Tiamat
Web aplikacija za upravljanje osobnim podacima

Lucian Tin Udovičić
Broj indeksa : 0165073866
Upravljanje poslovnim procesima
Sveučilište Jurja Dobrile u Puli

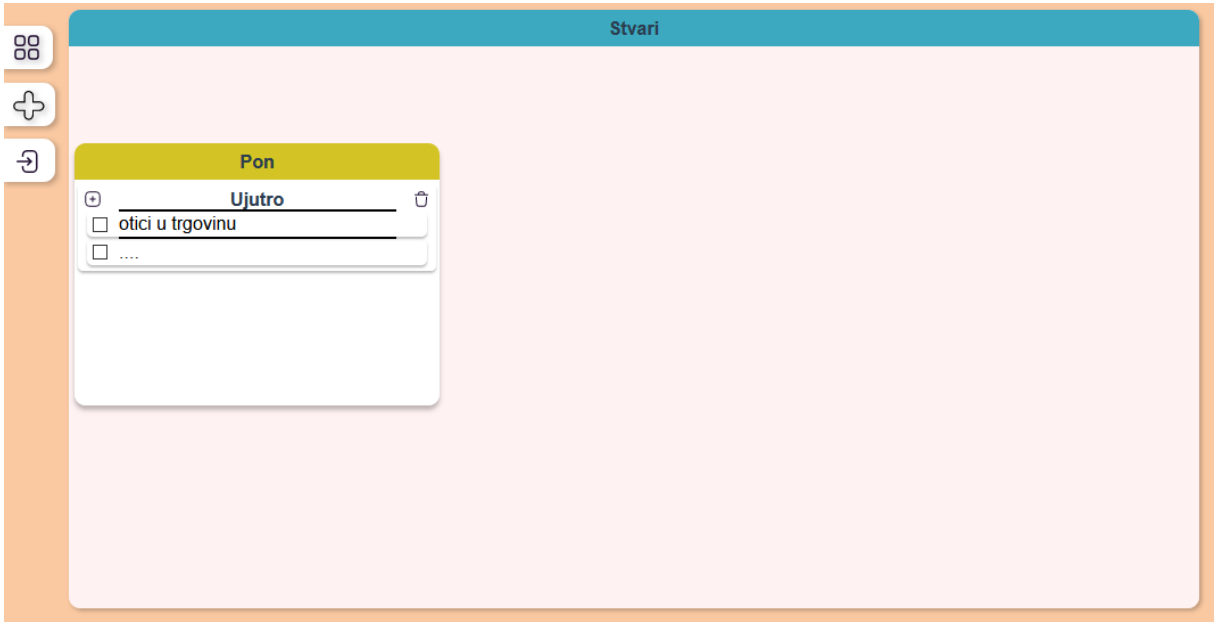
Sadržaj

1 Sažetak

Tiamat je web aplikacija koja služi za upravljanje osobnim podacima poput bilježaka, zadataka, poveznica i slika. Aplikacija je napisana u Vue js-u, za autentifikaciju korsiti firebase te kao glavnu bazu podataka koristi firebase real time database a za spremanje datoteka koristi firebase storage.

Osim samog Vue.js-a korsiti se i Vuex kao posrednik između baze podataka i aplikacije te se time dobije 'single source of truth' u Vuex-u. Za usmjeravanje korisnika koristi se Vue-router.

Aplikacija radi tako što korisnik prvo dobije prikaz svog 'Dashboard-a' gdje može stvoriti novi 'Stuffspace' gdje bi mu se nalazile njegove stvari. Elementi na Dashboard-u se mogu micati po ekranu a klikom na 'Enter' korsinik ode na određeni 'Stuffspace'. Unutar 'Stuffspace-a' postoji više elemenata koji se mogu micati, proširiti, brisati, dodati, itd. Svaki element ima svoj sloj po kojem se smije micati, container se smije micati samo po ekranu, groupa se smije micati samo unutar containera. Unutar grupa korisnik može stvoriti novi section u kojem bi dodao razne elemente poput bilješki, zadataka, poveznica i slika. Cilj tih slojeva je da korisnik može lakše organizirati i kategorizirati svoje podatke.



