Univerzitet u Novom Sadu,
Fakultet tehničkih nauka

SEMINARSKI RAD

Nastavni predmet: Tehnologije i sistemi eUprave
Naziv teme: Zdravstvo

Student,
Boris Lauš SR11/2021

Jun, 2024

1.Sažetak

U ovom radu je opisan sistem zdravsta, u sklopu fakultetske službe. Sistem pruža korisnicima mogućnosti vezane za usluge u zdravstvu, a ostalim sistemima informacije o datom korisniku koje su potrebne za njihovu funkcionalnost. Za izradu projekta su korišćeni UML dijagrami klasa i slučajeva korišćenja, programski jezik Go[1] za backend, React[2] za frontend, Docker[3] za dokerizaciju servisa. Sistem prati mikroservisnu arhitekturu, i koristi single sign-on (SSO) za autentifikaciju i autorizaciju korisnika. Korisnici, u ulozi pacijenta imaju mogućnost uvida u svoj karton i generisanje dokumenata, dok u ulozi lekara imaju i mogućnost izmene pomenutog sadržaja.

2. Ključne reči

- zdravstvo
- studentska služba
- veb alpikacija
- mikroservisna arhitektura

3. Uvod

Primena modernih tehnologija u zdravstvenoj službi danas predstavlja jedan od ključnih faktora za unapređenje efikasnosti i kvaliteta pružene zdravstvene nege. Zbog rasta broja korisnika, dolazi do potrebe za skladištenjem i obradom velike količine podataka. Zbog osetljive prirode ovih podataka, mora postojati i nivo kontrole koji garantuje da se njima rukuje na odgovarajući način, i da je pristup istim dozvoljen isključivo ovlašćenim licima. Pored problema kvaliteta pružene zdravstvene nege, postoji i problem efikasnosti. Veliki broj korisnika stvara situaciju da vreme čekanja i vreme neophodno za uvid i analizu korisničkih podataka raste. Zbog ovoga, postoji potreba za automatizacijom delova procedure.

Zdravstvena služba po svojoj prirodi zahteva veliku količinu dokumenata radi pružanja adekvatne nege. Bez savremenih tehnologija, ti dokumenti bi morali biti fizički dokumenti, što stvara problem potrebe za prostorom za skladištenje, kao i problem pristupa dokumentima fizičkim putem.

Ovaj projekat rešava probleme skladištenja i automatizacije procedure i pruža korisnicima mogućnost da neke od neophodnih koraka za dobijanje zdravstvene nege odrade putem korišćenja aplikacije.

Ostatak rada je organizovan kao što je objašnjeno u nastavku. U četvrtom poglavlju su predstavljena srodna istraživanja koja se bave pružanjem zdravstvene nege putem savremene tehnologije.

4. Srodna rešenja

U ovom poglavlju je dat pregled postojećih rešenja za pruženje zdravstvene nege putem savremene tehnologije.

4.1 Integrisani zdravstveni informacioni sistem Republike Srbije (IZIS)

Integrisani zdravstveni informacioni sistem Republike Srbije[4] predstavlja centralni elektronski sistem, koji obezbeđuje jedinstvo podataka u zdravstvu i jedinstvenu informaciono-komunikacijsku infrastrukturu za upravljanje zbirkama podataka i prenos podataka. Dobre strane ovog rešenja su centralizacija koja pruža pristup podacima širom svih zdravstvenih ustanova u skolu zemlje i smanjenje administrativnih troškova putem digitalizacije. Loše strane ovog rešenja su sigurnosni rizici koji dolaze sa centralizacijom podataka, kao i neadekvatna prilagođenost sistema za potrebe random okruženja

