu opisan je servis za fakultet koji je deo univerzitetske službe. Sistem pruža korisnicima mogućnosti vezane za usluge fakulteta, a drugim sistemima omogućava pristup informacijama o korisnicima neophodnim za njihovu funkcionalnost. Za izradu projekta korišćeni su UML dijagrami klasa i slučajeva korišćenja, programski jezik Go za backend, React za frontend, te Docker za kontejnerizaciju servisa. Sistem koristi mikroservisnu arhitekturu i single sign-on (SSO) za autentifikaciju i autorizaciju korisnika. Korisnici, u ulozi studenata, mogu pregledavati svoje podatke, podnositi zahteve za produženje statusa studenta, tražiti potvrdu o studiranju, podnositi zahteve za polaganje državnog ispita, te podnositi zahtev za stipendiju. Korisnici, u ulozi profesora, mogu podneti zahtev za produženje statusa profesora.

2. Ključne reči

- fakultet
- studentska služba
- veb
- mikroservisna arhitektura
- eUprava
- go
- docker
- sso
- react
- uml

3. Uvod

U današnjem digitalnom dobu, univerziteti se sve više oslanjaju na sofisticirane informacione sisteme kako bi unapredili efikasnost i korisničko iskustvo svojih studenata i osoblja. Jedan od ključnih segmenata ovog napretka jeste implementacija servisa za **fakultet** koji integriše različite funkcionalnosti unutar univerzitetskog okruženja. Ovaj seminarski rad ima za cilj da detaljno opiše razvoj takvog sistema, fokusirajući se na **arhitekturu baziranu na mikroservisima**, koristeći napredne tehnologije poput programskog jezika **Go** za backend i **React** za frontend aplikacije, te da prikaže korisćenje jednog ovakvog servisa u funkciji **studentske službe** i samog **fakulteta**. Dodatno, implementacija **Docker** kontejnerizacije osigurava skalabilnost i pouzdanost sistema. Kroz primenu **UML** dijagrama klasa i slučajeva korišćenja, sistem pruža studentima mogućnost da lako pristupe svojim podacima, podnesu zahteve za administrativne procedure kao što su produženje statusa studenta ili zahtev za dobijanje potvrda. Istovremeno, profesorski kadar ima specifične opcije za administraciju, uključujući zahteve za produženje statusa profesora. Implementacija jedinstvenog prijavljivanja (**SSO**) unapređuje sigurnost sistema, osiguravajući da korisnici imaju siguran i pouzdan pristup svojim funkcionalnostima. Ovaj rad predstavlja korak napred ka modernizaciji univerzitetskog upravljanja, pružajući stabilan i intuitivan servis za sve uključene u univerzitetski proces.

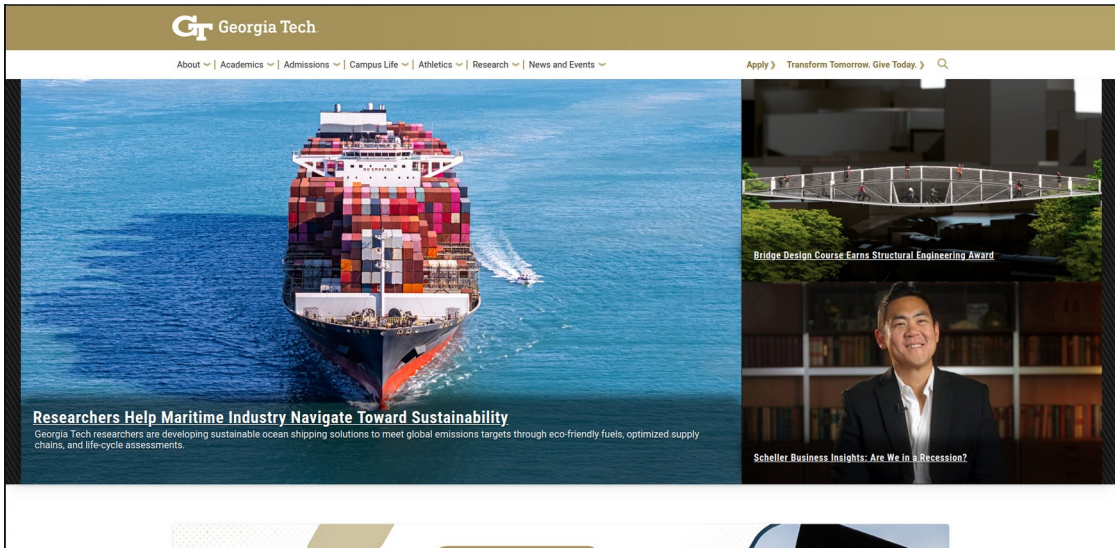
4. Srodna istraživanja

Jedan od primera visokoškolske institucije koja koristi sličan integrirani informacioni sistem za upravljanje studentima i administrativnim procesima je **Georgia Institute of Technology (Georgia Tech) u Sjedinjenim Američkim Državama.**