Slika 1.1: Prikaz Prozora

2 Uvod

Ideja je da se bolje iskoristi prostor na ekranu, umjesto običnog popisa elemenata, možda bi se postigla bolje vizualizacija podataka u 2D prostoru sa vidljivim granicama gdje spadaju podaci jedne kategorije a gdje druge. Može se zamisliti kao KanBan ploča samo što nije popis nego više prozora koji se mogu micati i rastegnuti, te bi svaki taj prozor sadržavao manje prozorčiće sa raznim podacima koji se mogu micati iz jednog prozora u drugi. Korisnik bi onda mogao bolje organizirati svoje podatke jer umjesto popisa koristi nešto kao 'oglasnu ploču'.

Volio bih reći da je preko 70% željenih featura implementirano, neke stvari sam skužio tek kasnije da sam pogriješio, za neke kao npr. proširivanje grid-a nisam baš skužio kako implementirati ali to iz pogleda UX-a a ne koda, dosta sam razmišljao, ne samo za to, kako bi korisnicima bilo lakše koristiti aplikaciju. Da se mogu vratiti natrag u virijeme, barem pola koda bi bilo drugačije, a neke stvari ja mislim da sam pogodio kako najbolje napraviti, rijetko ali desilo se.

3 Motivacija

Umjesto neke obične aplikacije sa stupcem slika ili popisa proizvoda htio sam napraviti aplikaciju sa više interakcije između korisnika i same aplikacije.

Nisam baš razmišljao koje tržište ciljam sa tom aplikacijom, imao sam ideju onoga što želim napravit a tek kasnije sam razmišljao kako bi moja ideja bila korisna bilo kome, pa i meni. Vjerojatno bi to bili ljudi koji koriste neke slične aplikacije za bilješke i sl. ali bi im se više svidjelo raditi sa ovom. Ideja je da bi se moglo raditi sve kao u konkurentnim aplikacijama poput Trello, Evernote, OneNote, Todoist, Notion ,... samo umjesto popisa, korisnik radi nad prozorima. Aplikacija bi privukla korisnike samo zbog načina vizualizacije njihovih podataka, toj je taj glavni feature aplikacije.

Prednost ove aplikacije je prvenstveno vizualizacija podataka, ništa osim toga.

Najveća mana i ogroman razlog zbog kojeg korisnici ne bi skočili na ovu aplikaciju je potreba za velikim ekranom, na mobilnim uređajima bi bilo gotovo nemoguće koristiti aplikaciju zbog potrebe za velikim prostorom, jer je teže za koristiti.

S	W
- vizualizacija podataka - veča sloboda za organizaciju podataka - bolja interakcija sa podacima	- potreba za velikim ekranom - nakon puno podataka možda bi popis bio pregledniji

O	T
- konkurencija nemože vrlo lako prebaciti podatke na takav prikaz, možda su limitirani početnom arhitekturom podataka	- želja korisnika za jednostavnijim aplikacijama