4.2 Nacionalni zdravstveni sistem Velike Britanije (NHS)

Nacionalni zdravstveni sistem Velike Britanije[5] je nacionalni zdravstveni sistem koji pruža širok spektar zdravstvenih usluga stanovnicima zemlje. Sastoji se iz više organizovanih jedinica, uključujući bolnice, klinike, lekarske ordinacije i druge zdravstvene ustanove. Osim zdravstvenih usluga, pruža i pristup podacima putem API-ja za integraciju sa raznim sistemima. Dobre strane ovog rešenja su univerzalna pokrivenost, jer pruža jednaku dostupnost zdravstvene zaštite bez obzira na socioekonomske faktore, i elektronski zdravtsveni zapis, koji omogućava efikasno praćenje i upravljanje zdravstvenim podacima. Loše strane su opterećenje sistema koje stvara mogućnost dugih lista čekanja, i administrativna složenost, koja može otežati pružanje adekvatne zdravstvene nege.

5. Korišćene tehnologije

U ovom poglavlju je dat pregled, i objašnjenje korišćenih tehnologija pomoću kojih je realizovan projekat.

5.1 Docker

Docker[3] je skup *platform as a service* (PaaS) proizvoda koji koriste OS-nivo virtualizacije da postignu kontejnerizaciju softvera. Ovime omogućava relativno jednostavno pakovanje, distribuciju i pokretanje softvera u izolovanim okruženjima. Iz ovih raloga je posebno koristan za razvijanje softvera koji prate mikroservinu arhitekturu, jer pruža mogućnost lakog pokretanje i upravljanja servisima

5.2 Single sign-on (SSO)

Single sign-on je metoda autentifikacije koja omogućava korisnicima da se prijave u sistem putem jednog jedinstvenog identifikatora, i time dobiju korisničke privilegije u nekoliko različitih povezanih sistema. Ovime poboljšava korisničko iskustvo, i olakšava administraciju softvera smanjivanjem sigurnosnih rizika.

5.3 MongoDB

MongoDB[6] je NoSQL sistem za upravljanje i čuvanje podataka. Čuvanje podatka je dokument-orijentisano, a podaci se čuvaju u BSON formatu (binary JSON) kao dokumenti u kolekciji. Ovo pruža fleksibilnost pri razvijanju softvera jer nike potrebno definisati šeme za strukturu podataka. MongoDB podržava i indeksiranje, što može znatno poboljšati performanse sistema. Horizontalno je skalabilan, što ga čini pogodnim za razvijanje softvera koji zahtevaju obradu i skladištenje velike količine podataka.

6. Specifikacija zahteva

U ovom odeljku su opisani zahtevi koje je potrebno da ispunjava softversko rešenje predstavljeno u radu.

6.1 Funkcionalni zahtevi

U ovom odeljku su opisani funkcionalni zahtevi koje je potrebno da ispunjava softversko rešenje predstavljeno u radu. Ovi zahtevi su prikazani putem UML dijagrama slučajeva korišćenja, prikazano na slici 1, i putem tabela u nastavku teksta.

Naziv	Zakazivanje termina
Učesnici	Student
Preduslovi	Korisnik je ulogovan, izabrana uloga je “patient”
Koraci	1. Student bira dan 2. Student jedan od ponuđenih termina 3. Student bira tip termina 4. Sistem zakazuje termin
Rezultat	Student ima zakazan termin, termin se dodaje u karton studenta
Izuzeci	Problem autorizacije studenta

Tabela 1 - Opis slučaja korišćenja “Zakazivanje termina”

Naziv	Generisanje uplatnice
Učesnici	Student
Preduslovi	Korisnik je ulogovan, izabrana uloga je “patient”
Koraci	1. Student bira tip uplate 2. Sistem dobavlja informacije o korisniku
Rezultat	Student dobija generisane informacije za uplatu
Izuzeci	Problem u dobavljanju informacija o korisniku

Tabela 2 - Opis slučaja korišćenja “Generisanje uplatnice”

Naziv	Generisanje uverenja
Učesnici	Student
Preduslovi	Korisnik je ulogovan, izabrana uloga je “patient”, korisnik ima uverenje
Koraci	1. Student bira opciju za generisanje uverenja 2. Sistem dobavlja informacije o korisniku
Rezultat	Student dobija generisane informacije za uverenje
Izuzeci	Problem u dobavljanju informacija o korisniku