Georgia Tech je implementirao sistem pod nazivom "**OneUSG Connect**", koji integriše različite funkcionalnosti kao što su ljudski resursi, finansijske usluge i upravljanje studentskim informacijama u jedan centralizovani sistem. Ovaj sistem koristi modernu tehnologiju kao što su mikroservisi za različite komponente sistema, **SSO** (Single Sign-On) za jednostavan pristup korisnicima, kao i napredne tehnike za analizu podataka kako bi podržao donošenje odluka i upravljanje performansama.

Implementacija ovog sistema omogućava studentima, fakultetu i administrativnom osoblju jednostavan pristup relevantnim informacijama i uslugama, čime se unapređuje efikasnost i korisničko iskustvo unutar univerzitetskog okruženja.

Ovaj primer pokazuje kako visokoškolske institucije mogu iskoristiti moderne IT tehnologije i pristupe za optimizaciju operacija i pružanje boljeg servisa svojim članovima zajednice.



Slika 1 - početna stranica Georgia Institute of Technology

5. Korišćene tehnologije

5.1 Go programski jezik

Go je programski jezik otvorenog koda nastao u Guglu 2007. godine. Osmišljen je od strane Roberta Grisimera, Roba Pajka i Ken Tomposn-a. Jezik je kompajliran, sa statičkim tipovima podataka, strukturnim tipovima i sadrži automatsko upravljanje memorijom. Ovaj jezik je prvenstveno bio namenjen sistemskom programiranju. Početak rada na jeziku je septembar 2007. godine, a zvanično je promovisan 2009. godine.

Postoje dve glavne implementacije, gc koja je glavna implementacija koju je razvio Gugl,gcgcgo iz GNU kolekcije kompajlera. Najveći uticaj na ovaj jezik su jezici iz porodice C I Paskal. Danas, ovaj jezik je široko prihvaćen u industriji zbog svoje jednostavnosti, efikasnosti I podrške za konkurentno programiranje. Koristi se za web aplikacije, mrežne alate, distribuirane sisteme i mikroservise. Neke od poznatih kompanija koje ga koriste su Dropbox, Netflix, Uber, Docker i mnoge druge. Često je izbor u DevOps okruženjima, zbog dobre podloge za razvoj infrastukture i efikasno upravljanje Cloud-om.

Slika 2 - gobhere koij je simbol Go programskog jezika

5.2 React

React je popularna JavaScript biblioteka za izgradnju korisničkih interfejsa, koju je razvio Facebook. Ovaj alat se koristi za kreiranje dinamičnih i interaktivnih korisničkih interfejsa (UI) u web aplikacijama. Evo nekoliko ključnih karakteristika i prednosti React tehnologije:

1. **Komponentna arhitektura**: React se zasniva na komponentnoj arhitekturi, što znači da aplikaciju možete graditi pomoću malih, ponovno upotrebljivih komponenti. Svaka komponenta predstavlja deo korisničkog interfejsa koji može da sadrži HTML kod, CSS stilove i JavaScript logiku.
2. **Virtualni DOM**: React koristi virtualni DOM (Document Object Model) za efikasno ažuriranje prikaza aplikacije. Umesto direktnog manipulisanja DOM-om, React prvo ažurira virtualni DOM, a zatim vrši poređenje sa stvarnim DOM-om i primenjuje samo neophodne promene, što dovodi do efikasnijeg renderovanja i boljeg performansa aplikacije.
3. **JSX sintaksa**: JSX je proširenje JavaScript sintakse koje omogućava pisanje HTML-a direktno unutar JavaScript koda. Ova kombinacija HTML-a i JavaScripta olakšava kreiranje i održavanje korisničkih interfejsa u React aplikacijama.
4. **Jednosmerna data veza**: React promoviše jednosmernu vezu podataka (one-way data binding), što znači da podaci teku jednim smerom od roditeljskih komponenti ka dečijim komponentama. Ovo olakšava upravljanje stanjem aplikacije i smanjuje mogućnost grešaka.
5. **Velika zajednica i ekosistem**: React ima veliku i aktivnu zajednicu programera, što rezultira obiljem dostupnih biblioteka, alata i resursa za učenje. Postoji mnogo dodataka i rešenja treće strane koji mogu proširiti funkcionalnosti React aplikacija.
6. **Raznovrsne primene**: React se može koristiti za izradu različitih vrsta aplikacija, uključujući jednostranične aplikacije (SPA), mobilne aplikacije putem React Native-a, kao i za rendering na serverskoj strani putem Node.js-a.

