

# **Tiamat**

## **Web aplikacija za upravljanje osobnim podacima**

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

# Sadržaj

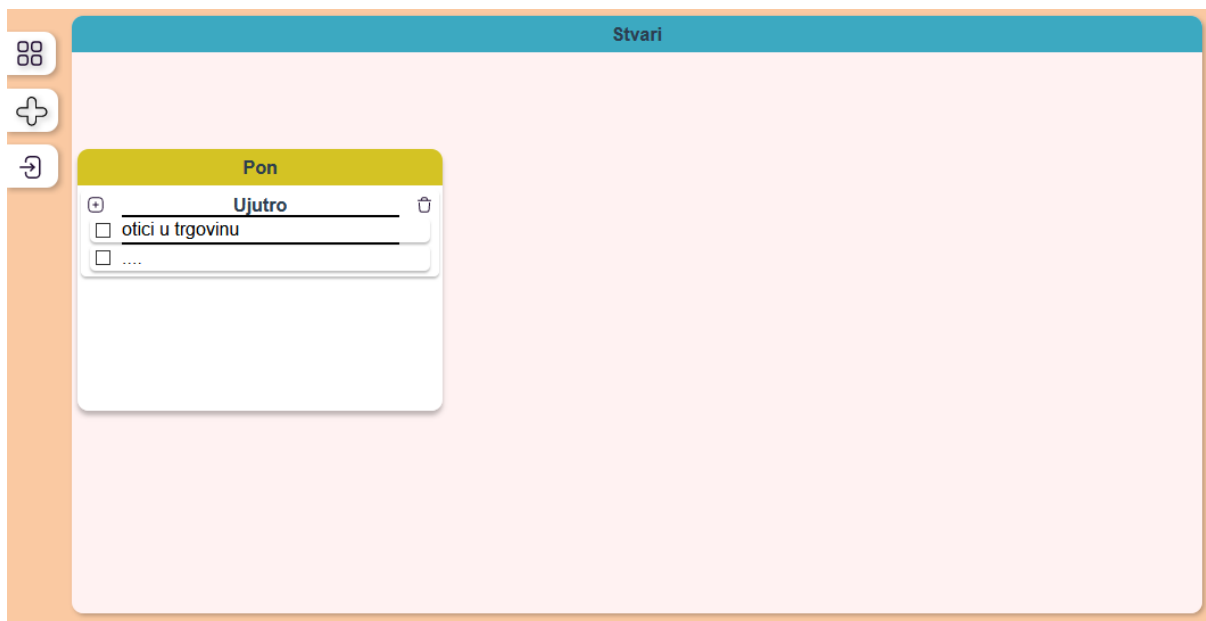
<b>1</b>	<b>Sažetak</b>	<b>1</b>
<b>2</b>	<b>Uvod</b>	<b>2</b>
<b>3</b>	<b>Motivacija</b>	<b>3</b>
<b>4</b>	<b>Razrada funkcionalnosti</b>	<b>4</b>
4.1	Korisnik . . . . .	4
4.2	Dashboard . . . . .	5
4.3	Izbornik . . . . .	5
4.4	Stuffspace . . . . .	6
4.5	Use case model . . . . .	7
4.6	Class model . . . . .	8
4.7	Use case sequence dijagram . . . . .	9
<b>5</b>	<b>Implementacija</b>	<b>10</b>
5.1	Arhitektura . . . . .	10
5.1.1	Podaci . . . . .	12
5.1.2	Login i Signup . . . . .	13
5.1.3	Vuex . . . . .	14
5.1.4	Element Helper . . . . .	15
5.1.5	Grid . . . . .	16
5.1.6	Item Factory . . . . .	17
<b>6</b>	<b>Korisničke upute</b>	<b>18</b>

# 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 koristi firebase te kao glavnu bazu podataka koristi firebase real time database a za spremanje datoteka koristi firebase storage.

Osim samog Vue.js-a koristi 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' korisnik 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 po ekranu, grupa se smije micati samo unutar containera, te iz jednog u drugi. 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 prozorčice 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'.

Aplikacija radi tako što ima više ugniježđenih elemenata, Dashboard služi kao izbornik gdje korisnik stvara i odabire na koji Stuffspace želi otići. Stuffspace pogled služi kao glavni prikaz podatak (bilježaka, zadataka, poveznica i slika) no on sadrži više elemenata. Prvi element koji se može micati je Container, on služi za kategorizaciju podatak te sadrži Grupe koje sadrže popise elemenata. Container se smije micati u 2D prostoru po Stuffspace-u te se unutar njega mogu micati Grupe koje se također mogu micati iz jednog u drugi Container element. Cilj svega toga je dobiti aplikaciju koja pruža korisnicima bolje vizualno iskustvo za rad sa svojim podacima te se želi ostvariti veća razina interakcije između aplikacije i korisnika u nekakvom dinamičkom okruženju.

Unutar grupa možemo imati više popisa Itema koju se stavljene pod Section element kako bi se ostvarila još jedna razina grupacije elemenata.

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 vrijeme, barem pola koda bi bilo drugačije, a neke stvari ja mislim da sam pogodio kako najbolje napraviti, rijetko ali desilo se.

### 3 Motivacija

Ja mislim da bi aplikacija bila vrlo korisna svima koji rade na raznim projektima ili bi im se svidio način na koji aplikacija vizualno grupira razne ideje i zadatke. Aplikacija bi bila sljedeći korak nakon popisa. Htio sam napraviti aplikaciju sa više interakcije između korisnika i same aplikacije.