Tabela 3 - Opis slučaja korišćenja “Generisanje uverenja”

Naziv	Uvid u sopstveni karton
Učesnici	Student, Lekar
Preduslovi	Korisnik je ulogovan, izabrana uloga je “patient” ili “doctor”
Koraci	1. Korisnik bira opciju za prikaz kartona 2. Sistem proverava da li karton postoji 3. Ukoliko ne, sistem otvara karton za korisnika
Rezultat	Korisniku se prikazuje karton
Izuzeci	Problem u dobavljanju kartona

Tabela 4 - Opis slučaja korišćenja “Uvid u sopstveni karton”

Naziv	Uvid u kartone
Učesnici	Lekar
Preduslovi	Korisnik je ulogovan, izabrana uloga je “doctor”
Koraci	1. Lekar bira opciju za prikaz svih kartona 2. Sistem dobavlja sve kartone
Rezultat	Lekaru se prikazuju kartoni
Izuzeci	Problem u dobavljanju kartona

Tabela 5 - Opis slučaja “Uvid u kartone”

Naziv	Izdavanje recepta
Učesnici	Lekar
Preduslovi	Korisnik je ulogovan, izabrana uloga je “doctor”
Koraci	1. Lekar pristupa kartonu korisnika kome izdaje recept 2. Lekar bira opciju za izdavanje recepta 3. Lekar popunjava potrebne informacije za izdavanje recepta 4. Sistem kreira recept
Rezultat	Korisniku se u karton dodaje recept
Izuzeci	Problem u kreiranju recepta

Tabela 6 - Opis slučaja “Izdavanje recepta”

Naziv	Produzivanje recepta
Učesnici	Lekar
Preduslovi	Korisnik je ulogovan, izabrana uloga je “doctor”
Koraci	1. Lekar pristupa kartonu korisnika kome produžuje recept 2. Lekar pristupa receptu koji produžuje 3. Lekar bira opciju produživanje recepta 4. Sistem ažurira stanje recepta
Rezultat	Korisniku se u kartonu ažurira stanje recepta
Izuzeci	Problem u promeni stanja recepta

Tabela 7 - Opis slučaja “Produzivanje recepta”

Naziv	Izdavanje Uverenja
Učesnici	Lekar
Preduslovi	Korisnik je ulogovan, izabrana uloga je “doctor”
Koraci	1. Lekar pristupa kartonu korisnika kome izdaje uverenje 2. Lekar bira opciju za izdavanje uverenja 3. Lekar unosi potrebne informacije za izdavanje uverenja 4. Sistem kreira uverenje
Rezultat	Korisniku se u karton dodaje uverenje
Izuzeci	Problem u kreiranju uverenja