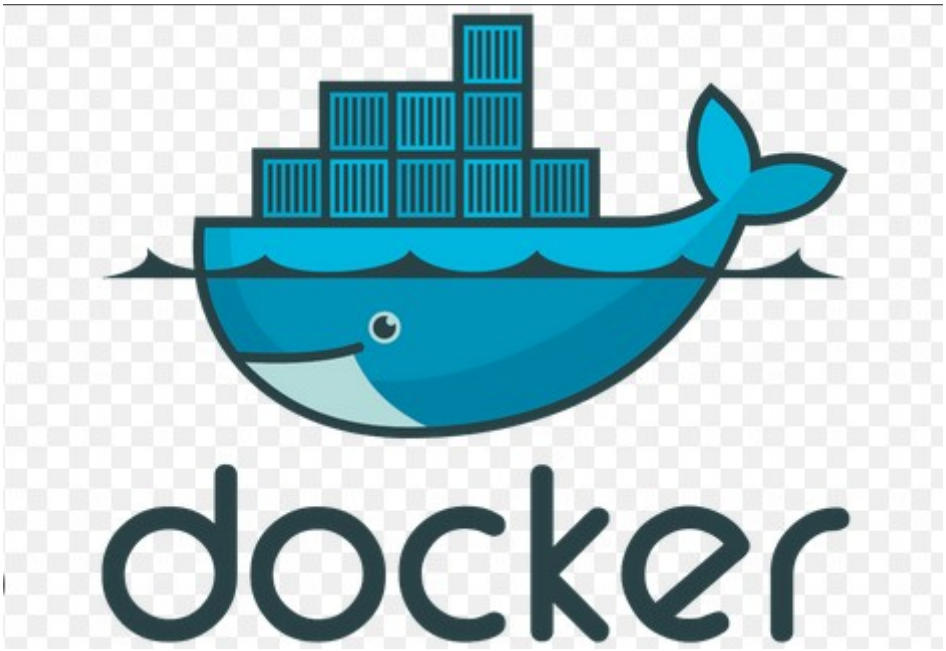
Zahvaljujući ovim karakteristikama, React je postao popularan izbor za razvoj modernih web aplikacija koje zahtevaju dinamičnost, skalabilnost i dobro korisničko iskustvo.

5.3 Docker

Docker je platforma koja omogućava programerima da razvijaju, pakiraju i pokreću aplikacije kao kontejnere, što olakšava njihovu brzu isporuku i pokretanje na različitim okruženjima, bez obzira na razlike u konfiguraciji i postavkama sistema. Evo ključnih karakteristika i prednosti Docker tehnologije:

1. **Kontejnerizacija**: Docker kontejneri su izolirana okruženja koja sadrže aplikaciju i sve njene zavisnosti (biblioteke, alate, konfiguraciju). Kontejneri omogućavaju da aplikacija bude pokrenuta u odredjenom okruženju bez obzira na host operativni sistem.
2. **Efikasnost i lakoća korišćenja**: Docker olakšava pakiranje aplikacija u kontejnere jednostavnom definicijom Dockerfile-a, koji opisuje korake za izgradnju slike aplikacije. Ove slike mogu se zatim deliti putem Docker Hub-a ili drugih registara.
3. **Skalabilnost**: Kontejneri omogućavaju horizontalno skaliranje aplikacija, tj. dodavanje ili uklanjanje instanci aplikacija prema potrebama, bez potrebe za kompleksnim podešavanjima.
4. **Konzistentnost okruženja**: Docker garantuje da će aplikacija raditi identično na svim okruženjima koja podržavaju Docker, od razvoja do produkcije, što smanjuje broj problema povezanih sa konfiguracijom okruženja.
5. **Sigurnost**: Docker kontejneri koriste osnovni operativni sistem za izvršavanje aplikacija, ali su izolovani jedan od drugog i od host sistema, čime se smanjuje rizik od uticaja zlonamernog softvera ili grešaka u aplikaciji na ostatak sistema.
6. **Zajednica i podrška**: Docker ima široku zajednicu programera i aktivnu podršku, što olakšava pronalaženje rešenja za probleme i pristupanje raznovrsnim alatima i resursima.
7. **Primena u mikroservisnim arhitekturama**: Docker je ključna tehnologija u mikroservisnim arhitekturama, gde omogućava nezavisno pakiranje i izvršavanje svake mikroservisne komponente kao kontejnera, što pojednostavljuje upravljanje kompleksnim aplikacijama sa više komponenti.