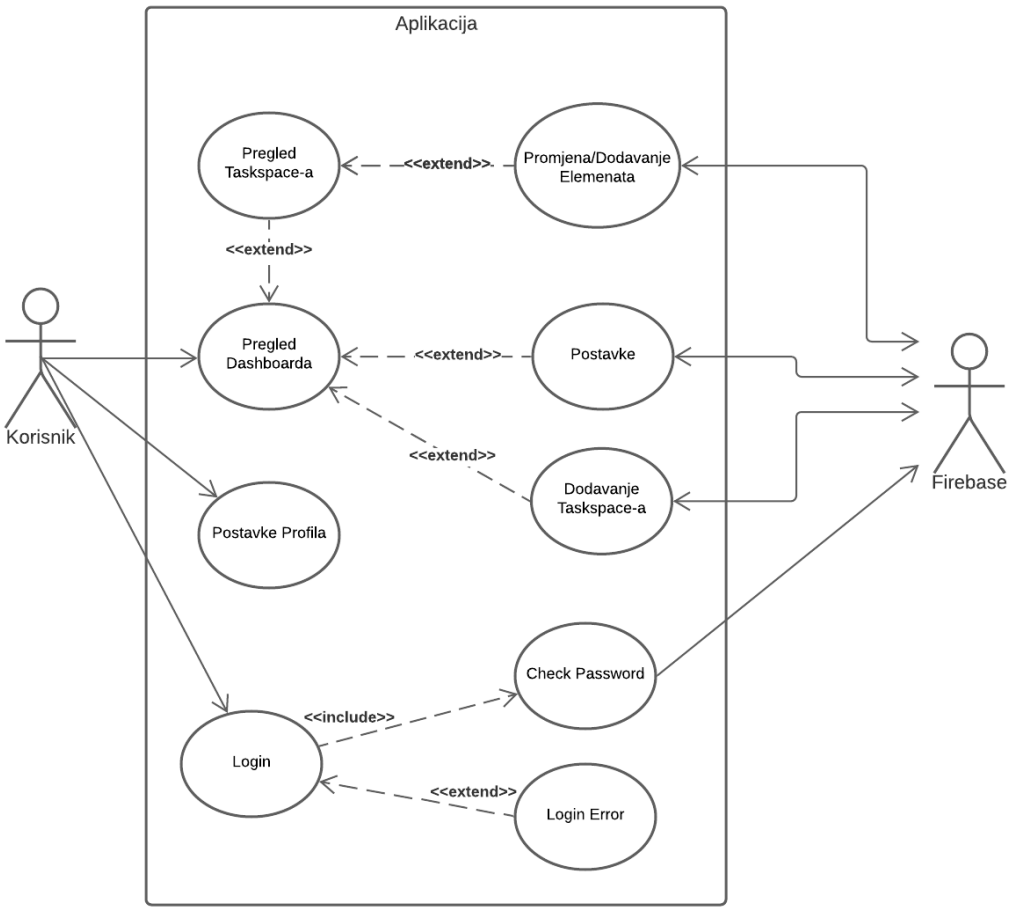
Tablica 3.1: SWOT analiza

4 Razrada funkcionalnosti

Od početne ideje aplikacije te od crtanja na papiru i osmišljavanja featura, dosta se promijenilo nakon što sam bolje razumio kako radi sam Vue i kako bi se najlakše izveo projekt.

4.1 Use case model

Nakon što korisnik uđe us voj račun on može pregledavati razne elemente web aplikacije, razumijem da nije prikazano da tek nakon što se ulogira da može pregledavati svoje podatke no to je samo ideja koja je bila na početku, malo se toga promijenilo, npr. Class model sam promijenio jer sam bolje razumio kako izgleda aplikacija dok use case model nije neš strašno ako bude relativno točan.