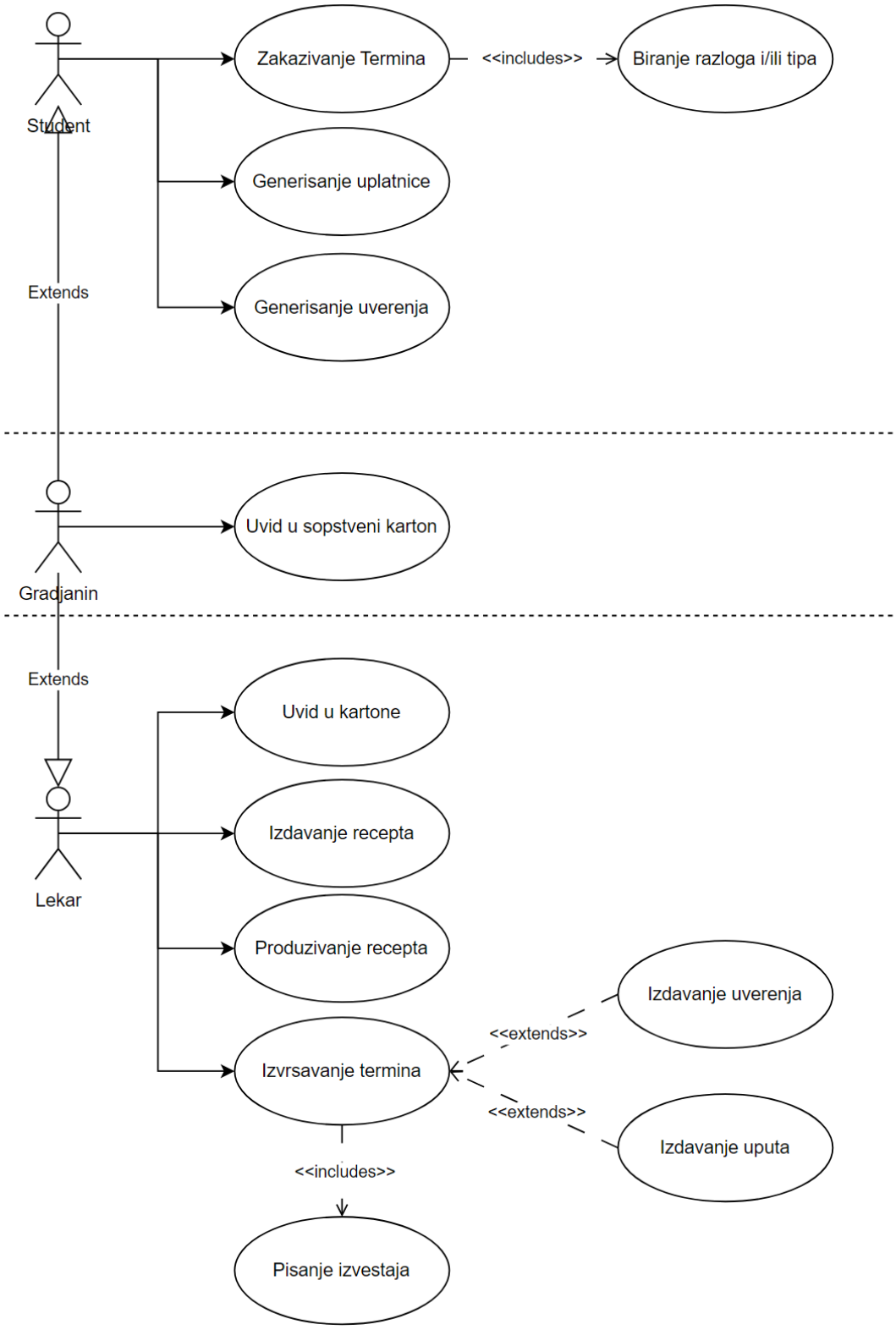
Tabela 8 - Opis slučaja “Izdavanje uverenja”

Naziv	Izdavanje uputa
Učesnici	Lekar
Preduslovi	Korisnik je ulogovan, izabrana uloga je “doctor”
Koraci	1. Lekar pristupa kartonu korisnika kome izdaje uput 2. Lekar bira opciju za izdavanje uputa 3. Lekar unosi potrebne informacije za izdavanje uputa 4. Sistem kreira uput
Rezultat	Korisniku se u karton dodaje uput
Izuzeci	Problem u kreiranju uputa

Tabela 9 - Opis slučaja “Izdavanje uputa”

Naziv	Izvršavanje termina
Učesnici	Lekar
Preduslovi	Korisnik je ulogovan, izabrana uloga je “doctor”
Koraci	5. Lekar pristupa terminu koji izvrsava 6. Lekar bira opciju za izvršavanje termina 7. Lekar unosi potrebne informacije za izvšavanje termina 8. Sistem ažurira termin
Rezultat	Korisniku se u kartonu ažurira termin
Izuzeci	Problem u ažuriranju termina

Tabela 10 - Opis slučaja “Izvršavanje termina”



Slika 1 - UML dijagram slučajeva korišćenja

6.2 Nefunkcionalni zahtevi

U ovom odeljku su opisani nefunkcionalni zahtevi koje je potrebno da ispunjava softversko rešenje predstavljeno u radu. Oni predstavljaju potrebe ponašanja sistema pri korišćenju.