Korisnici bi bili ljudi koji koriste neke slične aplikacije za bilješke i organizaciju 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 sa prozorima. Aplikacija bi privukla korisnike zbog načina vizualizacije njihovih podataka, toj je taj glavni *feature* aplikacije.

S	W
- vizualizacija podataka - veća 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

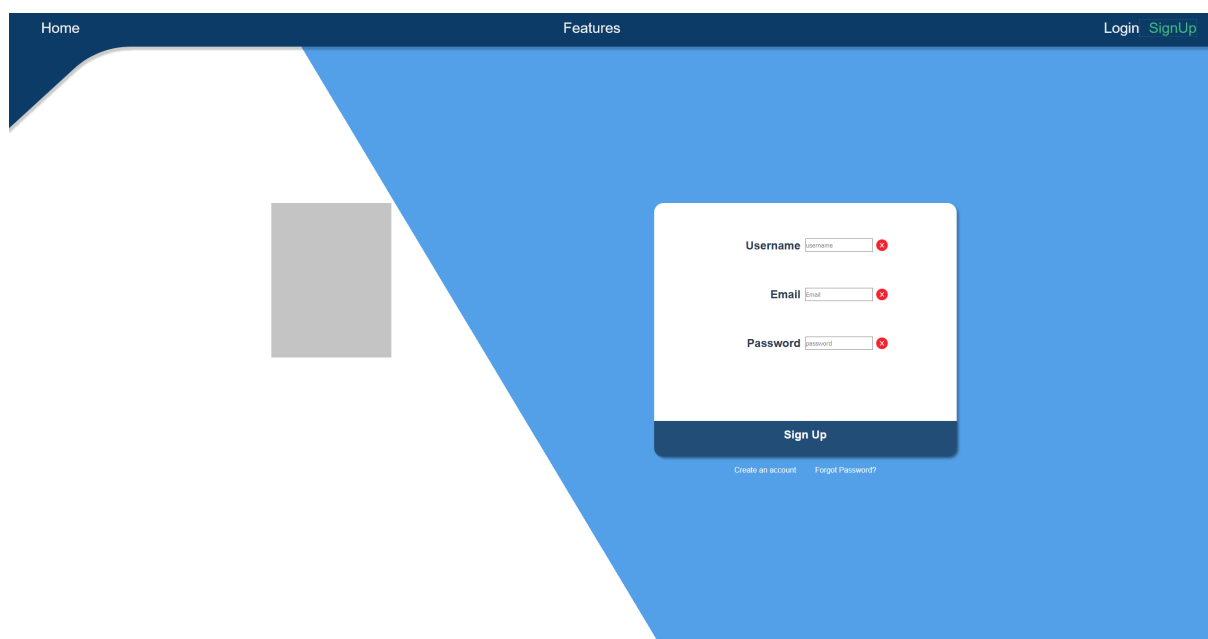
Prednost ove aplikacije je prvenstveno vizualizacija podataka, i bolja vizualizacija kategorija podatak.

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 ako je maleno sučelje.

## 4 Razrada funkcionalnosti

### 4.1 Korisnik

Na ovoj stranici korisnik može napraviti svoj račun, podaci se provjeravaju dali su točnog formata, u slučaju pogrešnog unosa pojavi se tekst sa greškom. Nakon što se stvori novi račun postavlja se baza podataka tako da se može nešto prikazati.



The screenshot displays a web application's registration page. At the top, a dark blue navigation bar contains the links 'Home', 'Features', 'Login', and 'SignUp'. The main content area has a blue background with a white diagonal graphic on the left. A central white registration form is shown, featuring three input fields: 'Username', 'Email', and 'Password'. Each field is followed by a red error icon, indicating validation failures. Below the fields is a dark blue 'Sign Up' button. At the bottom of the form, there are two links: 'Create an account' and 'Forgot Password?'. A gray square placeholder is visible on the left side of the page.

Slika 4.1: SignUp stranica

## 4.2 Dashboard

Dashboard stranica služi za stvaranje Stuffspace link elemenata koji prosljede korisnika na određeni Stuffspace, naziv se može promijeniti te se također mogu micati po ekranu i postaviti na željeno mjesto. sa lijeve strane se nalazi izbornik, više o njemu kasnije.



Slika 4.2: Dashboard stranica

## 4.3 Izbornik

Koristi se isti izbornik na Dashbord stranici kao i na Stuffspace stranici, prva ikona je za odabir Dashboarda gdje korisnik može prijeći na neki drugi ili izbrisati ili stvoriti novi. U sredini se nalazi područje od kuda korisnik može, klikom sa mišem i povlačenjem, stvoriti novi element te ga smjestiti na željeno mjesto. Ovisno o tipu, dali je Dashboard ili Stuffspace, stvoriti će se drugačiji elementi, ili novi Stuffspace ili novi Container element. Zadnja ikona služi za odjavu korisnika iz aplikacije te ga vrati na index stranicu.