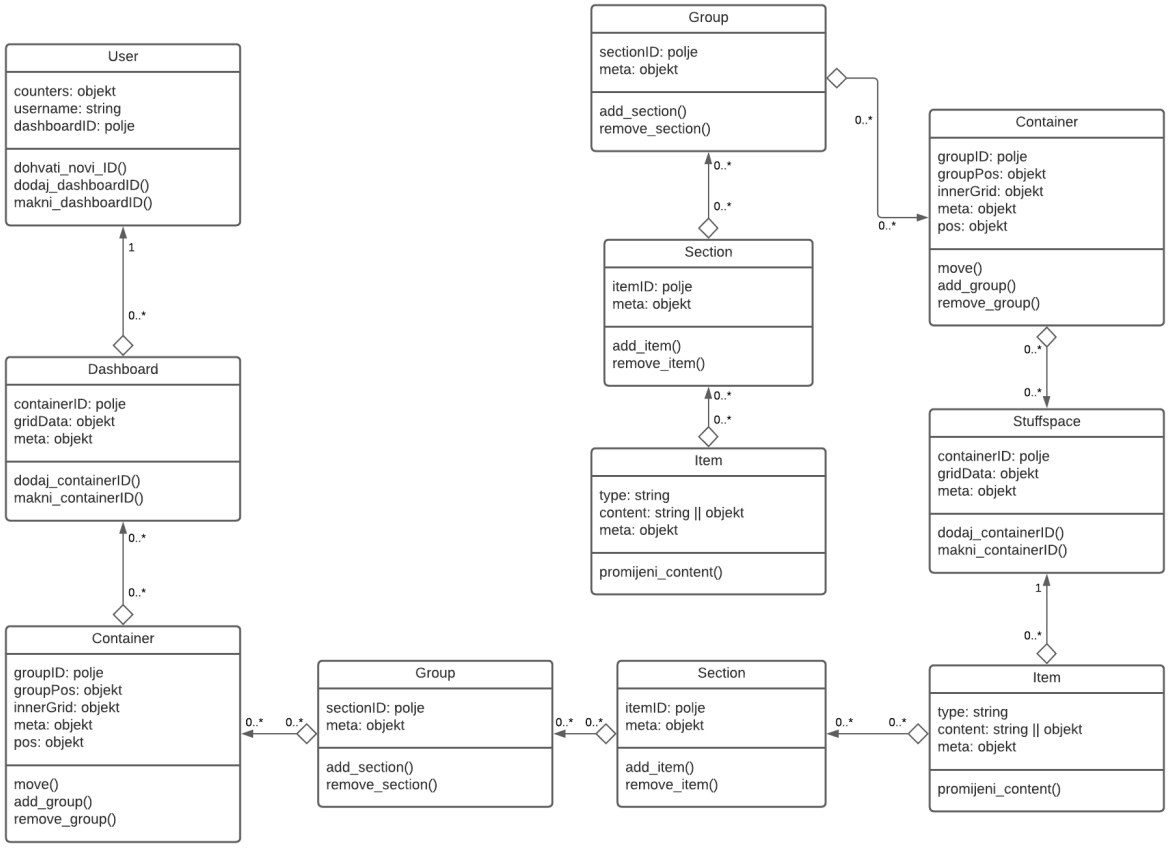


Slika 4.1: Use case model

4.2 Class model

Svako dijete ovisi o roditelju, User može imati 1 ili više Dashboard-a, Dashboard može imati jedan ili više container-a itd.. sve do prvog Item-a koji mora imati samo jedan ID Suffspace-a, arhitektura podataka unutar baze je takva da različiti roditelji smiju imati istu djecu (o tome malo kasnije) , to nije implementirano unutar aplikacije no to je vrlo bitna značajka tih podataka pa sam mislio da je bitno to uključiti u model, zato ima dosta veza gdje dijete smije imati 0 ili više roditelja te roditelj smije imati 0 ili više djece.

Nema ništa što sprječava korisnika da 2 različita Dashboarda imaju istog Containera (osim što nije implementirano) te također da Dashboard ima više Containera (to je implementirano). Meta se koristi za spremanje naziva elementa i još par stvari koje se ne koriste.



Slika 4.2: Class model

5 Implementacija

Nije sve opisano u implementaciji, neke elemente nisam stavio jer su ili maleni ili ne igraju neku veliku ulogu u aplikaciji, također nisam opisao početnu stranicu i ostale ”propagandne” stranice jer osim v-for i vue routera nema ništa posebnog kod njih.

Sve sam ja napisao, nisam koristio neke komponente sa interneta, mislio sam malo bolje shvatit kako to sve radi pa napisati sam, valjda je bilo vrijedno vremena....

