**Univerzitet u Nišu, Elektronski fakultet**

**Katedra za računarstvo**

**Razvoj NodeJS back-end aplikacije**

Nevena Čolić 1360

AppModule

**Predmet: Stručna praksa - Izveštaj**

**Komisija: prof. dr Dragan Stojanović**

**prof. dr Milorad Tošić**

**prof. dr Ivan Petković**

**Niš, 2023**

**Razvijanje NodeJS back-end aplikacije**

Nevena Čolić 1360

**AppModule**

**Predmet:**

Stručna praksa

**Komisija:**

prof. dr Dragan Stojanović, prof. dr Milorad Tošić, prof. dr. Ivan Petković

**Sadržaj**

[**1 - O kompaniji**](#_30j0zll) **3**

[**2 - Zadaci stručne prakse**](#_3dy6vkm) **7**

[**3 - Samostalni rad u pripremi i obavljanju prakse**](#_ioiyb2taecua) **9**

[**4 - Realizacija stručne prakse**](#_o3t9c7jwpbgl) **16**

[**5 - Zaključak**](#_15hti6123fz) **26**

[**Literatura**](#_wf5udqjytpc) **27**

# 1 - O kompaniji

AppModule je švajcarska firma sa kancelarijama u Srbiji i Švajcarskoj, osnovana 2013. godine. Na samom početku tim se sastojao od nekolicine ljudi i prvenstveno su razvijane mobilne aplikacije, međutim kako se tim širio i interesovanje u tehnološkoj industriji raslo, firma se počela baviti i razvojem veb aplikacija. Poslednjih par godina akcenat je stavljen na “Internet stvari” (eng. Internet of Things) i distribuirane sisteme.

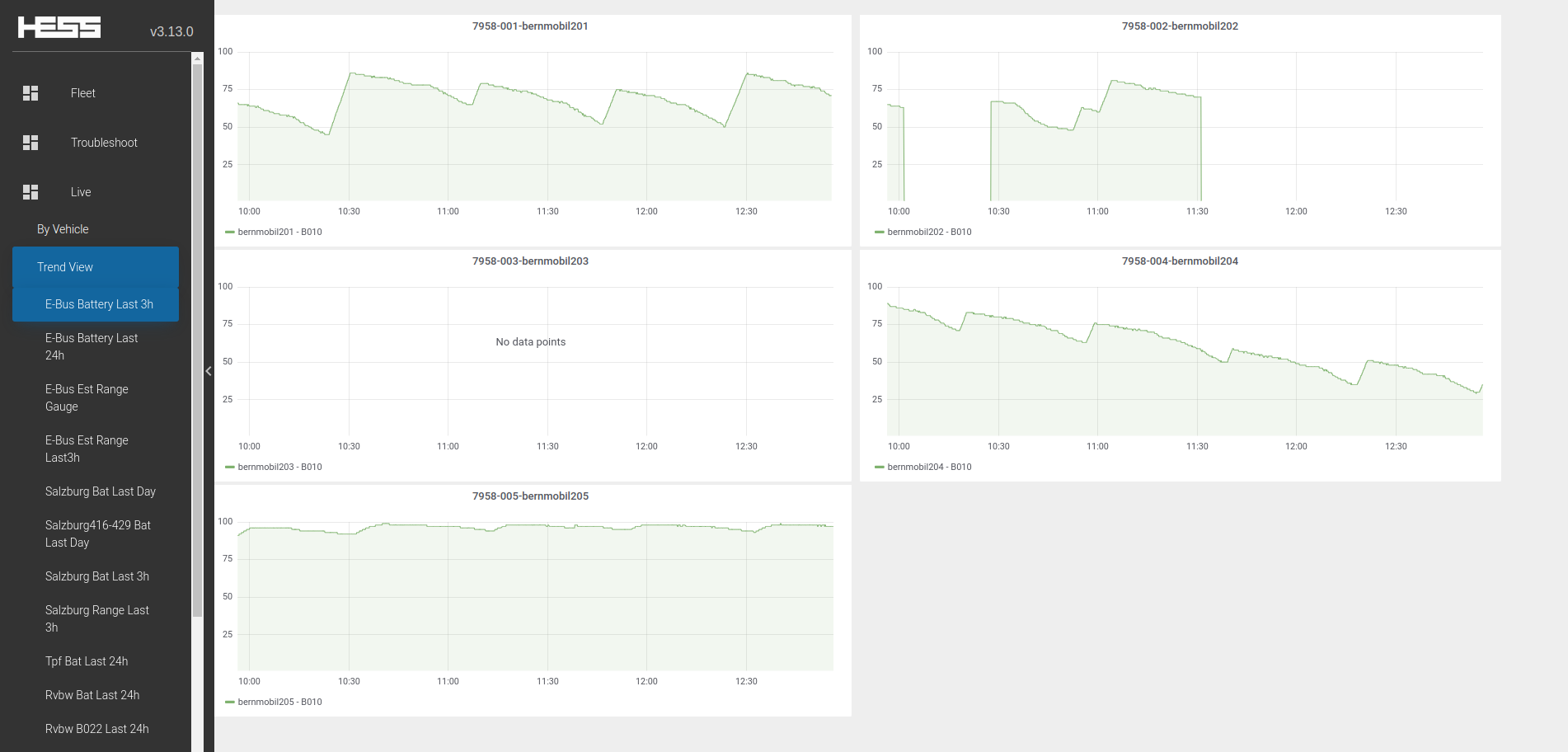
Distribuirani sistemi sa visokim stepenom dostupnosti se mogu nazvati glavnim adutom kompanije. Tim je razvio svoju IoT platformu (SIOT) i oko nje se kreiraju inovativni sistemi i proizvodi.