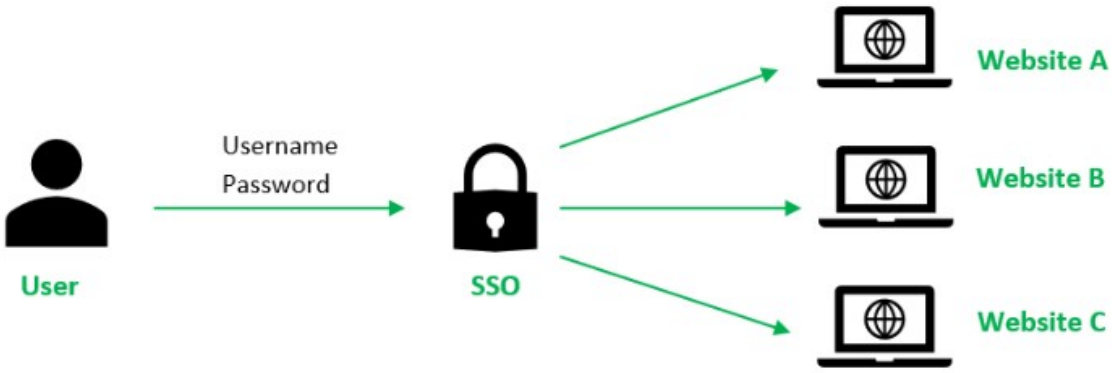
Docker je postao standardni alat za razvoj i upravljanje aplikacijama u modernim softverskim razvojnim timovima, obezbeđujući bržu isporuku, veću pouzdanost i bolju skalabilnost aplikacija.



Slika 3 – Docker logo

5.4 Single Sign-On (SSO)

Single sign-on je metoda za autentifikaciju korisnika i sesija, koja omogućava korisniku da koristi određene kredencijale, na primer korisničko ime i lozinku, kako bi mogao da pristupi skupu aplikacija. Ovim se smanjuje potrebu za korišćenjem više lozinki. Šematski prikaz nalazi se na slici 4.



Slika 4 – SSO

6. Specifikacija zahteva

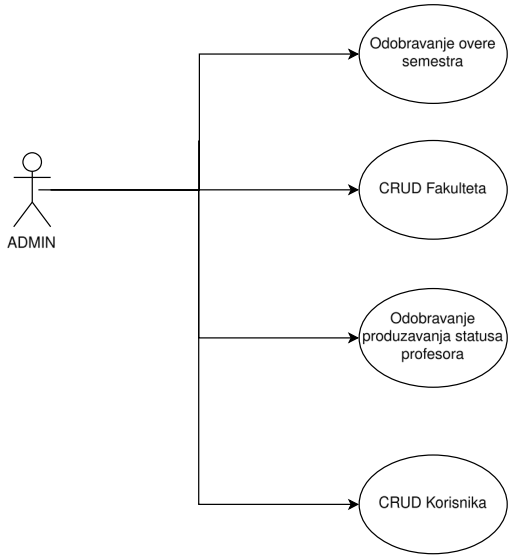
U ovom poglavlju objašnjeni su funkcionalni i nefunkcionalni zahtevi koje ima servis za fakultet koji je deo sistema univerzitetske sluzbe. Funkcionalni zahtevi prikazani su UML use case dijagramom (slike 5, 6).

Admin tok:

Admin se loguje na sistem kao admin, sa predefinisanim kredencijalima.
Ima opcija pristupa CRUD funkcionalnostima vezanim za fakultet i korisnika.

Odobravanje overe semestra:
Adminu stize notifikacija da je odredjeni student podneo zahtev za overu semestra,nakon cega on taj zahtev odobrava ili odbija.

Odobravanje produzavanja statusa profesora:
Adminu stize notifikacija da je odredjeni profesor podneo zahtev za produzavanje statusa profesora,nakon cega on taj zahtev odobrava ili odbija.



Gradjanin tok:

Gradjanin se loguje na sistem sa predefinisanim kredencijalima.

Podnosenje zahteva za polaganje prijemnog ispita:
Gradjanin podnosi zahtev za prijemni za odredjeni fakultet, nakon cega dobija notifikaciju sa mestom i vremenom odrzavanja ispita.

Biranje uloge:
Nakon logovanja na sistem gradjanin ima opciju da izabere da li se loguje kao Student ili Profesor.

slika 5

Profesor tok:

Nakon biranja uloge u sistemu, gradjanin je ulogovan u svojstvu profesora.

Podsnosenje zahteva za produzavanje statusa profesora:
Profesor podnosi zahtev za produzavanje statusa, uz koji mora da prilozi i potvrdu o nekaznjavanju.