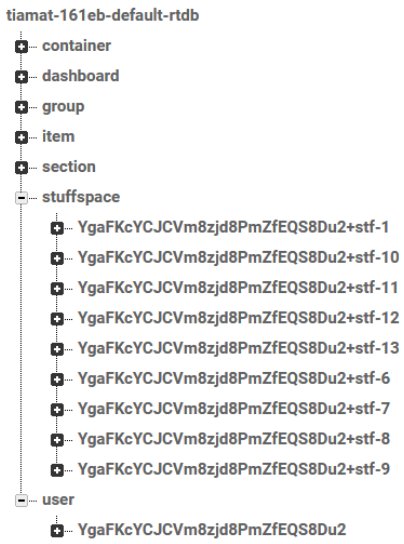
5.1 Arhitektura

Svaka komponenta poput container, group, section, item,... je odgovorna za svoje podatke i za stvaranje ”djece”, te prosljeđivanje podataka o njihovim roditeljima kako bi se mogli izbrisati iz njih i sl. Svaka promijena prolazi kroz Vuex (poneka ima još jedna korak prije nego što dođe do Vuex-a), on se najviše koristi za mijenjanje elemenata u bazi i za podatke vezane oko korisnika, također postoji jedna varijabla koja prati dali je baza spremna, to je vrlo bitno jer ako korisnik osvježi stranicu onda se firebase mora ponovno inicijalizirati prije nego što se mogu dohvatiti podaci.

5.1.1 Podaci

Firebase ima jedan hook za promijene u autentifikaciji, u njemu se inicijalizira baza, to je vrlo bitno, ne samo zato što sam postavio pravila unutar firebase-a nego zbog točnog adresiranja podataka. Svaki element u svom imenu sadrži UID korisnika, onda ima znak + pa tip tog elementa i znak - te id tog elementa, i tako se adresira svaki element, unutar podataka o korisniku su brojači koji mi daju novi ID za elemente.

Većinom svaki element sadrži ID-eve svoje djece, kako nije svaki element jedan veliki upis u bazi nego mnogo manjih, niti 1Kb memorije, odlučio sam se za ravnu strukturu baze, nije hijerarhijska jer bih volio svaki element posebno adresirati bez da pokupim i njihovu djecu, onda mi je također lakše prenijeti dijete sa jednog roditelja na drugog jer ne moram kopirati cijelo dijete nego samo promijeniti jedan broj u polju roditelja. Na pitanje zašto koristim real-time-database a ne firestore, ispunio sam upitnik od googla i kažu da ako imam mnogo malenih promjena sa malom količinom podataka onda mi je bolje koristiti RT-db.

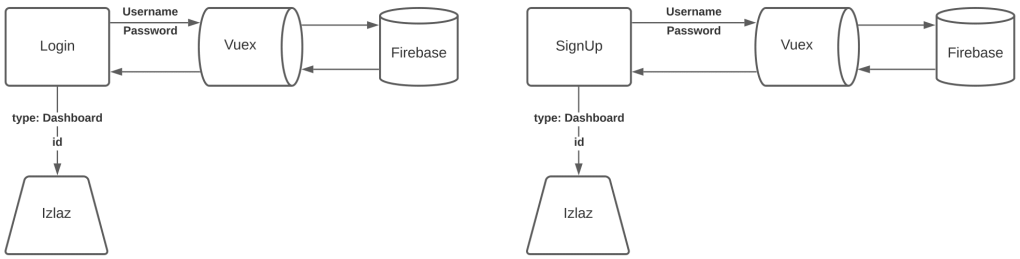


Slika 5.1: Struktura podataka unutar baze

5.1.2 Login i Signup

Jedina razlika između signup komponente i login komponente jesu podaci koji se trebaju unijeti i kod signup-a se mora postaviti baza kako bi novi korisnik imao početne podatke za prikazivanje. Sve ide preko vuex-a jer se osim firebase autentifikacije mora inicijalizirati vuex sa praznim podacima i nova instanca DatabaseHandler-a koji prima korisnikov UID za točno adresiranje podataka.

Nakon uspješne autentifikacije , korisnik se proslijedi na Grid koji prima svoj tip i ID unutar baze. To je prikazano sa elementom ”Izlaz” jer ima više načina za doći do Grid komponente. ID se proslijedi kao query string kako bi se mogao koristiti vue-router za navigaciju između različitih instanci Grid-a. Grid je glavna komponenta koja prikazuje container, group, section i item, no na različite načine ovisno o tipu tog Grida (Dashboard ili Stuffspace).