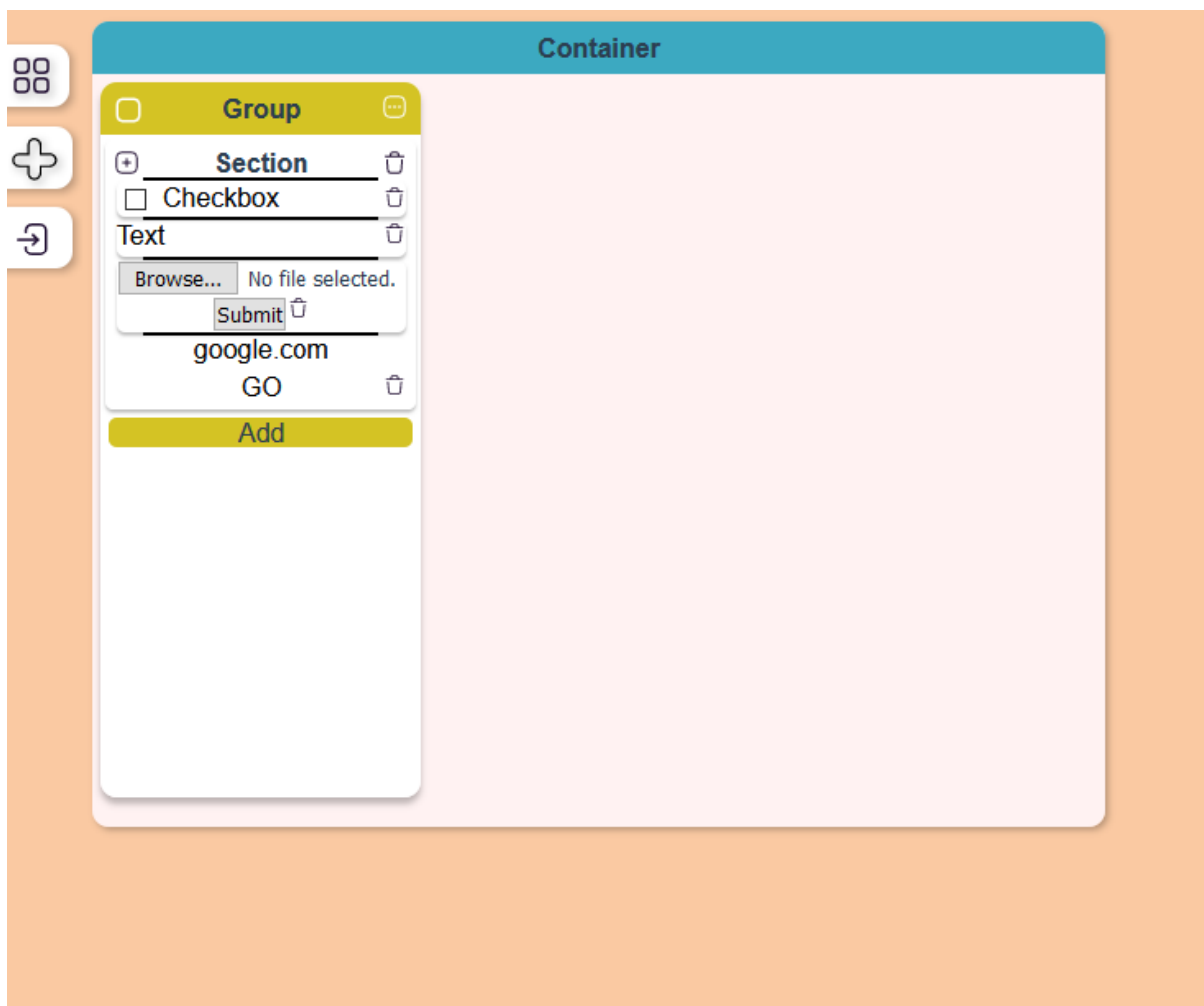


Slika 4.3: Izbornik

## 4.4 Stuffspace

Na slici iznad je prikazan primjer Stuffspace prozora. On sadrži elemente koje sam nazvao po njihovom tipu. Vidi se da se unutar Container elementa nalazi Group element koji sadrži Section kao popis Itema. Ti nazivi se mogu promijeniti.

Container i Group elementi imaju izbornik u gornjem desnom kutu, nakon što korisnik klikne na taj izbornik on se otvori te se sadržaj elementa zamijeni s tim izbornikom. Ta dva elementa također imaju ikonu za micanje, nalazi se u gornjem lijevom kutu, dok se u donjem desnom kutu nalazi područje gdje korisnik može promijeniti veličinu tog elementa.