1.1 Karakteristični proizvodi

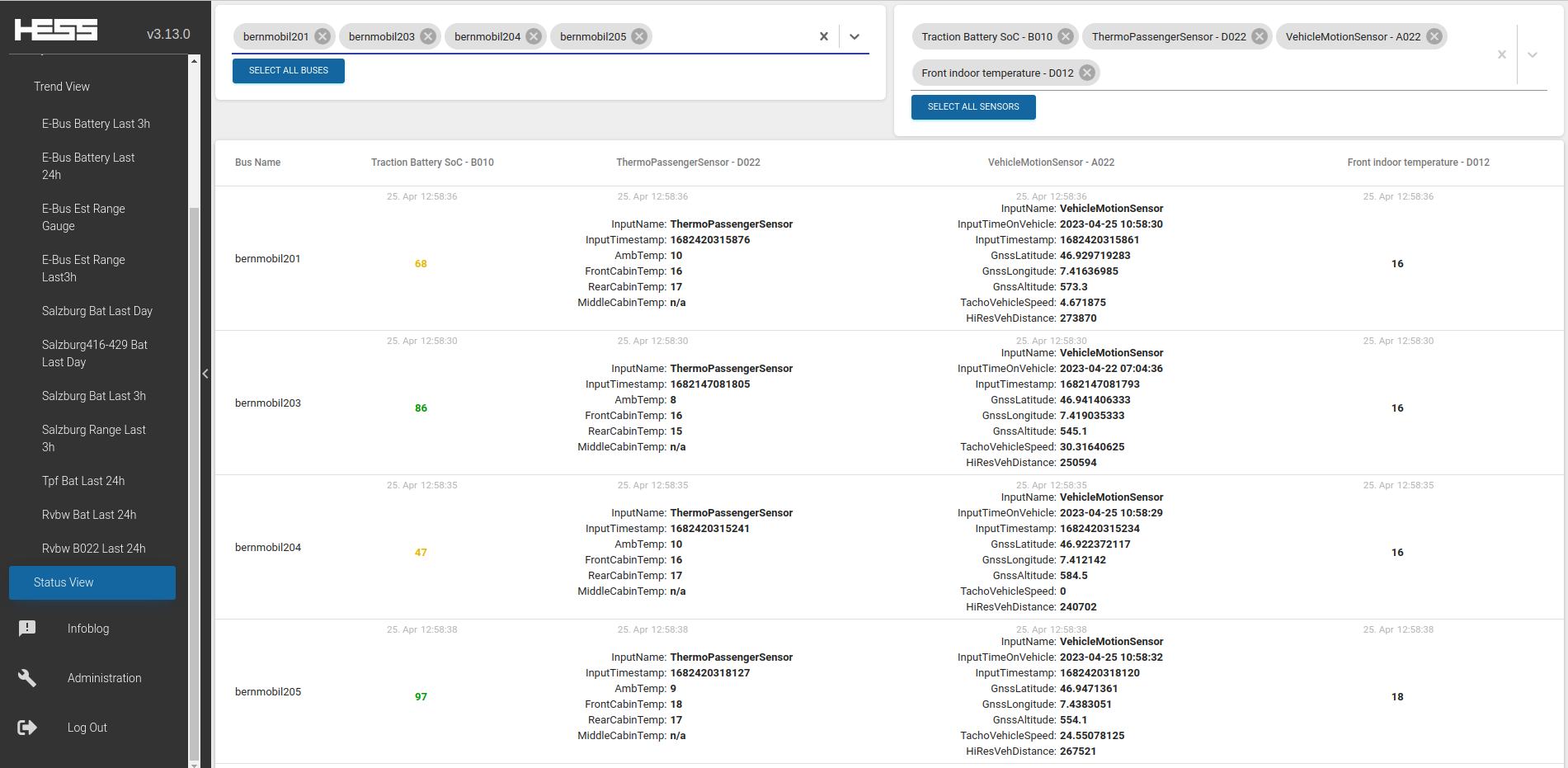
1. HESS

HESS AG je firma koja se bavi proizvodnjom autobusa, trolejbusa i drugim vozilima sa sedištem u Švajcarskoj. Radi se o električnim vozilima čije se baterije pune na različite načine: na svakoj četvrtoj stanici dok putnici izlaze i ulaze (ceo proces traje negde oko 20ak sekundi), ili se na poslednjoj stanici priključe na punjač (par minuta) a postoje i oni koji se pune tokom čitave noći da bi mogli da prelaze veće distance bez dodatnog vremena čekanja. Svaki autobus sadrži senzore koji u svakoj sekundi emitiju svoje stanje.

Kompanija AppModule ovoj firmi pruža usluge praćenja vozila u realnom vremenu, pomaže dijagnostiku, omogućava kreiranje izveštaja za svaki od senzora, na taj način sprečava greške ili pomaže u analizi istih, pruža mogućnost vizuelizacije stanja svakog od autobusa ali i olakšava njihovu administraciju.



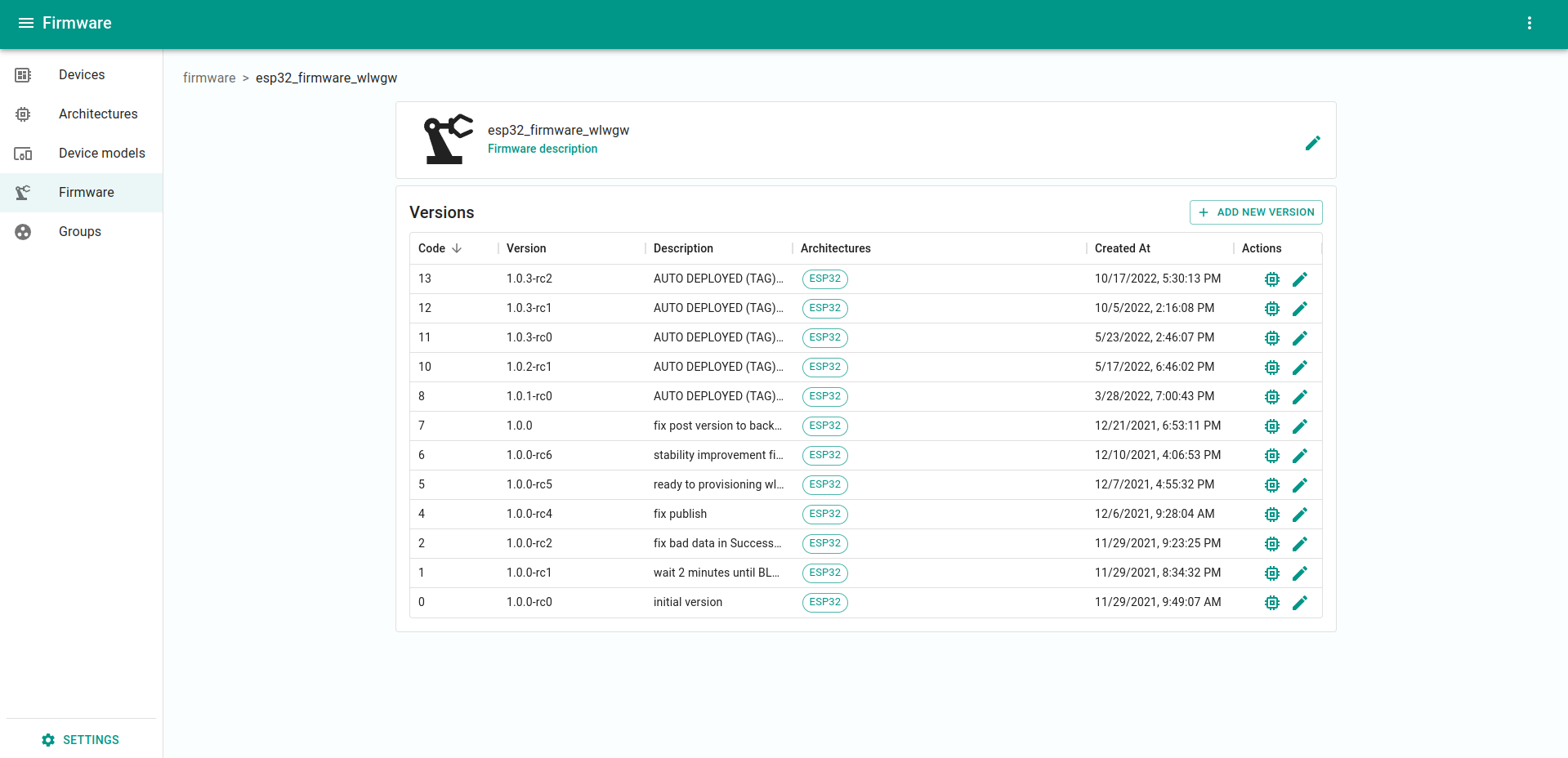
Slika 1. HESS graficki prikaz senzora flote autobusa



Slika 2. HESS status izabranih senzora izabranih autobusa

1. LAIM

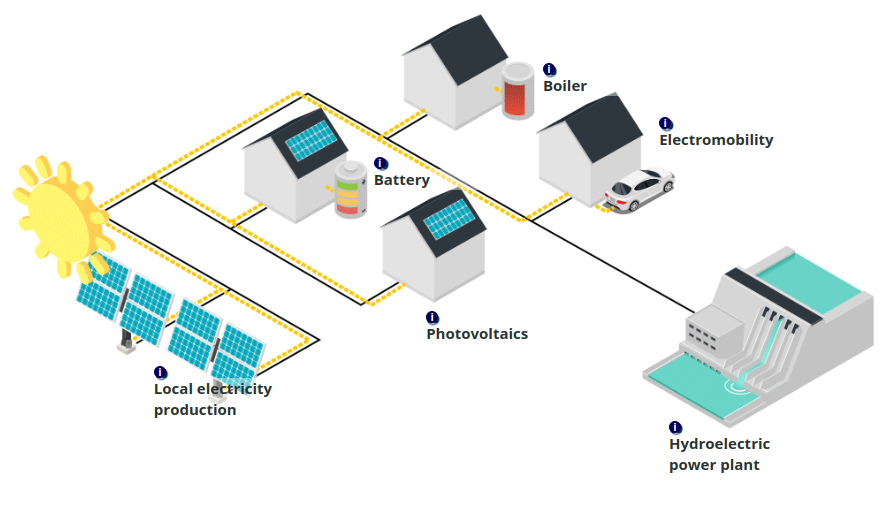
Logistics and Inventory management - interni proizvod kompanije koji se koristi za konfiguraciju i upravljanje hardverskih uređaja koji su na terenu.