Slika 5.2: Login i Signup diagram

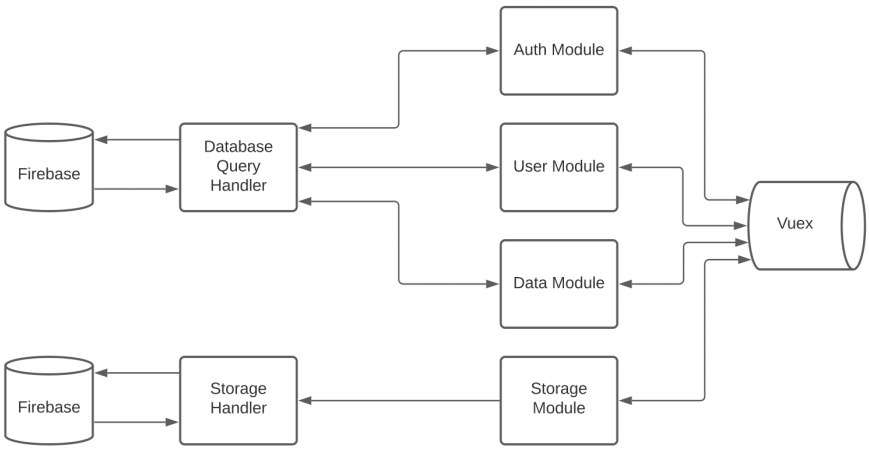
5.1.3 Vuex

Vuex se koristi kao jednostavni cahce podataka, jedino kada će vuex pitai bazu za podatke je ako vuex nema te podatke. Promijene podatak ide u vuex i u bazu podatak. Nisam htio koristiti firebase svugdje po kodu pa da neznam gdje se sta mijenja, to je još jedna dobra stvar oko vuex-a, svaka promijena mora proči koroz funkciju ili modu koji je zadužen za te promijene.

Bilo bi vrlo lagano dodati da se kod citanja podataka stvori callback tako da firebase može pisati podatke u vuex ako su se ti podaci promijenili negdje drugdje, onda bi se u realnom vremenu mogle vidjeti promijene na 2 klijenta, no to nije implementirano jer nisam vidio zašto bi netko to koristio, ali kod je takav da bi se moglo vrlo lagano to napraviti, jer ne treba kopati po kodu gdje se sve mijenjaju podaci.

Vuex ima 4 modula koji su zaduženi za različite podatke, moduli su : autentifikaciju , zadužen za inicijaliziranje svega što ovisi o UID-u i za autentifikaciju, user modul, on se brine o podacima o korisniku, data modul, on se brine o podacima vezanima za elemente na Grid-u i storage module, koji se brine za datoteke na firebase storage

Svaki upit za bazu podatak mora proći kroz Database Query Handler koji se brine da se stvori točan 'path' za podatke unutar firebase baze (spajanje stringova otp.). Storage Handler radi istu stvar samo za podatke na firebase storage bazi, ti podaci su organizirani tako da se nalaze unutar direktorija od korisnika čija je datoteka, direktorij se adresira sa korisnikovim UID-em.