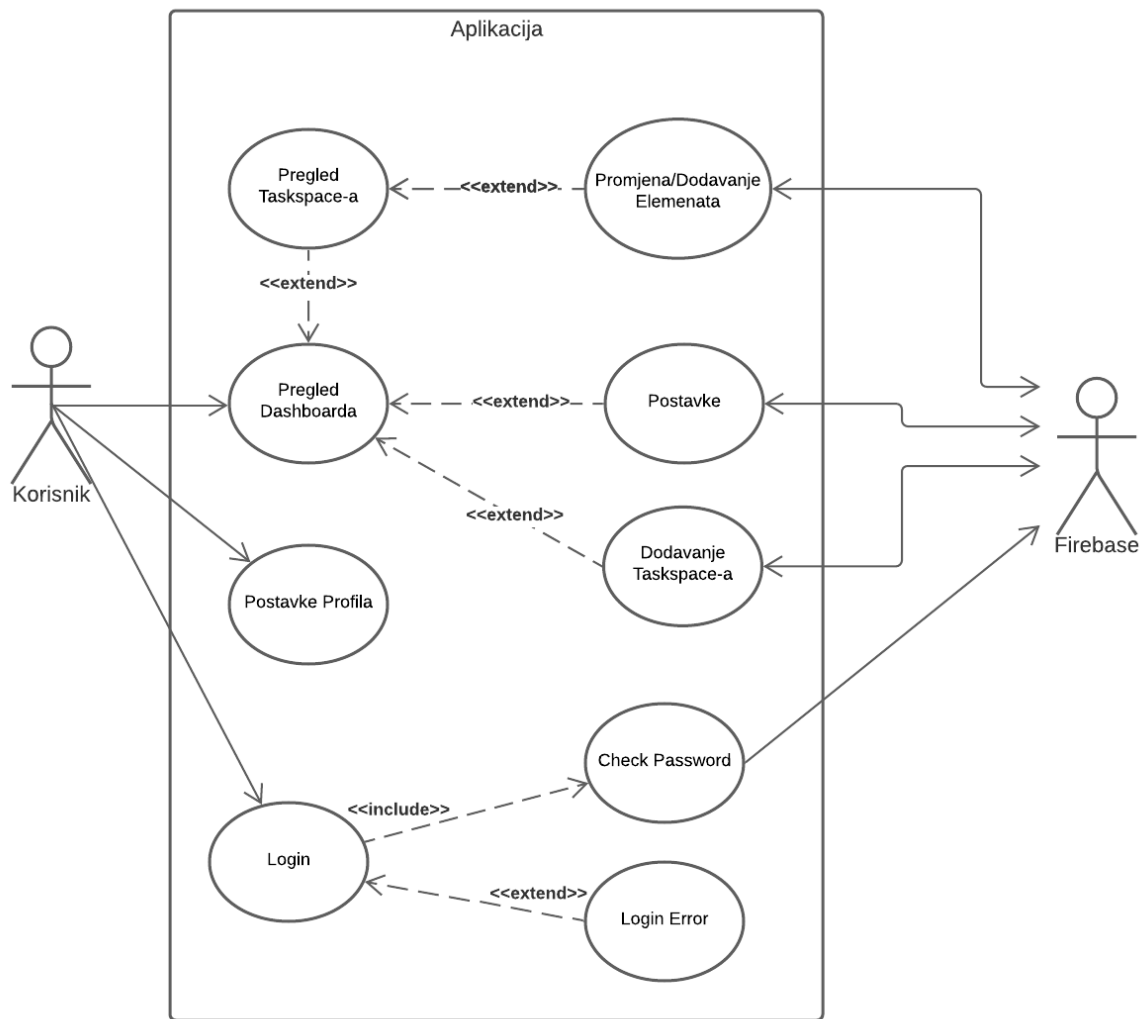


Slika 4.4: Stuffspace primjer



## 4.5 Use case model

Nakon što korisnik uđe u svoj račun on može pregledavati razne elemente web aplikacije. Vanjski sustav je firebase koji se koristi za spremanje podataka i autorizaciju korisnika.

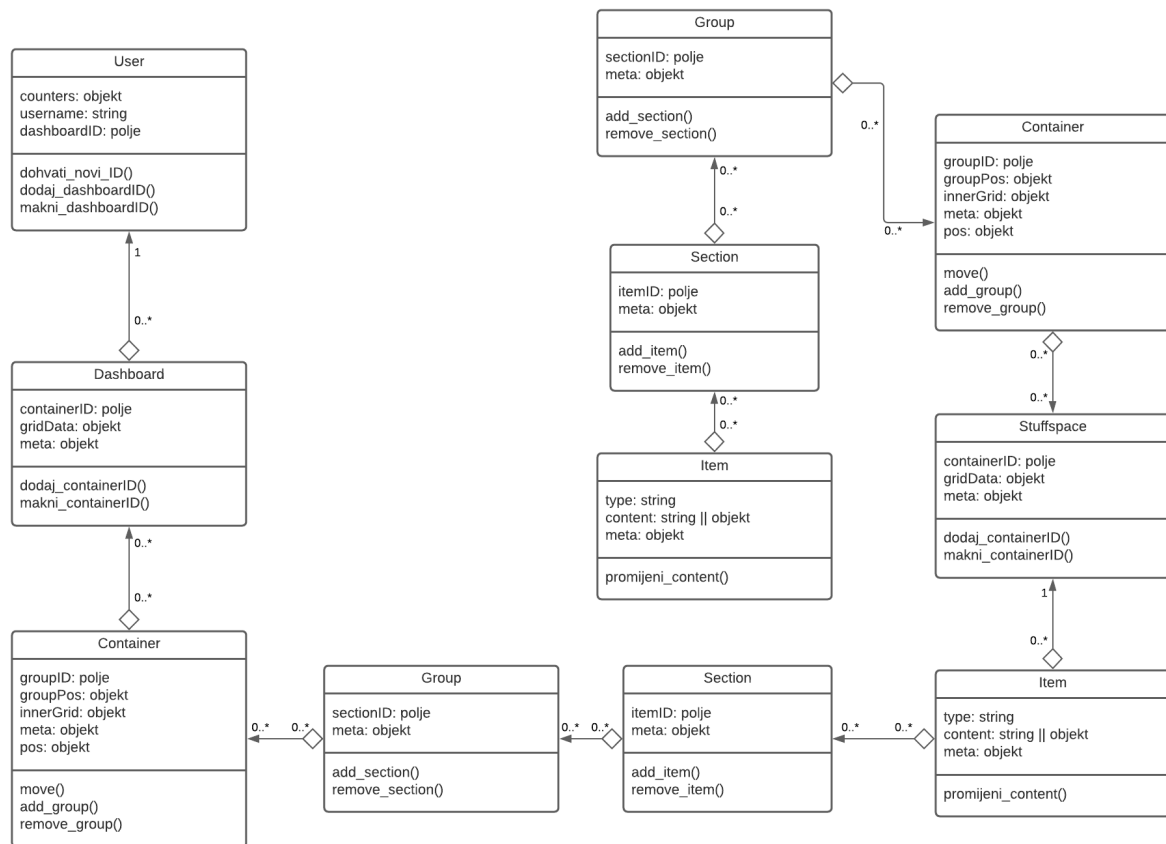


Slika 4.5: Use case model

## 4.6 Class model

Svako dijete ovisi o roditelju, User može imati 1 ili više Dashboard-a, Dashboard može imati nula 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.