Slika 3. LAIM lista instaliranih firmware-a (operativnih sistema)

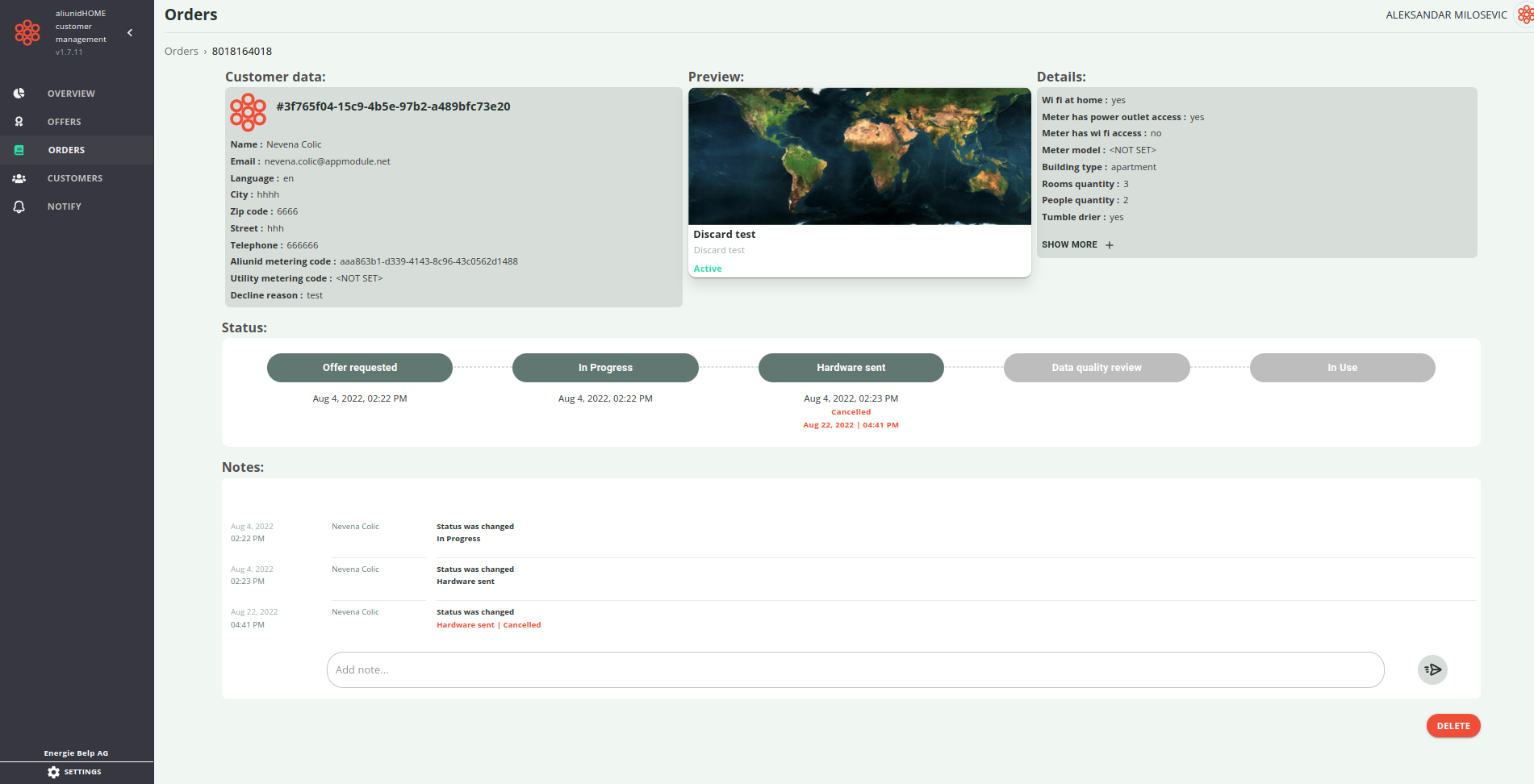
1. Aliunid

Aliunid je kompanija koja se bavi energetikom i fokusira se na bezbedno i jeftino snabdevanje obnovljivih izvora energije. Da bi tako nešto bilo moguće, njihova vizija jeste omogućiti decentralizovano, digitalno i dekarbonizovano snabdevanje koje štedi energiju i CO2. Ovako oni definišu svoj poslovni model. Cilj je da se trenutno snabdevanje energijom u Švajcarskoj prebaci na obnovljive izvore energije do 2050. godine, poveća energetska efikasnost, održi švajcarska hidroenergija i masovno proširi proizvodnja električne energije iz decentralizovane fotonaponske tj. solarne energije. Aliunid je start-ap osnovan 2018. godine finansiran od strane Švajcarske, a koji od januara 2020. finansira i Evropska unija. Brojne kompanije za snabdevanje energijom i proizvođači hidroenergije u Švajcarskoj su uključeni u aliunid AG.



Slika 4. Alinuid ilustracija

AppModule je partnerska firma koja u ove svrhe proizvodi hardver i softver. Harderski tim razvija uređaje (eng. blinker) koji se postavljaju na strujomere i odatle šalju podatke mobilnoj i veb aplikaciji pomoću kojih se vrši podešavanje profila i praćenje podataka.



Slika 5. Aliunid veb aplikacija - HCM (Home Customer Management)

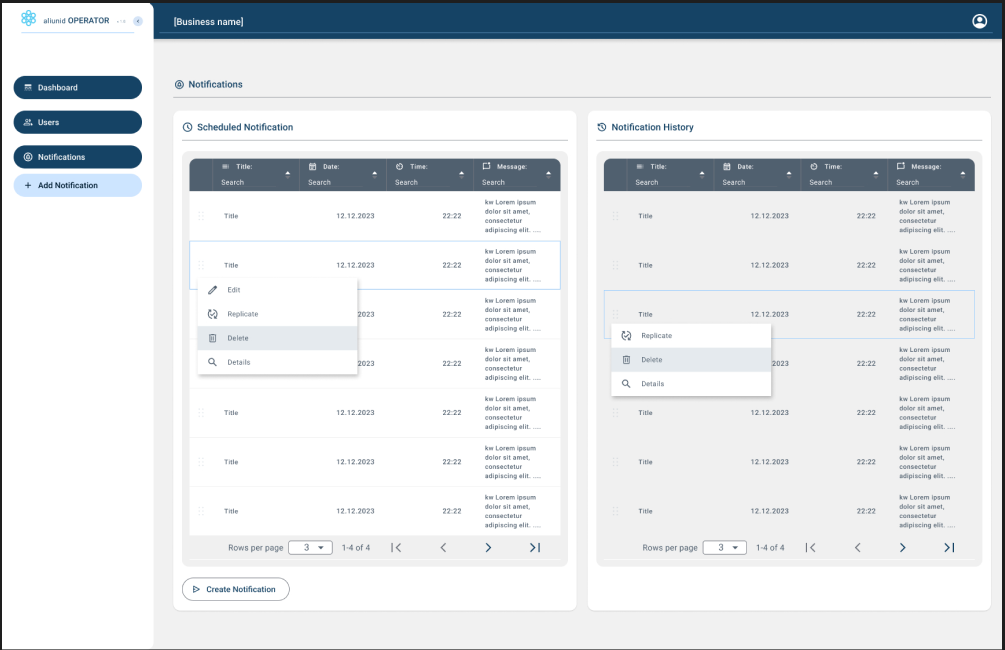
1.2 Lično iskustvo

Ja radim u AppModule kompaniji već četiri godine na full stack poziciji i odgovorna sam za HESS platformu kompletno. Sa kolegom sam razvijala i front-end i back-end aplikaciju a već dve godine projekat održavam sama. Često radim na LAIM-u za interne potrebe firme a u poslednje vreme je zelena energija sve popularnija tako da se najviše razvija aliunid gde povremeno radim back-end a ujedno i vodim front-end projekte kao što je na HCM prikazan na Slici 5. Na tim projektima sam lead i mentor mlađim kolegama. U svim projektima generalno radim i front-end i back-end aplikacije, zavisno od trenutnih potreba firme.

# 2 - Zadaci stručne prakse

Reč je o već pomenutom aliunid projektu, tj. njegovom novom mikroservisu. Aliunid je sačinjen od više mikroservisa i task koji ću ja sada opisati uključuje novi mikroservis koji se bavi notifikacijama.