Student tok:

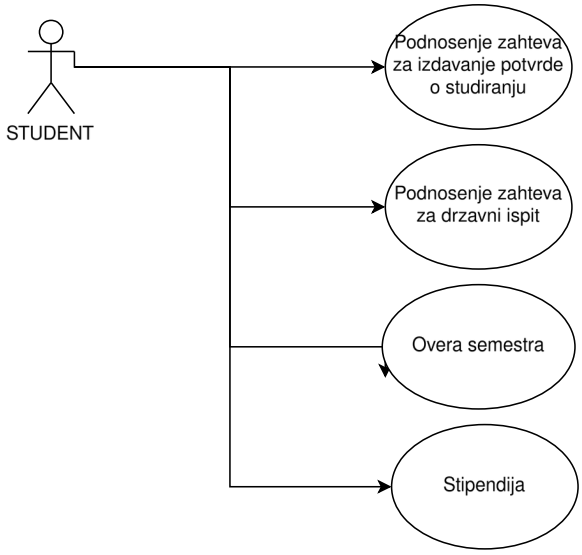
Nakon biranja uloge u sistemu, gradjanin je ulogovan u svojstvu studenta.

Podnosenje zahteva za izdavanje potvrde o studiranju:
Student podnosi zahtev za izdavanje potvrde o studiranju koja se izdaje u vidu pdf fajla, pod uslovom da je student aktivan.

Podnosenje zahteva za drzavni ispit:
Student podnosi zahtev za polaganje drzavnog ispita

Overa semestra:
Student podnosi zahtev za overu semestra, nakon cega admin usvaja zahtev i tako produzuje aktivnost studenta.

Stipendija:
Student podnosi zahtev za dobijanje stipendije, nakon automatskog rangiranja dobija obavestenje u vidu notifikacije da li je dobio stipendiju ili ne.



slika 6

6.1 Specifikacija funkcionalnih zahteva

Slučajevi korišćenja
Tabela 1 prikazuje opis slučaja korišćenja Podnosenje zahteva za prijemni ispit

Naziv	Podnosenje zahteva za prijemni ispit
Ucesnici	Gradjanin
Preduslov	Gradjanin je pristupio sistemu
Koraci	1. Gradjanin pristupio sistemu 2. Gradjanin podnosi zahtev za polaganje prijemnog ispita
Rezultat	Gradjanin dobija vreme I mesto polaganja ispita
Izuzeci	/

. *tabela 1 – Opis slucaja koriscenja Podnosenja zahteva za prijemni ispit*

Tabela 2 prikazuje opis slucaja koriscenja Podnosenje zahteva za produzavanje statusa profesora.

Naziv	Podnosenje zahteva za produzavanje
Preduslovi	1.Profesor postoji u sistemu 2. Profesor pristupa sistemu 3. Profesor ima potvrdu o nekaznjavanju
Ucesnici	Profesor
Rezultat	Status profesora se produzava
Koraci	1. Profesor pristupa sistemu 2. Profesor ima potvrdu o nekaznjavanju 3. Profesor podnosi zahtev
Izuzeci	Nepostojeca potvrda o nekaznjavanju

Tabela 2 - opis slucaja koriscenja Podnosenje zahteva za produzavanje statusa profesora.

Tabela 3 prikazuje opis slucajeva koriscenja Podnosenje zahteva za izdavanje potvrde o studiranju.

Naziv	Podnosenje zahteva za izdavanje potvrde o studiranju
Preduslovi	Student postoji u sistemu Student je pristupio sistemu
Ucesnici	Student
Rezultat	PDF potvrda o studiranju
Koraci	1.Student je pristupio sistemu 2.Student podnosi zahtev za izdavanje potvrde o studiranju
Izuzeci	/

Tabela 3 - opis slucajeva koriscenja Podnosenje zahteva za izdavanje potvrde o studiranju

Tabela 4 prikazuje opis slucajeva koriscenja Podnosenje zahteva za polaganje drzavnog ispita

Naziv	Podnosenje zahteva za polaganje drzavnog ispita
Preduslovi	Student postoji u sistemu Student je pristupio sistemu
Ucesnici	Student
Rezultat	Student dobija vreme I mesto polaganja ispita
Koraci	1.Student je pristupio sistemu 2.Student podnosi zahtev za polaganje drzavnog ispita
Izuzeci	Student nije ispunio potrebne uslove za polaganje drzavnog ispita

Tabela 4 - opis slucajeva koriscenja Podnosenje zahteva za polaganje drzavnog ispita

Tabela 5 prikazuje opis slucajeva koriscenja Overa semestra

Naziv	Overa semestra
Preduslovi	Student postoji u sistemu Student je pristupio sistemu
Ucesnici	Student
Rezultat	Studentu se produzava status za jos jedan semestar
Koraci	1.Student je pristupio sistemu 2.Student podnosi zahtev za overu semestra
Izuzeci	Student nije uradio sistematski pregled