Kontejnerizacija
Sve mikroservise i baze podataka je potrebno pokrenuti kao Docker kontejnere i koristiti Docker Compose alat

Otpornost na parcijalne otkaze
U slučaju parcijalnih otkaza, ostali nepovezani servisi treba da nastave da funkcionišu ispravno

UI/UX
Sistem treba da bude jednostavan i intuitivan za korišćenje

7. Specifikacija dizajna

Ovde je predstavljeno objašnjenje dizajna softverskog rešenja

7.1 Arhitektura sistema

Sistem je osmišljen da prati mikroservisnu strukturu, što znači da se sastoji iz nekoliko komponenti koje međusobno komuniciraju. Sistem za zdravstvo je jedan od tih komonenti i za svoju funkcionalnost komunicira sa autentifikacionim servisom radi autentifikacije i autorizacije prijavljenih korisnika, i sa fakultetskim servisom za proveru posojanja odgovarajućih uloga korisnika.
Struktura sistema za zdravstvo je prikaza UML klasnim dijagramom na slici 2.

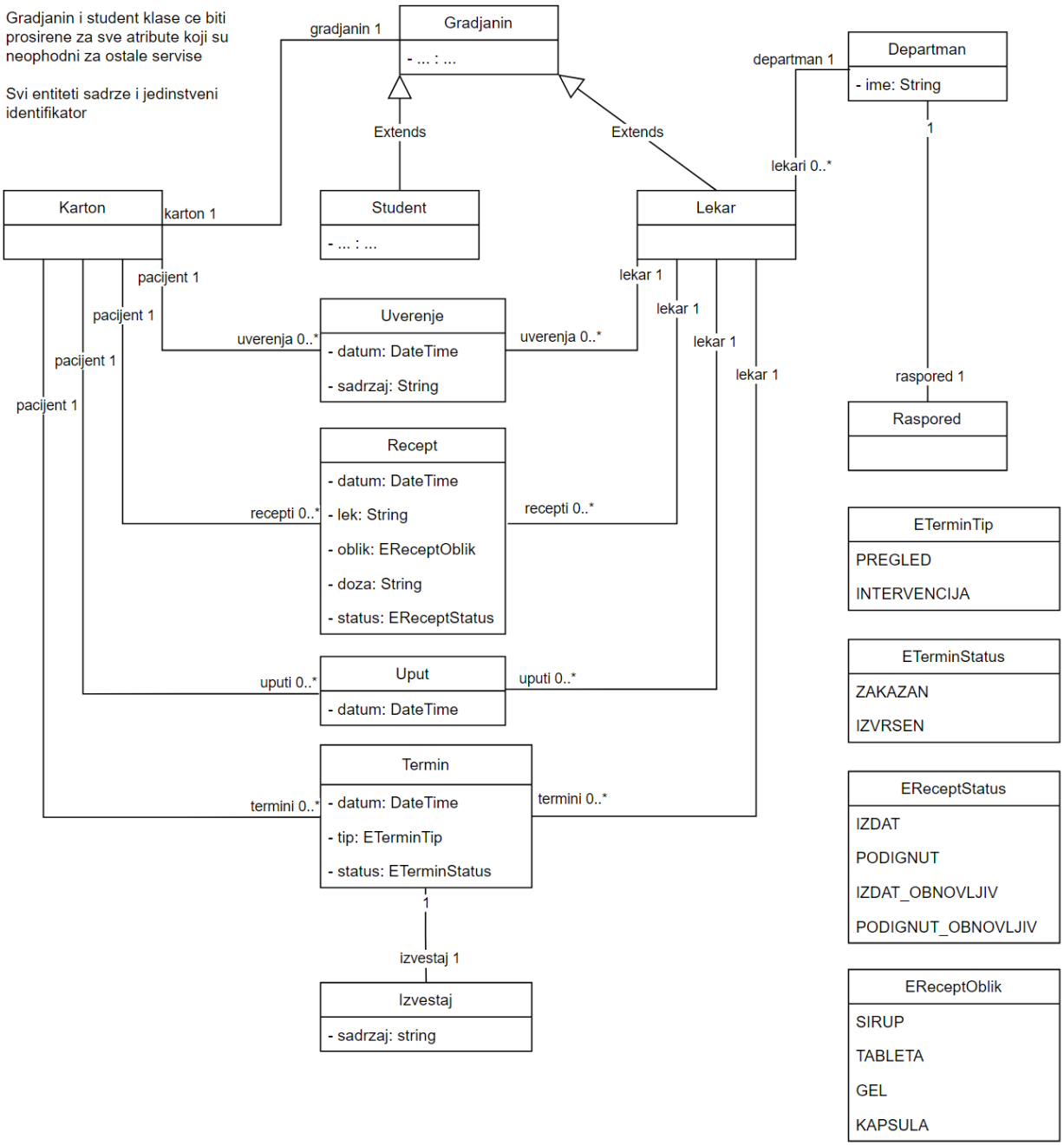
Glavna klasa zdravstvenog sistema je “Karton”. Korisnici sistema se dele na 2 uloge, “Student” i “Lekar”. Svaki korisnik ima svoj karton gde se čuvaju svi podaci vezani za korisnika u okviru sistema.