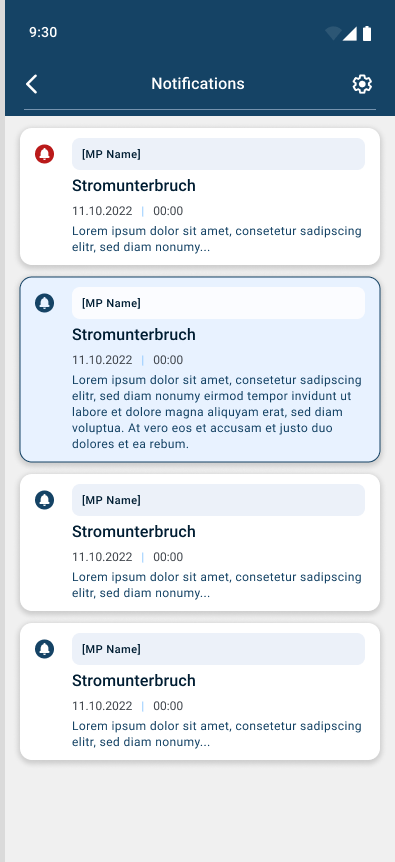
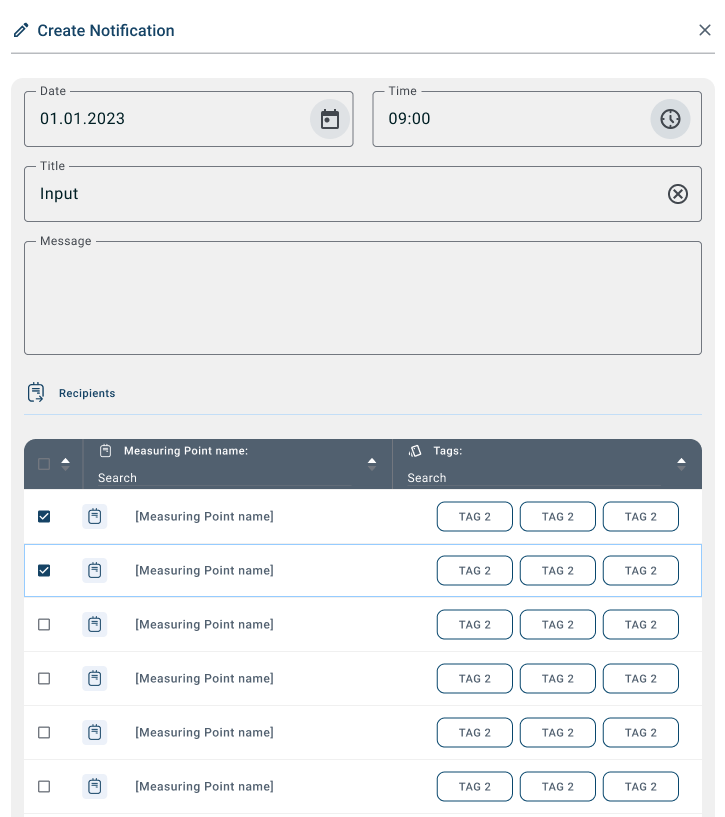
Notifikacija se kreira u veb aplikaciji i podešava se kada će ista biti poslata mobilnim korisnicima. Za slanje notifikacija se koristi notify mikroservis, koji je već implementiran i koristi se u sistemu. Moj zadatak je omogućiti kreiranje notifikacija, njihovo podešavanje i prosleđivanje notify servisu koji je pomoću OneSignal-a poslati konkretne poruke.



Slika 6. Aliunid veb aplikacija Operator za koju se razvija back-end

Notifikacija se šalje svim korisnicima u definisanim biznisima. Endpointi koji su potrebni uključuju:

* Istoriju svih notifikacija (mobilna aplikacija)
* Kreiranje notifikacije za odabrana merna mesta (veb aplikacija)
* Brisanje notifikacije (veb aplikacija)
* Izmena notifikacije (veb aplikacija)
* Lista svih notifikacija (veb aplikacija)
* Obrada Kafka evenata:
  + Kreiranje korisnika
  + Promenu uloge korisnika u sistemu
  + Brisanje uloge korisnika u sistemu



Slika 7. GUI za koji se piše back-end

Objekat koji se šalje notify mikroservisu da bi on znao kome treba

da pošalje notifikacije mora sadržati određene tagove tj. informacije

o korisniku i biznisu kom on pripada:

(<imeBiznisa>:<true>, <imeUlogeUBiznisu>:<true>).

Kreiranje korisnika u sistemu vrši se u drugom mikroservisu tako da

se sinhronizacija ovih mikroservisa obavlja pomoću Kafka tokova

podataka.

Okruženje za rad je Visual Studio Code i u pitanju je NodeJS tj. Express aplikacija. Baza podataka koja je korišćena je PostgreSQL.

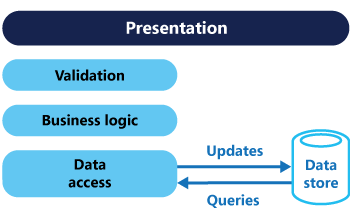
# 3 - Samostalni rad u pripremi i obavljanju prakse

3.1 CQRS

Command and Query Responsibility Segregation - obrazac koji razdvaja operacije čitanja i ažuriranja za skladište podataka. Implementacija CQRS-a u aplikaciju može pojačati njene performanse, skalabilnost i sigurnost. Fleksibilnost stvorena migracijom na CQRS omogućava sistemu da se bolje razvija tokom vremena i sprečava da naredbe ažuriranja uzrokuju sukobe spajanja na nivou domena.

U tradicionalnim arhitekturama, isti model podataka se koristi za upite i ažuriranje baze podataka. To je jednostavno i dobro funkcioniše za osnovne CRUD operacije. U složenijim aplikacijama, međutim, ovaj pristup može postati nezgrapan. Na primer, na strani za čitanje, aplikacija može da izvrši mnogo različitih upita, vraćajući objekte za prenos podataka (DTO) različitih oblika. Mapiranje objekata može postati komplikovano. Na strani pisanja, model može implementirati složenu validaciju i poslovnu logiku. Kao rezultat toga, možete završiti sa previše složenim modelom koji radi previše.

Radna opterećenja čitanja i pisanja su često asimetrična, sa veoma različitim zahtevima za performanse i razmere.



Slika 7. Tradicionalni model

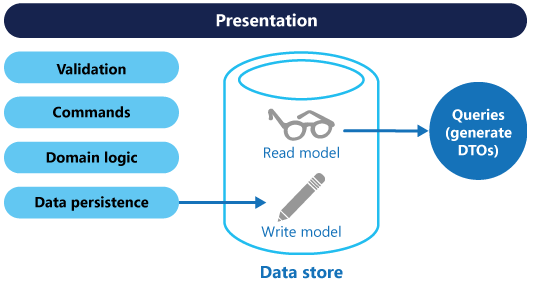
* Često postoji neusklađenost između prikaza podataka za čitanje i upisivanje, kao što su dodatne kolone ili svojstva koja se moraju ispravno ažurirati iako nisu potrebna kao deo operacije.
* Do sukoba podataka može doći kada se operacije obavljaju paralelno na istom skupu podataka.
* Tradicionalni pristup može imati negativan uticaj na performanse zbog opterećenja na skladištu podataka i sloju pristupa podacima, kao i zbog složenosti upita potrebnih za preuzimanje informacija.
* Upravljanje bezbednošću i dozvolama može postati složeno, jer je svaki entitet podložan operacijama čitanja i pisanja, što može izložiti podatke u pogrešnom kontekstu.

CQRS razdvaja čitanje i pisanje u različite modele, koristeći komande za ažuriranje podataka i upite za čitanje podataka.

Komande treba da budu zasnovane na zadacima, a ne na podacima. („Rezerviši hotelsku sobu“, a ne „postavi ReservationStatus na Rezervisano“). Komande se mogu staviti u red za asinhronu obradu, umesto da se obrađuju sinhrono.

Upiti nikada ne menjaju bazu podataka. Upit vraća DTO koji ne obuhvata nikakvo znanje o domenu.

Modeli se zatim mogu izolovati, kao što je prikazano na sledećem dijagramu, iako to nije apsolutni uslov.

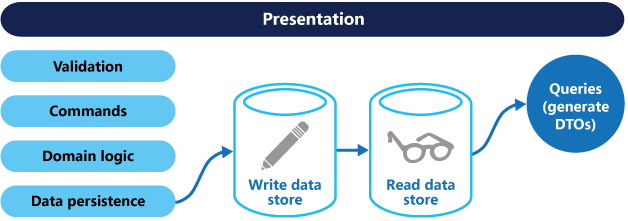


Slika 8. Read i write modeli

Posedovanje zasebnih modela upita i ažuriranja pojednostavljuje dizajn i implementaciju. Međutim, jedan nedostatak je što se CQRS kod ne može automatski generisati iz šeme baze podataka koristeći mehanizme kao što su O/RM alati.

Za veću izolaciju, mogu se fizički odvojiti podaci za čitanje od podataka za pisanje. U tom slučaju, čitana baza podataka može da koristi sopstvenu šemu podataka koja je optimizovana za upite. Na primer, može da skladišti materijalizovani prikaz podataka, kako bi se izbegla složena spajanja ili složena O/RM mapiranja. Možda čak koristi drugu vrstu skladišta podataka. Na primer, baza podataka za pisanje može biti relaciona, dok je baza podataka za čitanje baza podataka dokumenata.

Ako se koriste odvojene baze podataka za čitanje i pisanje, one moraju biti sinhronizovane. Obično se ovo postiže tako što model pisanja objavi događaj kad god ažurira bazu podataka.



Slika 9. CQRS

Skladište za čitanje može biti replika skladišta za pisanje, ili skladišta za čitanje i pisanje mogu imati potpuno drugačiju strukturu. Korišćenje više replika samo za čitanje može povećati performanse upita, posebno u distribuiranim scenarijima gde se replike samo za čitanje nalaze blizu instanci aplikacije.