Tabela 5 - opis slucajeva koriscenja Overa semestra

Tabela 6 prikazuje opis slucajeva koriscenja Stipendija

Naziv	Stipendija
Preduslovi	Student postoji u sistemu Student je pristupio sistemu
Ucesnici	Student
Rezultat	Student podnosi zahtev za stipendiju
Koraci	1.Student je pristupio sistemu 2.Student podnosi zahtev za stipendiju
Izuzeci	Student ne ispunjava uslov za dobijanje stipendija

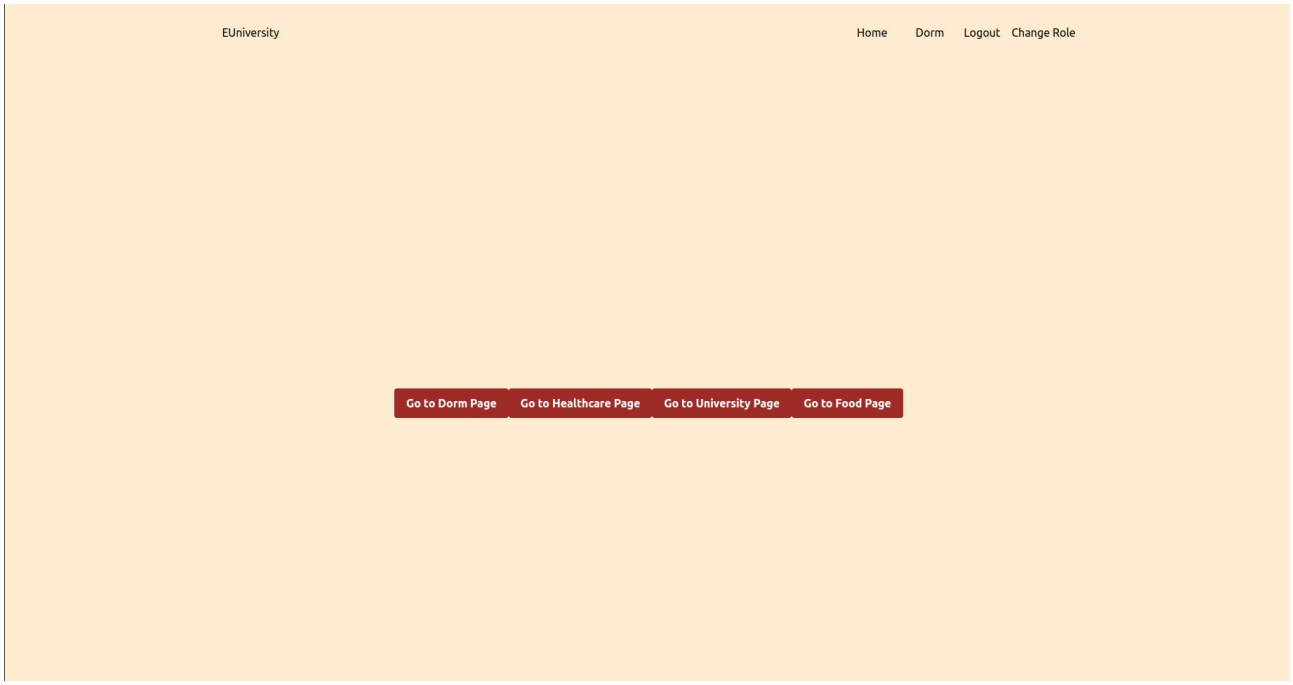
Tabela 6 - opis slucajeva koriscenja Stipendija

6.2 Specifikacija nefunkcionalnih zahteva

Razmatranje specifikacija nefunkcionalnih zahteva, kao što su otpornost sistema na parcijalne otkaze i korisnički prijateljski korisnički interfejs (UI), ključno je za osiguravanje kvaliteta i korisničkog iskustva u razvoju softverskih sistema. Evo detaljnijeg objašnjenja za oba zahteva:

1. **Otpornost sistema na parcijalne otkaze**: Ovaj zahtev se odnosi na sposobnost sistema da nastavi sa radom bez prekida i gubitka funkcionalnosti uprkos delimičnom otkazu komponenti ili podsistema.
2. **UI (Korisnički interfejs)**: Ovaj zahtev se odnosi na dizajn i funkcionalnosti korisničkog interfejsa koji olakšavaju korisnicima interakciju sa sistemom na intuitivan i efikasan način. Razvoj UI-a koji je jednostavan za navigaciju, jasno organizovan, sa dobro označenim akcijama i brzim odgovorom na korisničke akcije. Korišćenje React-a za razvoj može omogućiti dinamičan UI koji se brzo ažurira bez potrebe za osvežavanjem stranice

Dodatno, ova dva zahteva mogu biti detaljno specifikovana kroz scenarije koriscenja performanse sistema i kroz metrike kvaliteta kako bi se osiguralo da sistem zadovoljava očekivanja u pogledu pouzdanosti i korisničkog iskustva. Integracija ova dva aspekta u proces razvoja softvera može značajno doprineti uspehu i prihvaćenosti sistema od strane korisnika.