U karton se smestaju podaci pomocu klasa “Uverenje”, “Recept”, “Uput” i “Termin”. Pomocna klasa za “Termin” je “Izveštaj” gde se čuvaju dodatni podaci o izvršavanju termina. Sve ove klase se kreiraju od strane lekara, osim “Termin” koja se kreira prilikom rezervacije termina u okviru klase “Raspored”.

Klasa “Uverenje” predstavlja podatke o lekarskom uverenju koje je izdato studentu od strane lekara. Klasa “Recept” predstavlja podatke o receptu koji je izdat od strane lekara, i čije stanje može da se menja radi produživanja recepta. Klasa “Uput” predstavlja podatke o uputu koji je izdat studentu od strane lekara. Klasa “Termin” predstavlja podatke o zakazanom terminu, i čije stanje se može menjati prilikom izvršavanje termina, prilikom čega mu se pridodeljuju dodatne informacije putem klase “Izveštaj”.

Klase “Departman” i “Raspored” su klase koje služe kao nosioci službe za korisnike sa ulogom lekar. Unutar “Departman” klase se smešta “Raspored” koji se koristi za formiranje rasporeda termina, kao i za njihovo zakazivanje od strane studenta.

Enumeracije “ETerminTip”, “ETerminStatus”, “EReceptStatus” i “EReceptOblik” služe za određivanje predefinisanih vrednosti za data polja u klasama, kako bi se smanjila šansa za stvaranje greške tokom obrade podataka.



Slika 2 - UML dijagram klasa

8. Implementacija

Ovde su predstavljene neke od implementacija funkcionalnosti softverskog rešenja predstvaljenog u ovom radu.

Implementacija zakazivanja termina

Ovde je opisana implementacija funkcije za zakazivanje termina od strane studenta, koja je prikazana na listingu 1.

Prilikom primanja *POST* zahteva sa potrebnim informacijama, zahtev se prvo presreće od strane *middleware*-a gde se vrši autentifikacija i autorizacija. Nakon toga zahtev ulazi u *handler* koji iz URL-a izvlači naziv departmana i datum za koji se zakazuje termin. Nakon toga se od tela zahteva pravi struktura *SlotFill* koja predstavlja nosilac informacija za dalju obradu zahteva. Ukoliko je struktura JSON-a u telu neodgovarajuća, vraća grešku.

Ako je sve ispravno, poziva se servisni sloj za departman, koji dodaje korisnika u raspored termina, i time rezerviše taj termin za datog korisnika. Ako je neka od prosledjenih informacija pogrešnog formata, vraća grešku.