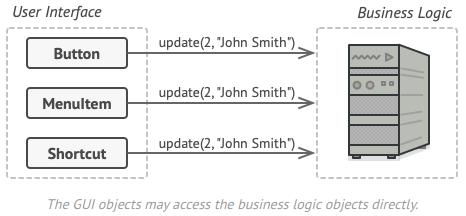
Neke implementacije CQRS-a koriste obrazac “Event sourcing”. Sa ovim šablonom, stanje aplikacije se čuva kao niz događaja. Svaki događaj predstavlja skup promena podataka. Trenutno stanje se konstruiše ponavljanjem događaja. U kontekstu CQRS, jedna od prednosti ovog obrasca je to što se isti događaji mogu koristiti za obaveštavanje drugih komponenti – posebno za obaveštavanje modela čitanja. Model čitanja koristi događaje da napravi snimak trenutnog stanja, što je efikasnije za upite. Međutim, ovaj obrazac dodaje složenost dizajnu. U svakom slučaju, CQRS ima brojne prednosti CQRS-a koje uključuju:

* **Nezavisno skaliranje** **-** CQRS dozvoljava radnim opterećenjima čitanja i pisanja da se nezavisno skaliraju i može dovesti do manjeg broja sporova oko zaključavanja.
* **Optimizovane šeme podataka** **-** Strana za čitanje može da koristi šemu koja je optimizovana za upite, dok strana za pisanje koristi šemu koja je optimizovana za ažuriranja.
* **Bezbednost** **-** Lakše je osigurati da samo pravi entiteti domena upisuju podatke.
* **Odvajanje celina** **-** Razdvajanje strane za čitanje i pisanje može rezultovati modelima koji su lakše održavani i fleksibilniji. Većina složene poslovne logike ide u model pisanja. Model čitanja može biti relativno jednostavan.
* **Jednostavniji upiti** **-** Čuvanjem materijalizovanog pogleda u bazi podataka za čitanje, aplikacija može da izbegne složena spajanja prilikom postavljanja upita. [1]

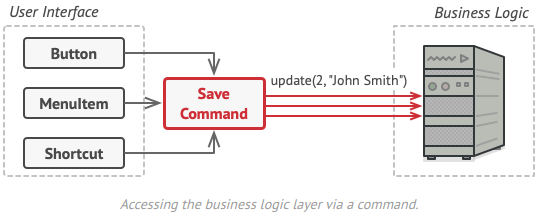
3.2 Command pattern

U pitanju je obrazac ponašanja koji pretvara zahtev u samostalni objekat koji sadrži sve informacije o zahtevu. Ova transformacija omogućava prosleđivanje zahteva kao argumente metoda, odlaganje ili stavljanje u red izvršenja zahteva i obradu operacija koje se ne mogu izvršiti.

Dobar dizajn softvera se često zasniva na principu razdvajanja briga, što obično rezultuje razbijanjem aplikacije na slojeve. Najčešći primer: sloj za grafički korisnički interfejs i drugi sloj za poslovnu logiku. GUI sloj je odgovoran za prikazivanje slike na ekranu, snimanje bilo kog unosa i prikazivanje rezultata onoga što korisnik i aplikacija rade.

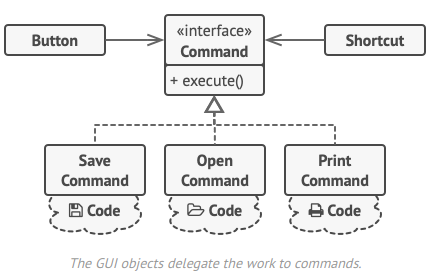
Slika 10. GUI primer za Command pattern

U kodu to može izgledati ovako: GUI objekat poziva metodu objekta poslovne logike, prosleđujući mu neke argumente. Ovaj proces se obično opisuje kao jedan objekat koji drugom šalje zahtev. Obrazac komande sugeriše da GUI objekti ne bi trebalo da šalju ove zahteve direktno. Umesto toga, trebalo bi izdvojiti sve detalje zahteva, kao što su objekat koji se poziva, ime metode i lista argumenata u posebnu klasu komande sa jednom metodom koja pokreće ovaj zahtev. Komandni objekti služe kao veze između različitih GUI i objekata poslovne logike. Od sada, GUI objekat ne mora da zna koji objekat poslovne logike će primiti zahtev i kako će biti obrađen. GUI objekat samo pokreće komandu, koja obrađuje sve detalje.

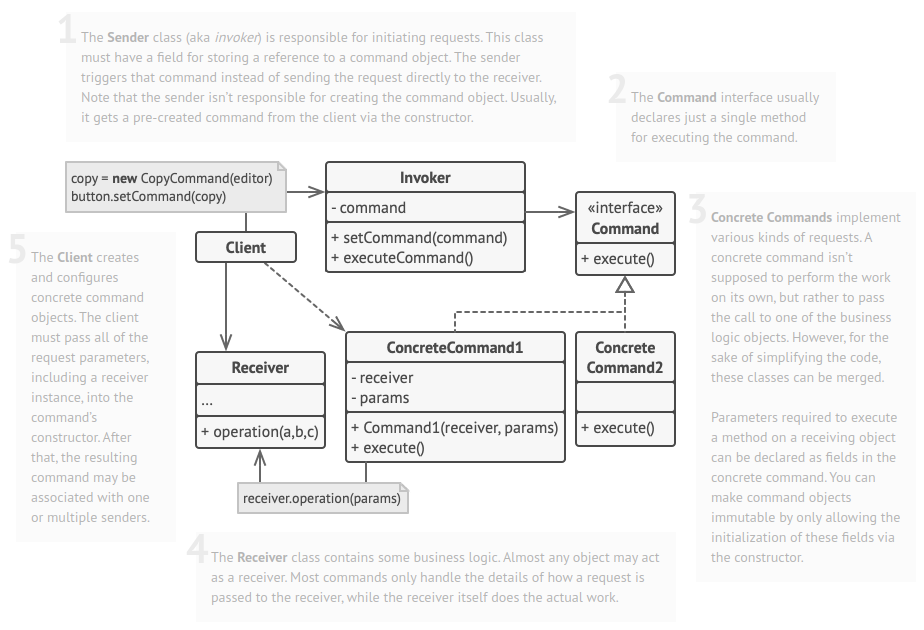


Slika 11. Save komanda

Sledeći korak je da komande implementiraju isti interfejs. Obično postoji samo jedan metod izvršenja koji ne uzima nikakve parametre. Ovaj interfejs omogućava korišćenje različitih komandi sa istim pošiljaocem zahteva, bez povezivanja sa konkretnim klasama komandi. Sada se mogu menjati komandni objekti povezani sa pošiljaocem, efektivno menjajući ponašanje pošiljaoca tokom izvršavanja.



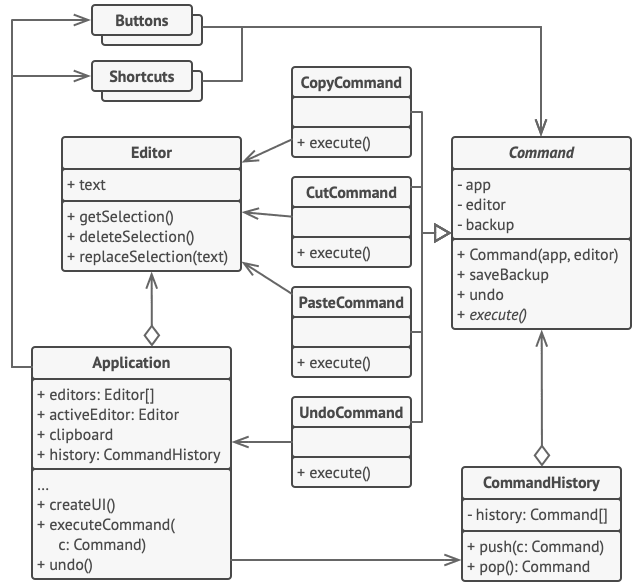
Nakon što se primeni obrazac komande, više nisu potrebne sve te podklase dugmadi da bi se primenila različita ponašanja klikova. Dovoljno je staviti jedno polje u osnovnu klasu Button koja čuva referencu na komandni objekat i naterati dugme da izvrši tu komandu na klik. Drugi elementi GUI, kao što su meniji, prečice ili celi dijalozi, mogu se implementirati na isti način. Oni će biti povezani sa komandom koja se izvršava kada korisnik stupi u interakciju sa GUI elementom. Elementi koji se odnose na iste operacije biće povezani istim komandama, sprečavajući bilo kakvo dupliranje koda. Kao rezultat toga, komande postaju pogodan srednji sloj koji smanjuje povezivanje između slojeva GUI i poslovne logike.



Slika 12. Primer komandnog obrasca

U ovom primeru , komandni obrazac pomaže u praćenju istorije izvršenih operacija i omogućava vraćanje operacije ako je potrebno.

Klijentski kod (GUI elementi, istorija komandi, itd.) nije povezan sa konkretnim klasama komandi jer radi sa komandama preko komandnog interfejsa. Ovaj pristup omogućava unošenje nove komande u aplikaciju bez razbijanja postojećeg koda.