Slika 7 – Pocetna strana sistema

7 Specifikacija dizajna

Aplikacija je implementirana po mikroservisnoj arhitekturi - svaki podsistem **eUprave** je zaseban servis koji ima komunikaciju sa barem jednim drugim servisom. Pored fakulteta, koji je fokus ovog seminarskog rada, u ovom sistemu imamo i aplikaciju za dom, zdravstvo i ishranu, kao i servis za autorizaciju. Fakultet komunicira sa aplikacijom za zdravstvo da bi dobio informaciju da li je student uradio sistematski pregled.

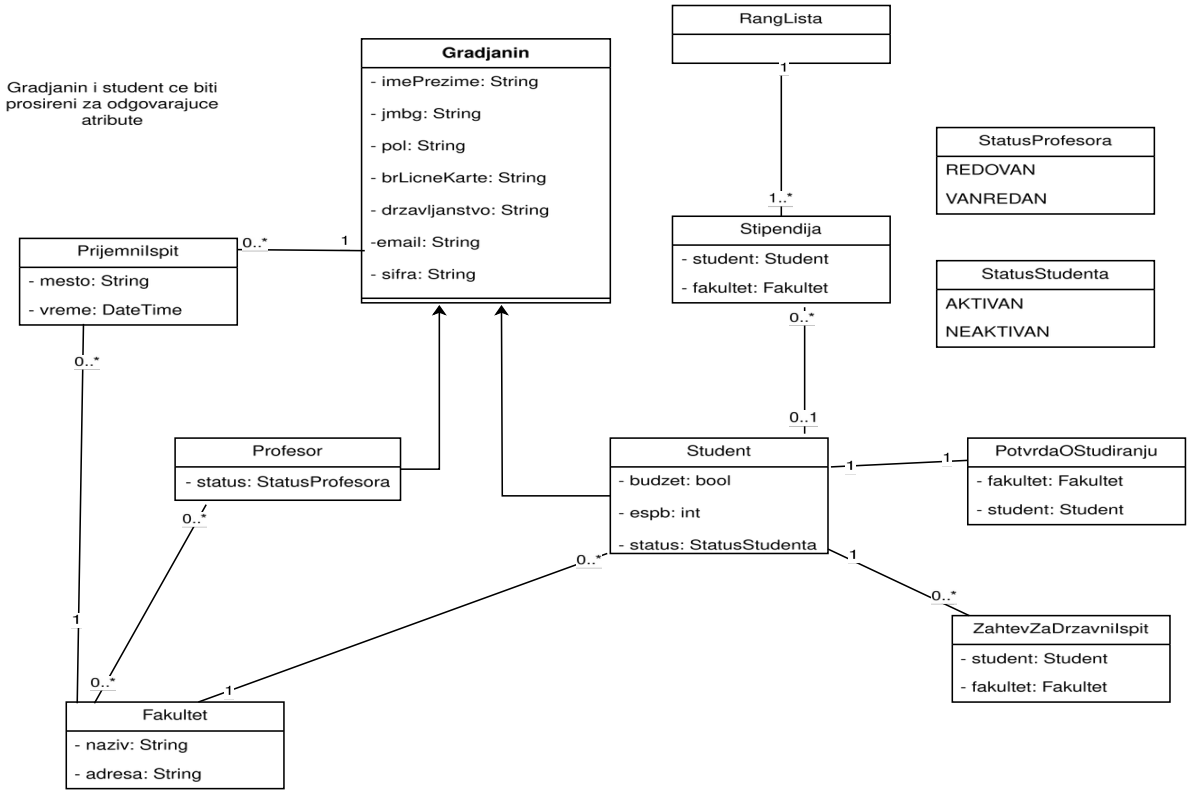
Komponente sistema unutar aplikacije za Fakultet:

1. **Korisnički interfejs (Frontend)** – React aplikacija, preko koje građani, admini, studenti i profesori pristupaju sistemu kroz veb pregledač.

2. **Servisni sloj (Backend) –** servisni sloj implementiran u Go programskom jeziku, koja u sebi sadrzi sledecu arhitekturu: model podataka, servisni sloj, repo sloj, hendler sloj, rukovanje http zahtevima, custom error handling, pomocne funkcije.

3. **Baza podataka** – MongoDB NoSQL baza podataka, gde se cuvaju informacije o fakultetu,studentima, profesorima.