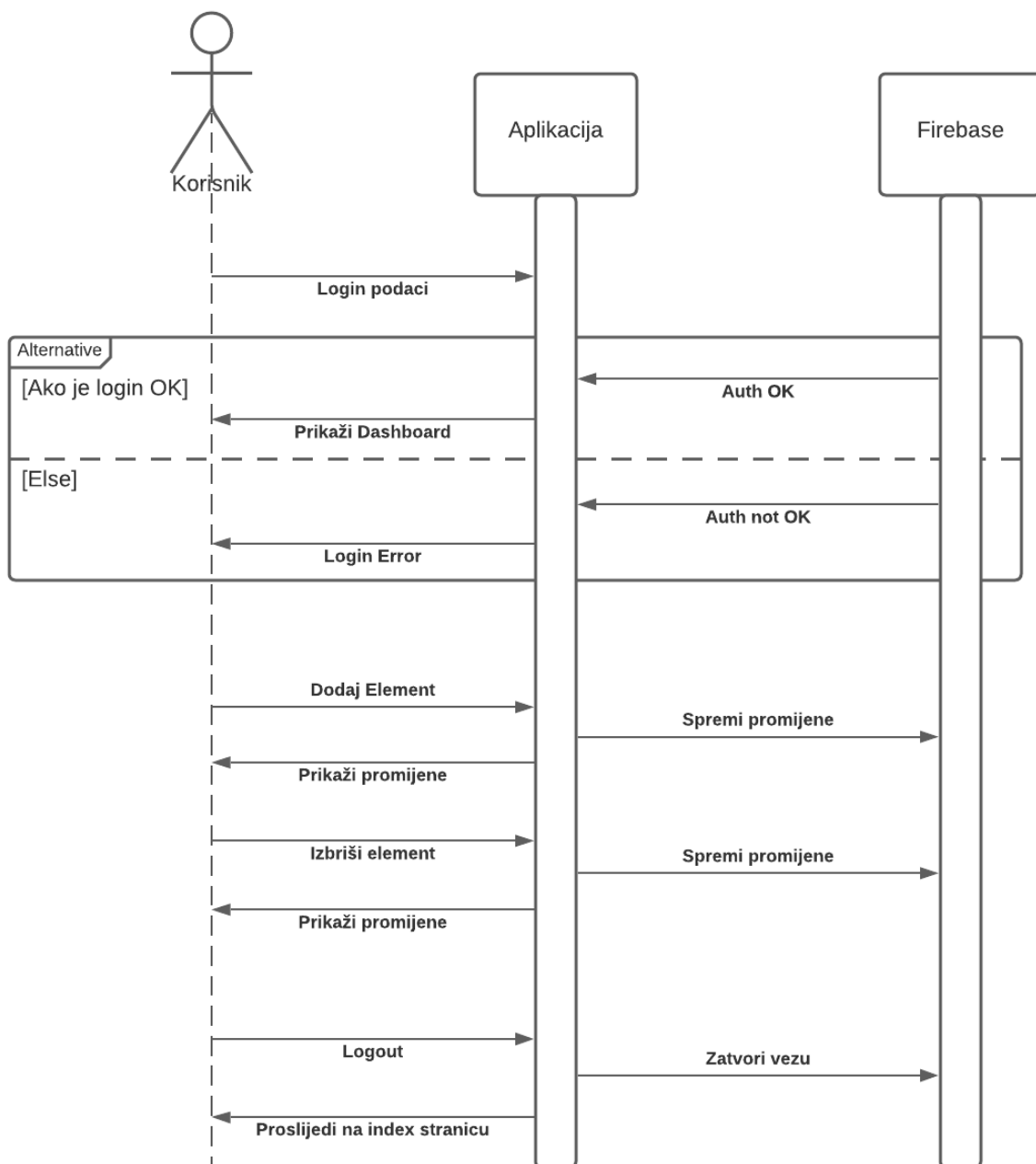
Nema ništa što sprječava korisnika da 2 različita Dashboarda imaju istog Containera (osim što nije implementirano) te također da Dashboard ima više Containera (to je implementirano). Meta se koristi za spremanje naziva elementa i još par podataka o njima.



Slika 4.6: Class model

## 4.7 Use case sequence diagram

Svi scenariji korištenja aplikacije su zapravo isti, nema puno razlike između stvaranja novog Itema ili bilo kojeg drugog elementa u aplikaciji. Zbog prirode aplikacije ne znam dali bi ovaj dijagram pomogao u razumijevanju korištenja aplikacije. Kao što se vidi na dijagramu, nakon što korisnik pristupi svom računu sve se svodi na primanje i slanje podataka. Kada korisnik izmjeni neki element onda se ta promjena pošalje u bazi te se te promjene prikažu korisniku. Aplikacija nema neke posebne načine korištenja, npr. dodaj u košaricu i kupi, sve se svodi na izmjenu elemenata, stoga ovaj dijagram nije previše kompliciran.