Slika 13. Primena komandnog obrasca

3.3 Apache Kafka

Apache Kafka (Kafka) je open source, distribuirana striming platforma koja omogućava (između ostalog) razvoj aplikacija vođenih događajima u realnom vremenu.

Danas milijarde izvora podataka kontinuirano generišu tokove zapisa podataka, uključujući tokove događaja. Događaj je digitalni zapis radnje koja se dogodila i vremena kada se desila. Tipično, događaj je radnja koja pokreće drugu radnju kao deo procesa. Klijent koji daje porudžbinu, bira mesto na letu ili podnosi obrazac za registraciju su primeri događaja. Događaj ne mora da uključuje osobu – na primer, izveštaj povezanog termostata o temperaturi u datom trenutku je takođe događaj.

Ovi tokovi nude mogućnosti za aplikacije koje reaguju na podatke ili događaje u realnom vremenu. Platforma za striming omogućava programerima da grade aplikacije koje kontinuirano konzumiraju i obrađuju ove tokove izuzetno velikim brzinama, sa visokim nivoom vernosti i tačnosti na osnovu ispravnog redosleda njihovog pojavljivanja.

LinkedIn je razvio Kafku 2011. godine kao posrednika za poruke velike propusnosti za sopstvenu upotrebu, a zatim ju je načinio da bude otvorenog izvora i donirao Apache Softvare Foundation-u. Danas je Kafka evoluirala u najrasprostranjeniju platformu za striming, sposobnu da unese i obradi trilione zapisa dnevno bez ikakvog uočljivog zaostajanja u performansama u skali obima. Organizacije kao što su Target, Microsoft, AirBnB i Netflik, oslanjaju se na Kafku da svojim klijentima isporuče iskustva zasnovana na podacima u realnom vremenu.

Kafka ima tri osnovne sposobnosti:

* Omogućava aplikacijama da objavljuju ili se pretplate na tokove podataka ili događaja.
* Ona tačno skladišti zapise (tj. redosledom kojim su se pojavili) na otporan i trajan način.
* Obrađuje zapise u realnom vremenu (kako se pojavljuju).

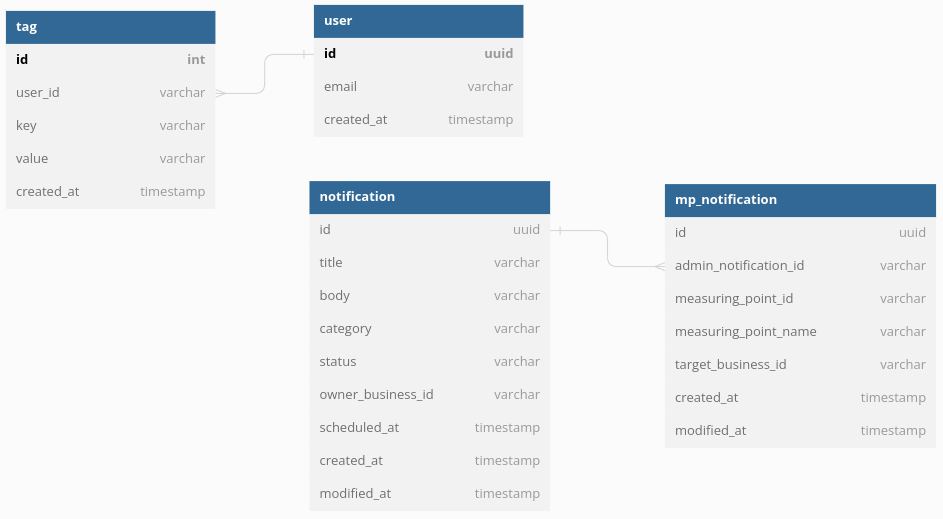
Programeri mogu da iskoriste ove Kafkine mogućnosti preko četiri API-ja:

* **Producer API:** Ovo omogućava aplikaciji da objavi strim na Kafka topik. Topik je imenovani zapisnik koji skladišti zapise redosledom kojim su se javljali u odnosu jedan na drugi. Nakon što je zapis upisan na topik, ne može se menjati ili brisati; umesto toga, ostaje u topiku unapred konfigurisano vreme—na primer, dva dana—ili dok ne ponestane prostora za skladištenje.
* **API za potrošače:** Ovo omogućava aplikaciji da se pretplati na jedan ili više topika i da unese i obradi strim sačuvan u topiku. Može da radi sa zapisima u topiku u realnom vremenu, ili može da unese i obradi prethodne zapise.
* **API za strimove:** Ovo se zasniva na API-jima za proizvođače i potrošače i dodaje kompleksne mogućnosti obrade koje omogućavaju aplikaciji da obavlja kontinuiranu obradu tokova od fronta do kraja – konkretno, da koristi zapise iz jedne ili više topika, da ih analizira ili agregira ili transformiše po potrebi i za objavljivanje rezultujućih strimova na isti ili drugi topik. Dok se API-ji za proizvođače i potrošače mogu koristiti za jednostavnu obradu strimova, API za strimove omogućava razvoj sofisticiranijih aplikacija za strimovanje podataka i događaja.
* **API konektora:** Ovo omogućava programerima da grade konektore, koji su proizvođači ili potrošači za višekratnu upotrebu i koji pojednostavljuju i automatizuju integraciju izvora podataka u Kafka klaster.

Kafka je distribuirana platforma — radi kao otporan na greške, visoko dostupan klaster koji može obuhvatiti više servera, pa čak i više centara podataka. Kafkini toppici su podeljeni i replicirani na takav način da mogu da opslužuju veliki broj istovremenih potrošača bez uticaja na performanse. Kao rezultat toga, prema Apache.org, „Kafka će raditi isto bilo da imate 50 KB ili 50 TB trajnog skladišta na serveru.” [3]

# 4 - Realizacija stručne prakse

4.1 Dijagram baze podataka



Slika 14. Dijagram baze podataka

Tabela user čuva informacije o korisnicima dok id korisnika služi kao strani ključ u tabeli tag koja sadrži informacije o biznisima u kojima je user i njegovim ulogama u istim.

Notifikacije su podeljene u dve tabele zbog problema sa jedinstvenim id-em. Jedna notifikacije kreirana na vebu sadrži niz mernih mesta (eng. measuring point). Za svako merno mesto se kreira i šalje mobilna notifikacija tako da je ove dve stvari potrebno i domenski odvojiti.

4.2 API

Kada su u pitanju sami endpoint-i, njihova dokumentacija izgleda na sledeći način:

1. Istorija notifikacija

[GET] /business/:id/notifications/history

Preuzeti podatke iz mp\_notification tabele i vratiti odgovarajući odgovor:

{

id: uuid,

title: string,

message: string,

time: number, // scheduled time

category: "MP\_SPECIFIC | GENERAL | ...",

businessId: string, // target business id

mpName: string

}[]

1. Kreiranje notifikacije za odabrana merna mesta (eng. measuring point)

[POST] /business/:id/notifications

Objekat koji stiže iz front-end aplikacije:

{

title: string,

body: string,

scheduledAt: number (timestamp),

category: string, "MP\_SPECIFIC | GENERAL | ...",

mpData: {

mpId: string,

mpName: string,

targetBusiness: string

}[]

}

Prvo je potrebno izvršiti određene validacije objekta - da li je polje *scheduledAt* možda postavljeno na prošlo vreme, vratiti odgovarajuću grešku ukoliko jeste.

Zatim generisati status koji je u trenutrku kreiranja - SCHEDULED, a kada notifikacija bude poslata potrebno je ažurirati ovo polje na vrednost HISTORY. Dalje sledi kreiranje odgovarajućih objekata i sačuvati ih u njihovim bazama.

Na kraju je potrebno podesiti cron job tako da se izvršava svakog minuta i obavljati sledeće:

* Iz notification tabele uzeti sve objekte čiji je scheduled\_at <= now()
* Poslati notifikacije za svaki measuring point mobilnim klijentima (preko notify mikroservisa)
* Promeniti status notifikaciji na HISTORY

Telo zahteva koji se šalje notify mikroservisu (Send Notification endpoint):

{

id: mp\_notification's uuid,

push: {

title: <title>,

content: <body>,

data: object, // must contain businessId and mpName

background: **true**

},

query: {

<target\_business\_id>: **true** // tags

}

}

1. Ažuriranje notifikacije

[POST] /business/:id/notifications/:id/change

{

title: string,

body: string,

scheduledAt: number,

category: string, "MP\_SPECIFIC | GENERAL | ...",

mpData: {

mpId: string,

mpName: string,

targetBusiness: string

}[]

}

1. Brisanje notifikacije

[DELETE] /notifications/:id

Brisanje notifikacije se mora izvršiti u obe tabele.