U servisnom sloju se prvo proverava da li je dati korisnik student u sklopu sistema za fakultet. Ako korisnik nike student, funkcija vraća grešku.
Nakon ove provere se dobavlja departman iz baze, i ažurira se stanje datog termina u memoriji, tako da se studentov identifikator smesti u polje “patientID” čime taj termin postaje rezervisan. Nakon ovog se kreira struktura *AppointmentSchedule* koja postaje novi nosilac informacija za dalji proces. Prosleđivanje ove strukture u funkciju *CreateAppointment*, pravi se struktura *Appointment* koja se perzistira u kartonu korisnika, i koja predstavlja informacije o samom terminu koji će eventualno biti izvršen.
Nakon uspesnog ažuriranja kartona korisnika, perzistira se i promena u strukturi departmana, kako bi pri sledećem prikazu slobodnih termina, prethodno pomenuti termin bio izostavljen.

Ukoliko se svi koraci izvrše bez greške, *handler* će vratiti novu ažuriranu verziju departmana u obliku DTO strukture.

Listing 1 - Prikaz *handler* i *service* sloja funkcionalnosti za zakazivanje termina

```
func (hh HealthcareHandler) AddPatientToSchedule(rw http.ResponseWriter, h *http.Request) {
  vars := mux.Vars(h)
  name := vars["name"]
  date := vars["date"]
  var slotFill *models.SlotFill
  if !utils.DecodeJSONFromRequest(h, rw, &slotFill) {
     utils.WriteErrorResp("wrong structure", 500,
"api/healthcare/department/{name}/docSchedule/{date}/", rw)
     return
  }
  dept, err := hh.DepartmentService.AddPatientToSlot(name, slotFill.ID, date, slotFill.Time,
slotFill.AppType)
  if err != nil {
     utils.WriteErrorResp(err.GetErrorMessage(), err.GetErrorStatus(),
"api/healthcare/department/{name}/schedule/{date}/", rw)
     return
  }
  utils.WriteResp(dept, 200, rw)
}
func (d DepartmentService) AddPatientToSlot(name, id, date, resTime, appType string)
(*models.DepartmentDTO, *errors.ErrorStruct) {
  _, err := d.universityClient.CheckIfStudent(id)
  if err != nil {
     return nil, err
  }
  foundDept, err := d.DepartmentRepository.GetDepartmentByName(name)
  if err != nil {
     return nil, err
  }
  var doctorId string
  for i := range foundDept.Schedule.Date[date] {
     slot := &foundDept.Schedule.Date[date][i]

     if slot.Time == resTime && slot.PatientID == "" {
        slot.PatientID = id
        doctorId = slot.DoctorID
     }
  }
  dateTime := fmt.Sprintf("%s %s", date, resTime)
  appointment := models.AppointmentSchedule{
     DateOfIssue:     dateTime,
     PatientID:       id,
     DoctorID:        doctorId,
     AppointmentType: getTypeEnum(appType),
  }
  _, err = d.HealthcareService.CreateAppointment(appointment)
  updated, err := d.DepartmentRepository.UpdateDepartment(*foundDept)
  if err != nil {
     return nil, err
  }
  return d.DTOService.DeptToDeptDTO(*updated), nil}
```

Implementacija izvršavanja termina
Ovde je opisana implementacija funkcionalnosti za izvršavanje termina od strane lekara, koja je prikazana na listingu 2.

Prilikom primanj *POST* zahteva sa potrebnim informacijama, zahtev se prvo presreće od strane *middleware*-a gde se vrši autentifikacija i autorizacija. Nakon toga zahtev ulazi u *handler* koji iz tela zahteva pravi strukturu *CompletionReport* koja postaje nosilac informacija za dalji proces. Ukoliko je struktura JSON'a u telu neodgovarajuća, vraća grešku.

Ako je sve ispravno poziva se funkcija *UpdateAppointment* servisnog sloja koja prvo kreira strukturu *Report*. Ova struktura predstavlja dodatne informacije za izvršeni termin, i perzistira se. Nakon toga servis dobavlja termin, na osnovu identifikatora, i ažurira ga. Nakon toga sledi poziv *UpdateAppointment* repozitorijuma, koja perzistira termin.