Slika 4.7: Use case sequence diagram

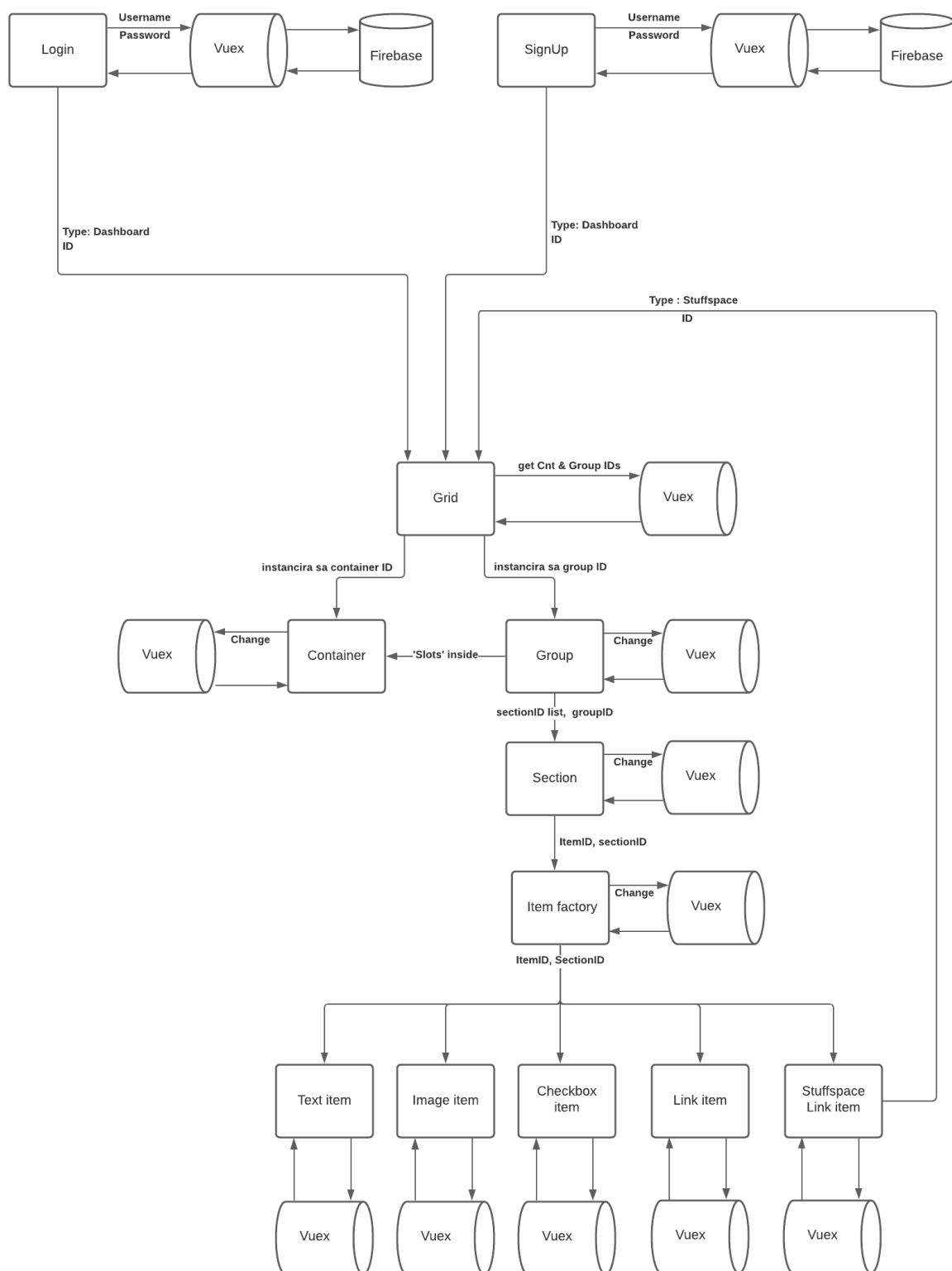
## 5 Implementacija

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.

### 5.1 Arhitektura

Svaka komponenta poput containera, groupe, sectiona i itema 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 osviježi stranicu onda se firebase mora ponovno inicijalizirati prije nego što se mogu dohvatiti podaci.

Nisam koristio klasne djiagrame za prikaz implementacije, imam preko 40 vue datoteka i preko 30 js datoteka, sa sveukupno više od 7500 linija koda, (+/- par datoteka jer neke ne koristim ili su starije verzije) pa tako da nisam mislio da bi pravi klasni dijagrami ikome pomogli da nabrzinu razumije rad aplikacije, niti se meni da prolaziti kroz kod i zapisati sve metode te kako sve točno komunicira. Pravokutnik predstavlja klasu, vodoravni valjak predstavlja vuex, okomiti valjak predstavlja firebase i trapezoid predstavlja ulaz ili izlaz iz jednog dijela u drugi.