1. Izlistavanje svih notifikacija

[GET] /business/:id/notifications?filter={{history OR scheduled}}

Ukoliko filter nije postavljen, vratiti sve notifikacije. Objekat u rezultatu treba kombinovati podatke iz obe tabele tako da je potrebno kreirati view koji će ih objediniti.

{

adminNotificationId: string, // notification id

title: string,

body: string,

category: string,

status: string,

scheduledAt: number (timestamp),

mpData: {

mpId: string,

mpName: string,

targetBusiness: string

}[]

}[]

1. Kafka event - UserCreated

Potrebno je sačuvati korisnika i njegove podatke u odgovarajuću tabelu i poslati zahtev notify mikroservisu da je potrebno registrovati novog korisnika.

1. Kafka event - UserRoleChanged i UserRoleDeleted

{

id: string,

role: {

business: string,

role: string

}

}

Najpre je potrebno sačuvati promenu u bazi a zatim obavestiti notify mikroservis o novopostavljenim tagovima:

[POST] /account/:id/tags

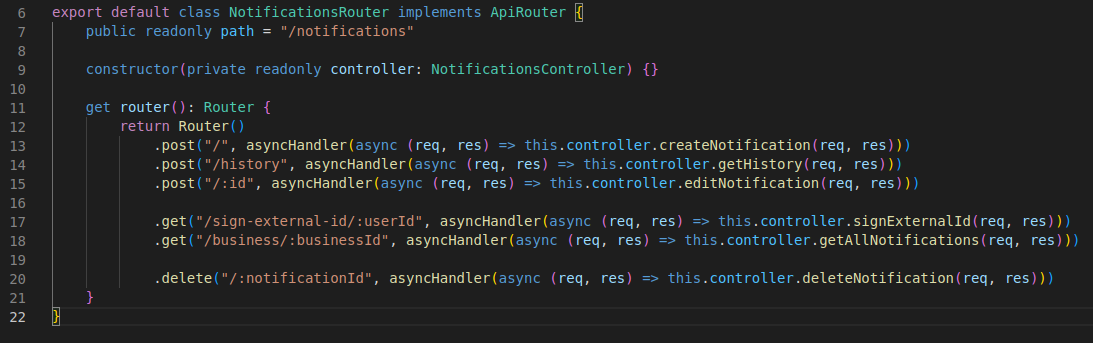
{

<businessId>: **true**,

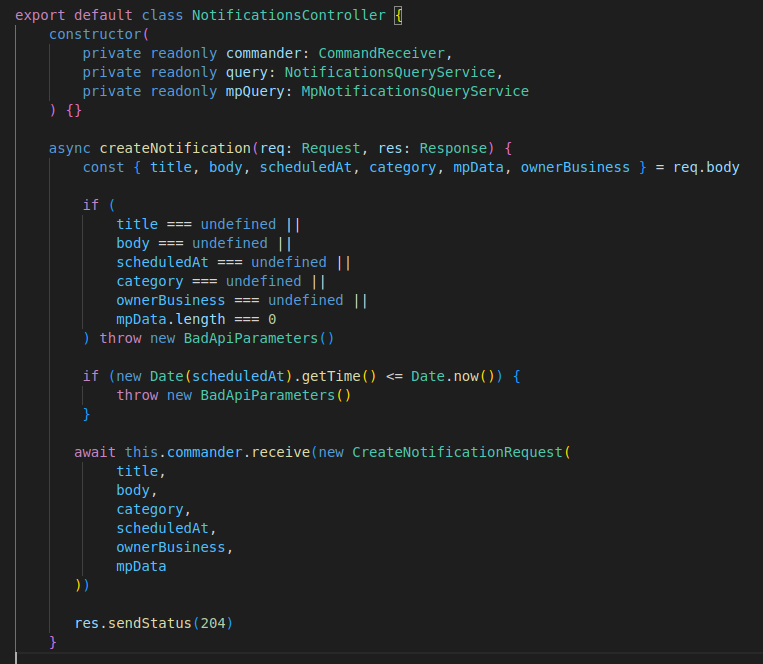
<businessId>\_role: **true**

}

4.3 Implementacija

Slika 15. Ruter

Na Slici 15. prikazan je ruter. Na svaki poziv neke od ruta, okida se odgovarajući kontroler odgovoran za datu rutu. Kontroler može pozvati repository ili queryService, u zavisnosti od rute. QueryService odgovaran je za upite čitanja dok se repository bavi svim ostalim pozivima (CQRS).



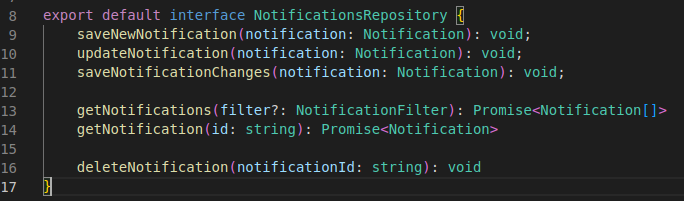
Slika 16. Kontroler - kreiranje notifikacije

Na Slici 16. prikazan je kontrolerska funkcija odgovorna za kreiranje notifikacije. Prvo se vrši validacija svakog od parametara koji stižu a zatim se komandi prosleđuje zahtev za kreiranje notifikacije.

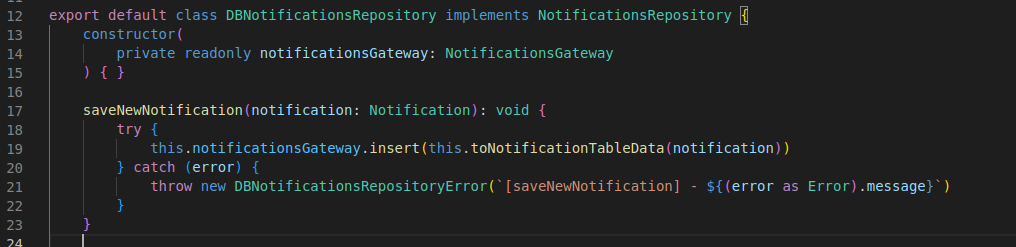


Slika 17. Zahtev za kreiranje notifikacije

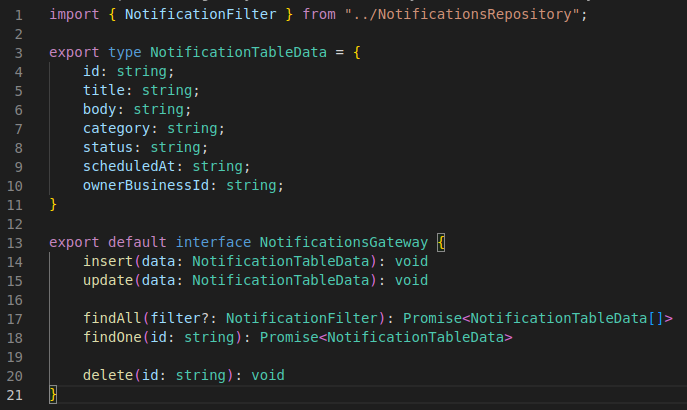
Zahtev za kreiranje notifikacije enkapsulira sve što je potrebno da se jedna komanda izvrši. Počev od neophodnih parametara do same logike. Sve je smešteno u CreateNotificationRequest-u. Jednom kada su napravljeni odgovarajući objekti, poziva se repository da iste smesti u njihove tabele. Interfejs i implementacija repozitorijuma su prikazani na Slikama 18. i 19. Repository funkcija poziva Gateway - u pitanju je implementacija Table data gateway dizajn obrasca. Gateway sadrži najosnovnije funkcije za rad sa jednom tabelom, kao što se i može videti na Slici 20.