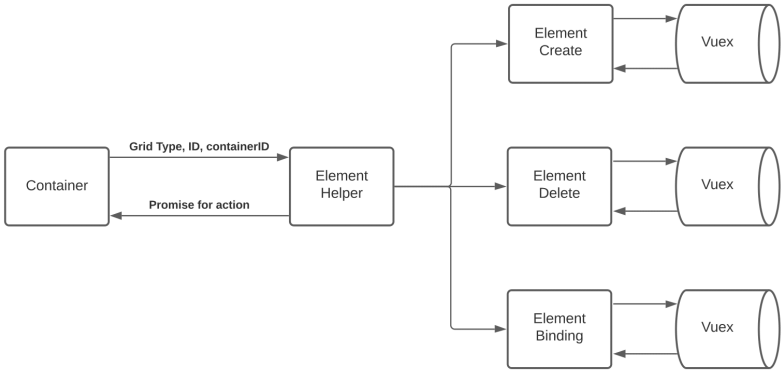


Slika 5.3: Vuex diagram

5.1.4 Element Helper

Svaka funkcija koja nešto radi sa podacima, vezanima za Grid, je smještena u js datoteku ovisno o tipu zadatka koju obavlja. Ako želim stvoriti novi element onda samo moram pozvati funkciju koja se s time bavi, isto i za brisanje i spajanje elemenata. Potrebno je više toga pripremiti (više poziva u vuex, odnosno u bazu) prije brisanja te isto tako i za spajanje i stvaranje elemenata pa sam mislio da bi bilo najbolje da ne kopiram kod i stavio sam ih svaki u svoju datoteku.

Ovaj diagram je samo primjer za container komponentu, isto vrijedi i za group, section i item komponente koje koriste iste ili slične funkcije. Većinom se mora proslijediti ID roditelja i dijeteta te se onda dobije promise nakon kojeg se mogu osviježiti podaci i prikazati promijene.



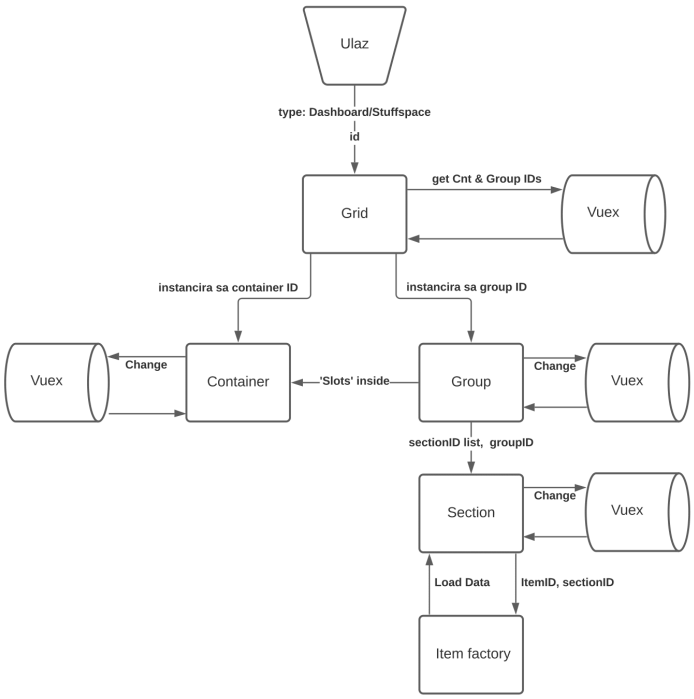
Slika 5.4: Stvaranje elemenata diagram

5.1.5 Grid

Grid je glavna komponenta koja se brine za micanje, širenje te dodavanje elemenata. Prvo što grid radi je čeka dok se zastavica za stanje baze ne aktivira (odnosono dok baza nije aktivna (zapravo dok ne dobimo UID od korisnika)). Nakon što se baza aktivira tek onda može pitati za podatke o trenutnom gridu (Dashboard ili Stuffspace), kako bi mogao instancirati djecu mora dobiti ID za od svakog containera te nakon toga svaki od tih containera u kojem dobije ID od grupe. Nakon što dobije te podatke onda ih tek može prikazati.

Mislim da je bitno napisati da se svi podaci kopiraju tako da svaki element ima lokalnu kopiju tih podataka, ima dosta podataka koji se mijenjaju sve dok nije korisnik zadovoljan s promijenom, (naziv elementa, micanje elementa, itd..) te mi se činilo lakše i spremiti podatke u vuex tek kada je korisnik gotov s promijenom, ne vjerujem da bi aplikacja imala nešto lošije performanse ako bih ažurirao stanje kontinuirano, ali postoji veći overhead od obične promijene varijable.