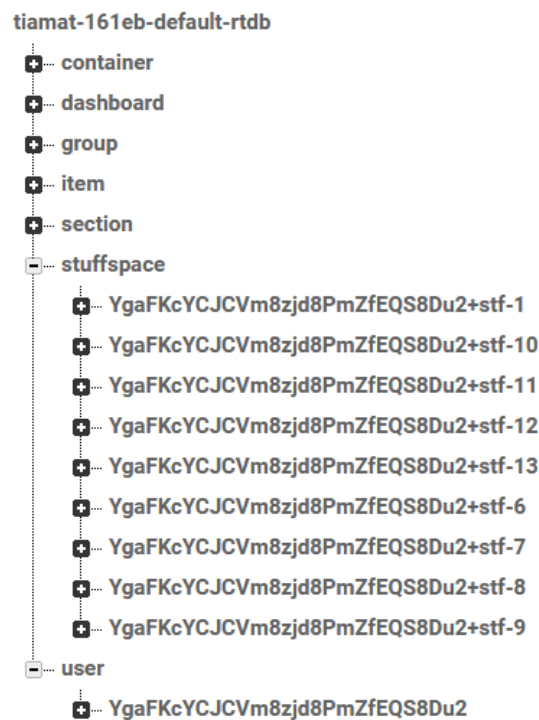


Slika 5.1: Dijagram komponenti aplikacije

### 5.1.1 Podaci

Firebase ima jedan hook za promijene u autentifikaciji, u njemu se inicijalizira baza, to je vrlo bitno 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.



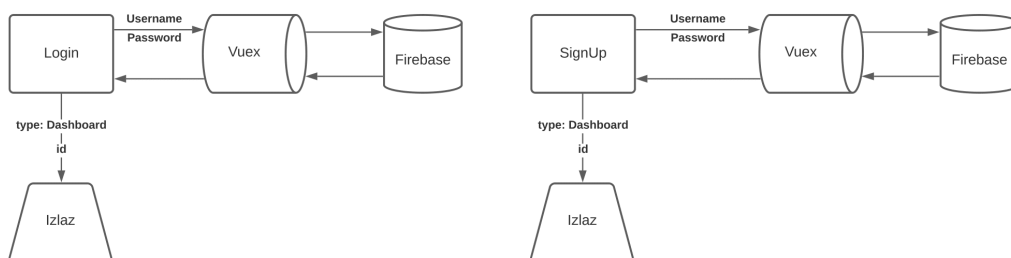
Slika 5.2: Struktura podataka unutar baze

Pravila unutar firebasea su takva da se podaci smiju čitati i pisati samo ako naziv "direktorija" sadrži korisnikov UID. Također bi tako radilo i na firebase Storage gdje korisnik smije pristupiti datotekama samo ako se nalaze unutar direktorija kojemu je naziv korisnikov UID.

### 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, korisnika 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 5.3: Login i Signup dijagram

### 5.1.3 Vuex

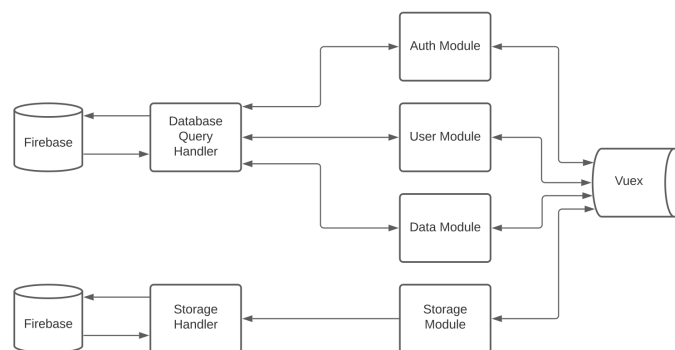
Vuex se koristi kao jednostavni cache podataka, jedino kada će vuex pitati bazu za podatke je ako vuex nema te podatke. Promijene podatak ide u vuex i u bazu podatak. Nisam htio koristiti firebase svugdje po kodu pa da ne znam gdje se što mijenja, to je još jedna dobra stvar oko vuex-a, svaka promjena mora proći kroz funkciju ili modu koji je zadužen za te promijene.

Bilo bi vrlo lagano dodati da se kod čitanja 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.

- Modul za 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
- 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.



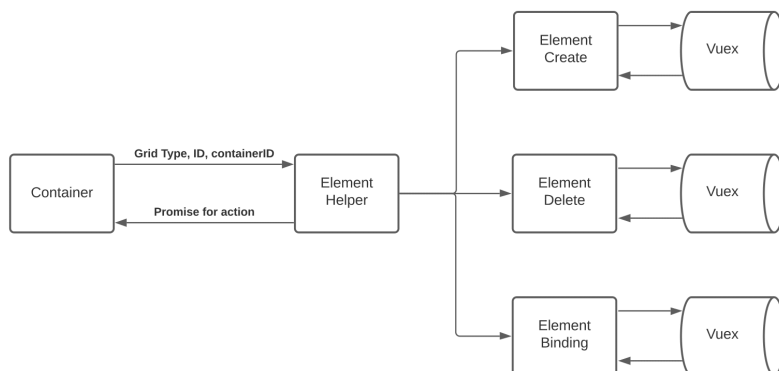
Slika 5.4: Vuex dijagram



### 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 djiagram 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.