Slika 18. Interfejs repozitorijuma notifikacija



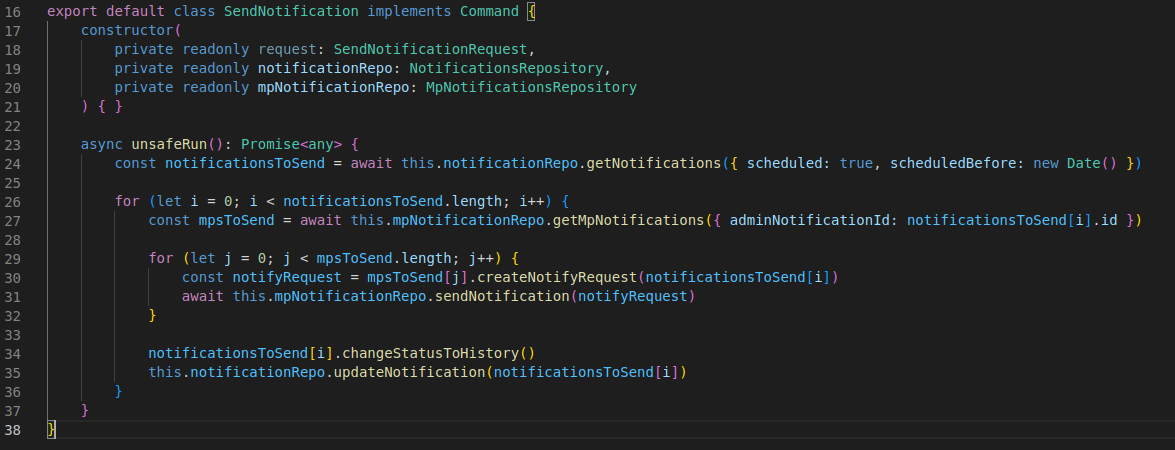
Slika 19. Implementacija repozitorijuma notifikacija



Slika 20. Notification Gateway interfejs

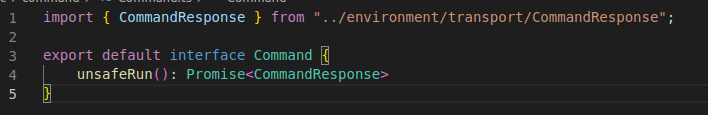


Slika 21. Implementacija gateway-a

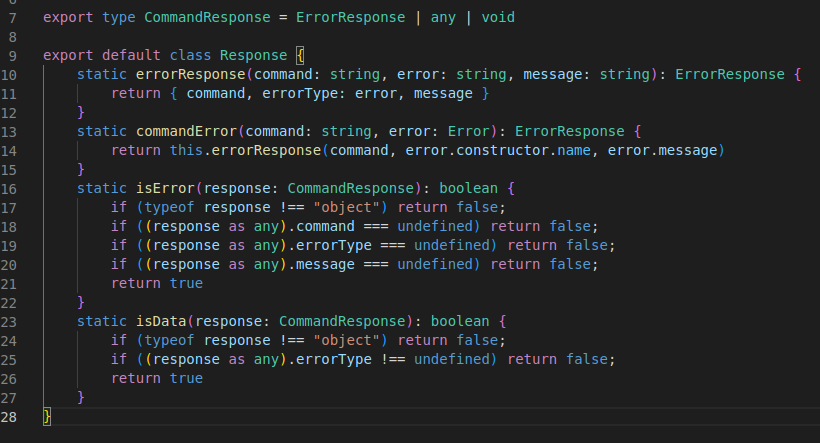


Slika 22. Slanje notifikacije

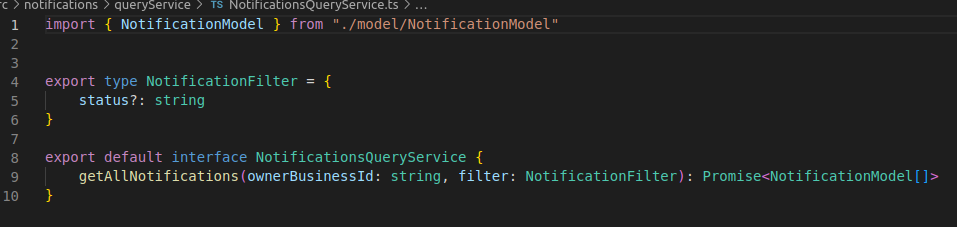
Za svaku od notifikacija kreiranih sa veba (parent iliti admin notifikaciju) postoji niz mernih mesta na koja je potrebno poslati datu. Najpre se iz tabele admin\_notification preuzmu sve čiji je datum zakazivanja pre danasnjeg, tj. sadašnjeg trenutka i za svaku se pronalazi odgovarajuća dete notifikacija, tj. ona koja sadrži podatke o mernom mestu i biznisu kom je potrebno poslati (tačnije svim korisnicima u datom biznisu). Notifikacija se dalje šalje notify mikroservisu koji je odmah šalje korišćenjem OneSignal-a.



Slika 23. Command obrazac

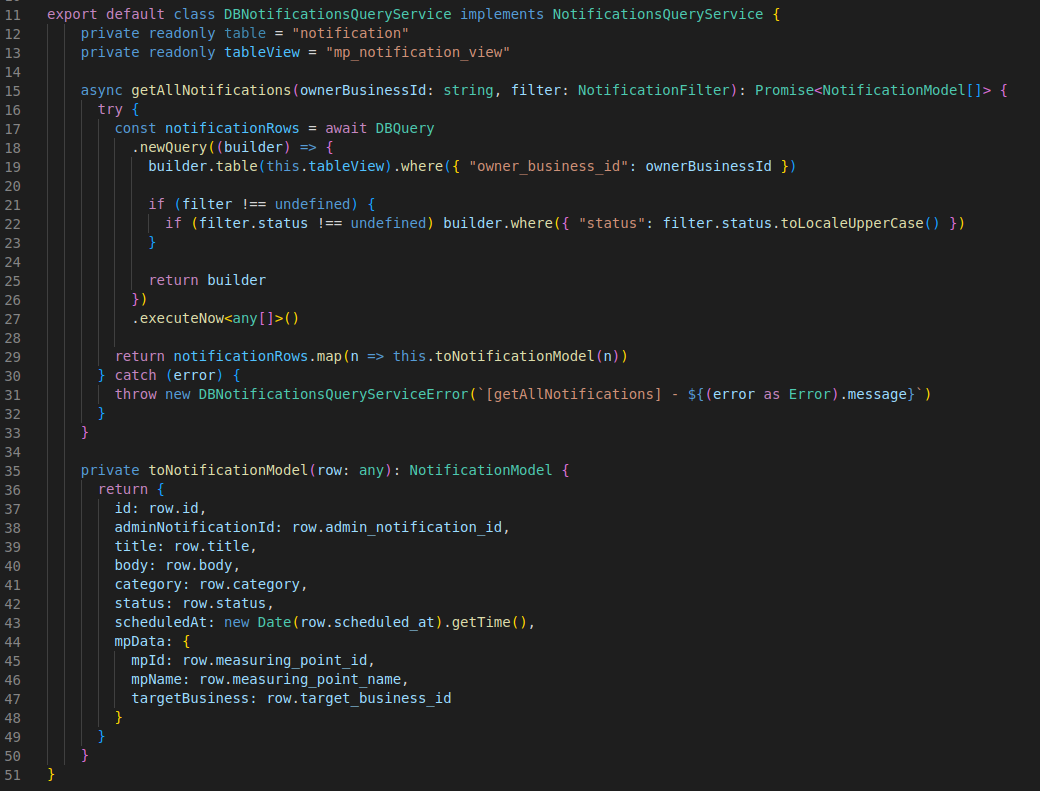


Slika 24. Rezultat komande

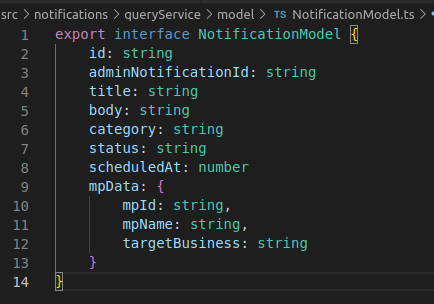


Slika 25. QueryService - interfejs

QueryService služi za sve READ operacije. Njegova implementacija prikazana je na Slikama 25. i 26.



Slika 26. Implementacija Query servisa



Slika 27. Model notifikacije

# 5 - Zaključak

Pozitivna strana ovog projekta je edukacija o CQRS obrascu, Kafki i implementacija odredjenih arhitekturnih obrazaca, sto mi je bilo posebno zanimljivo. Profesionalni aspekt komunikacije u firmi je izvrstan, ali ujedno se potencira i na samostalnom radu, sto omogucava inzenjeru da detaljnije pise, pregleda i finalizira kod i logiku iza njega. Svi mentori su voljni da pomognu i objasne detalje problema koliko god puta to bilo potrebno.

Što se tice poboljšanja, navela bih to što mi nije, na pocetku, bio detaljno objašnjen problem u sistemu i svi njegovi delovi. Trebalo mi je vremena da se naviknem i uklopim u projekat, jer nisam dovoljno dobro poznavala sam sistem a na projektu sam bila sama. Bila mi je potrebna detaljnije opisana povratna informacija jer se sistem sastoji iz više mikroservisa i potrebno je znati bar otprilike kako oni rade i šta se očekuje da se njima pošalje. Međutim kako su svi uvek na raspolaganju da pomognu i atmosfera za rad u firmi je prijatna i bez pritiska, projekat sam uz par dodatnih konsultacija izvela do kraja bez većih poteškoća.

# Literatura

[1] Online resurs

<https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs>

[2] Online resurs

<https://refactoring.guru/design-patterns/command>

[3] Online resurs

<https://www.ibm.com/topics/apache-kafka>