Radi bolje performanse prilikom pretrage, postoji redudansa u sekundarnoj memoriji, tako da se termin čuva i u posebnu kolekciju za termine, i u sklopu strukture korisnickog karton.
Kako bi zagarantovali da se perzistiranjem sistem neće naći u nekonzistentom stanju, oba ova procesa su automatizovana prilikom perzistiranja. Pozivom funkcije *updateByDepth* se postiže perzistiranje u sklopu kartona korisnika. Funkcija prvo dobavlja karton, ažurira ga, i nakon toga ga perzistira.

Ukoliko se svi koraci izvrše bez greške, *handler* će vratiti novu ažuriranu verziju termina u obliku DTO strukture.

Listing 2 - Prikaz *service* i *repository* sloja funkcionalnosti za izvršavanje termina

```
func (h HealthcareService) UpdateAppointment(r models.CompletionReport)
(*models.AppointmentDTO, *errors.ErrorStruct) {
  report := models.Report{
     Title:       r.Title,
     Content:     r.Content,
     DateOfIssue: getNow(),
  }
  insertedReport, err := h.HealthcareRepository.SaveReport(report)
  if err != nil {
     return nil, err
  }
  appointment, err := h.HealthcareRepository.GetAppointmentByID(r.AppointmentID)
  if err != nil {
     return nil, err
  }
  appointment.AppointmentStatus = models.Completed
  appointment.Report = *insertedReport
  appointment.DoctorID = r.DoctorID

  updatedApp, err := h.HealthcareRepository.UpdateAppointment(*appointment)
  if err != nil {
     return nil, err
  }
  return h.DtoServ.AppToAppDTO(*updatedApp), nil
}
func (h HealthcareRepository) UpdateAppointment(appointment models.Appointment)
(*models.Appointment, *errors.ErrorStruct) {
  appointmentCollection := h.cli.Database(h1).Collection(app)
  filter := bson.M{"_id": appointment.ID}
  update := bson.D{
     {"$set", bson.D{
        {"appointmentStatus", appointment.AppointmentStatus},
        {"report", appointment.Report},
     }},
  }
  _, err := appointmentCollection.UpdateOne(context.TODO(), filter, update)
  if err != nil {
     return nil, errors.NewError(err.Error(), 500)
  }
  erro := h.updateByDepth(appointment.PatientID, nil, &appointment)
  if erro != nil {
     return nil, erro
  }
  return &appointment, nil
}
```

9. Zaključak

U ovom radu je predstavljen projekat Zdravsta. Prikazano rešenje olakšava korisnicima da dobiju potrebnu zdravstvenu negu na efikasan način, kao i da pristupe podacima o svom medicinskom stanju na lak način putem centralizovanog sistema. Ovim rešenjem je obuhvaćena i automatizacija i integracija sa ostalim servisima u sklopu univerzitetskog sistema, i olakšava procese gde su potrebne evidencije o medicinskom stanju. Sistem ne podržava integraciju van univerzitetskog sistema, te se za neke oblike zdravstvene nege mora ipak koristiti fizička evidencija. Sistem je napravljen po ugledu na IZIS, ali podržava samo mali podskup ukupnih mogućnosti jednog takvog sistema. Daljim usavršavanje sistema bi trebalo omogućiti integraciju sa spoljašnjim sistemima za zdravstvenu negu, i dodati robustniji način perzistiranja podataka u slučajevima otkaza ili korupcije podataka.

10. Literatura

- [1] Go. 2024. Go Programming Language, https://go.dev/
- [2] React. 2024. React Library, https://react.dev
- [3] Docker. 2024. https://www.docker.com/
- [4] MojDoktor. 2024. https:// www.mojdoktor.gov.rs/
- [5] National Health Service (NHS). 2024, https://www.nhs.uk
- [6] MongoDB. 2024. https://www.mongodb.com