Model podataka prikazan je na sledecem dijagramu:



Slika 8 – klasni dijagram

8. Implementacija sistema

U ovom poglavlju će biti opisane neke funkcionalnosti sistema koje su navedene unutar use case i klasnog dijagrama. Komunikacija teče od handlera do servisa koji se spušta na repo sloj. Svaka funkcija ima svoj API koji je otvoren i obezbeđen u skladu sa zahtevima.

1. **Overa semestra**

```
func (u UniversityService) ExtendStatus(personalIdentificationNumber string) (*models.Student, *errors.ErrorStruct) {  1 usage  janko33
    student, err := u.UniversityRepository.FindStudentById(personalIdentificationNumber)
    fmt.Println(student)
    if err != nil { return nil, err }

    healthStatusConfirmed, err := u.HealthCareClient.GetUserHealthStatusConfirmation(personalIdentificationNumber)
    if err != nil { return nil, err }

    if !healthStatusConfirmed { return nil, errors.NewError( message: "Health status confirmation failed",  status: 400) }
    student.Semester += 1
    updatedStudent, err := u.UniversityRepository.UpdateStudent(*student)
    if err != nil { return nil, err }

    return updatedStudent, nil
}
```

slika 9 – overa semestra

2. Podnosenje zahteva za prijemni ispit

```
func (u UniversityService) ExtendStatus(personalIdentificationNumber string) (*models.Student, *errors.ErrorStruct) {  1 usage  janko33
    student, err := u.UniversityRepository.FindStudentById(personalIdentificationNumber)
    fmt.Println(student)
    if err != nil { return nil, err }

    healthStatusConfirmed, err := u.HealthCareClient.GetUserHealthStatusConfirmation(personalIdentificationNumber)
    if err != nil { return nil, err }

    if !healthStatusConfirmed { return nil, errors.NewError( message: "Health status confirmation failed",  status: 400) }
    student.Semester += 1
    updatedStudent, err := u.UniversityRepository.UpdateStudent(*student)
    if err != nil { return nil, err }

    return updatedStudent, nil
}
```

slika 10 - Podnosenje zahteva za prijemni ispit

3. Podnosenje zahteva za drzavni ispit

```go
func (u UniversityService) ExtendStatus(personalIdentificationNumber string) (*models.Student, *errors.ErrorStruct) {  1 usage  janko33
    student, err := u.UniversityRepository.FindStudentById(personalIdentificationNumber)
    fmt.Println(student)
    if err != nil { return nil, err }

    healthStatusConfirmed, err := u.HealthCareClient.GetUserHealthStatusConfirmation(personalIdentificationNumber)
    if err != nil { return nil, err }

    if !healthStatusConfirmed { return nil, errors.NewError( message: "Health status confirmation failed",  status: 400) }
    student.Semester += 1
    updatedStudent, err := u.UniversityRepository.UpdateStudent(*student)
    if err != nil { return nil, err }

    return updatedStudent, nil
}
```

slika 11 – Podnosenje zahteva za drzavni ispit

4. Podnosenje zahteva za stipendiju

```go
func (u UniversityService) ExtendStatus(personalIdentificationNumber string) (*models.Student, *errors.ErrorStruct) {  1 usage  janko33
    student, err := u.UniversityRepository.FindStudentById(personalIdentificationNumber)
    fmt.Println(student)
    if err != nil { return nil, err }

    healthStatusConfirmed, err := u.HealthCareClient.GetUserHealthStatusConfirmation(personalIdentificationNumber)
    if err != nil { return nil, err }

    if !healthStatusConfirmed { return nil, errors.NewError( message: "Health status confirmation failed",  status: 400) }
    student.Semester += 1
    updatedStudent, err := u.UniversityRepository.UpdateStudent(*student)
    if err != nil { return nil, err }

    return updatedStudent, nil
}
```

slika 12 – Podnosenje zahteva za stipendiju

10. Zaključak

Zaključno, ovaj seminarski rad je detaljno opisao implementaciju servisa za fakultet unutar šireg eUprave sistema, koristeći mikroservisnu arhitekturu i moderne tehnologije poput React-a za frontend i Docker-a za kontejnerizaciju. Analizirane su funkcionalnosti sistema prema use case i klasnim dijagramima, naglašavajući važnost komunikacije između handlera i servisa koji interaguju sa repozitorijum slojem. Svaka funkcija je pristupna putem otvorenog i sigurnog API-ja, što doprinosi pouzdanosti i skalabilnosti sistema. Implementacija integracije sa drugim podsistemima eUprave, poput zdravstvenog sistema, dodatno demonstrira fleksibilnost i praktičnost ovakvog pristupa. Ovaj rad predstavlja korak napred u modernizaciji administrativnih procesa univerziteta, pružajući stabilan i efikasan servis za sve uključene aktere, od studenata do profesora.