Djeca dobe podatke o svojim roditeljima, kako bi se mogli pronači u bazi, dobiju još neke podatke, o stilu i sl. U Section elementu sam koristio teleport za prenjiet edit itema u modal, ali mi se nije baš sviđalo pa sam odustao od toga. Neki elementi emitaju signale, ima dosta event handlera, ja mislim da sam ih dobro nazvao.

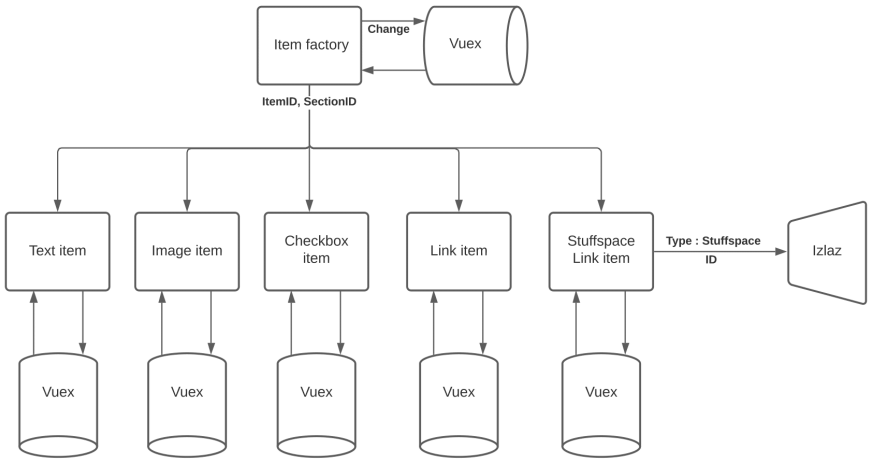


Slika 5.5: Grid diagram

5.1.6 Item Factory

Item Factory komponenta se brine za proizvbodnju itema ovisno o tipu, nažalaost nisam mogao koristiti :is jer dobijam različite signale te šaljem različite propove. Item factorz dobije id itema kojeg treba prikazati, zatim instancira komponentu tog tipa te se ta komponenta brine za sve izmjene tog itema, zbog toga je prikazano na diagramu da komunicira sa vuex-om.

Item-ima moram proslijediti i section ID na kojeg su vezani kako bi se mogli izbrisati (odvojiti). Stuffspace item je poseban po tome što se preko njega otvara stuffspace tako da se u grid pošalje tip tog grida (jer stil ovisi o tipu, i još par stvari) te još u query string mora staviti id, zbog vue-routera.



Slika 5.6: Item Factory diagram

5.2 Poboljšanja i problemi

Najveći problem je bilo vrijeme i razmjer projekta, mislim da možda nije baš bilo pametno uzeti ovo kao projekt, prvenstveno zbog količine podataka koja se mora prikazati na različite načine. Izgleda sitno ali je bilo puno posla napraviti to sve da radi. Dok sam radio na početnim stranicama činilo mi se puno lakše, nakupi se dosta toga.

Problem kod v-for unutar slot-a , pa sam morao ostati na verziji 3.0.0, nisam pogledao dali je rješeno.

Ništa se ne briše

6 Korisničke upute

Nakon što korisnik stvori novi račun, prvo što će vidjeti je Dashboard sa jednim prozorom gdje je link za novi Stuffspace.

Sa lijeve strane korisnik ima izbornik gdje može birati između više Dashboarad, preimenovati ih i izbrisati ih (ako ih je više od jednog).

Pritiskom na ikonu plusa te micanjem miša dok se drži pritisak gumba, može se stvoriti novi container ili stuffspace ovisno gdje se korisnik nalazi.

Ako korisnik ode na bilo koji Stuffspace može stvoriti novi Container, unutar Containera može stvrori novu Grupu te unutura grupe može stvoriti novi Item. Svaki element osim itema se može vući po ekranu tako što se drži lijevi klik nad slikom za povlačenje.