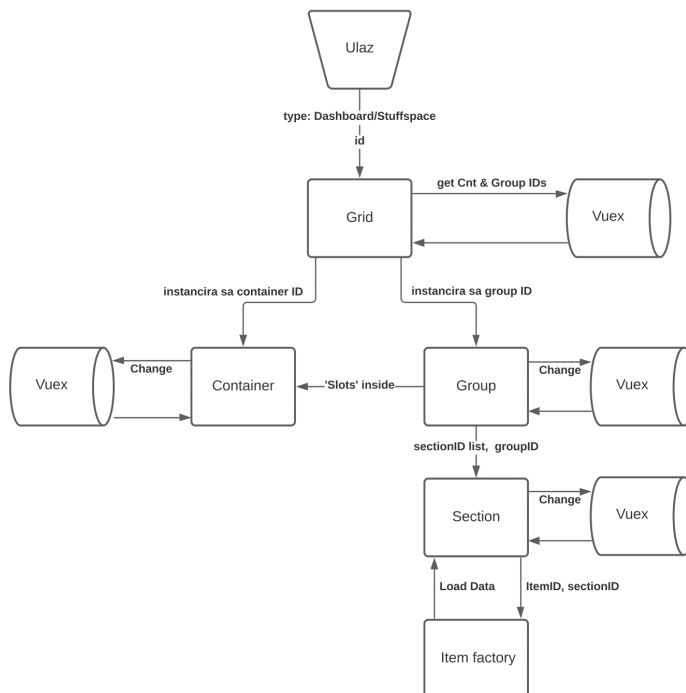
Slika 5.5: Stvaranje elemenata djiagram

### 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 (odnosno 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 svakog containera te nakon toga mora dobiti svaki od tih containera u kojem se nalazi polje s ID-em od grupa koje oni sadrže. 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 spremiti podatke u vuex tek kada je korisnik gotov s promijenom, ne vjerujem da bi aplikacija imala nešto lošije performanse ako bih ažurirao stanje kontinuirano.

Djeca dobe podatke o svojim roditeljima kako bi se mogla pronaći u bazi, dobiju još neke podatke, o stilu i sl. U Section elementu sam koristio teleport za prenijeti edit itema u modal, ali mi se nije baš svidalo pa sam odustao od toga. Neki elementi emitaju signale, ima dosta event handlera.

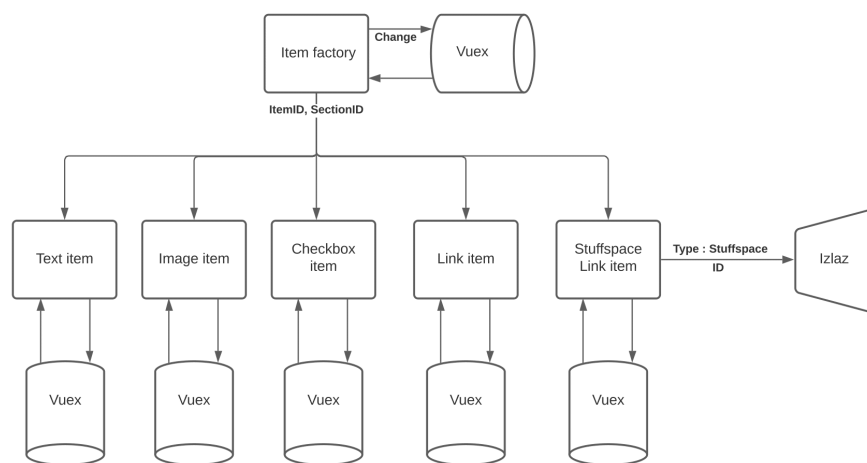


Slika 5.6: Grid djiagram

### 5.1.6 Item Factory

Item Factory komponenta se brine za proizvodnju itema ovisno o tipu. Item factory 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 djiagramu 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) te još u query string mora staviti ID, zbog vue-routera kako bi se korisnik lakše mogao vratiti na prijašnje poglede.



Slika 5.7: Item Factory dijagram

Podaci u bazi o Itemu su njegov tip i sadržaj, ovisno o tipu tog itema, njegov sadržaj se prikazuje na različite načine, kao ID za sliku, tekst ili link. Postoji event koji ide sve od Itema pa do Grid-a, s njim se proslijedi promise dobijen od Vuex-a (nakon neke izmjene), kada promise poprimi vrijednost onda se ponovno učitaju podaci. To se koristi zbog toga što svaki element ima svoje lokalne kopije podatak kako se ne bi baza ažurirala kontinuirano nego tek kada je korisnik završio sa nekom izmjenom. Taj signal onda služi za učitavanje podataka iz vuex-a nakon što neki element promijeni te podatke.

## 6 Korisničke upute

1. Nakon što korisnik stvori novi račun, prvo što će vidjeti je Dashboard sa jednim prozorom gdje je link za novi Stuffspace.
2. Sa lijeve strane korisnik ima izbornik gdje može birati između više Dashboarda, preimenovati ih i izbrisati ih (ako ih je više od jednog).
3. Pritiskom na ikonu plusa te micanjem miša dok se drži pritisak gumba, može se stvoriti novi container ili stuffspace ovisno gdje se korisnik nalazi.
4. Ako korisnik ode na bilo koji Stuffspace može stvoriti novi Container, unutar Containera može stvoriti novu Grupu te unutar grupe može stvoriti novi Item. Svaki element osim itema se može vući po ekranu tako što se drži lijevi klik nad slikom za povlačenje.
5. Svaki naziv elementa se može promijeniti klikom na taj tekst. Korisnik zatim mora kliknuti izvan tog elementa kako bi se promijene spremile.
6. Unutar Section elementa, klikom na plus, otvori se izbornik za dodavanje Itema. Svaki element osim Itema ima svoj izbornik s jednom ili više opcija.
7. Elementi se brišu tako što korisnik klikne na sliku kante za smeće. Svaki element unutar obrisano elementa će se također izbrisati.
8. Text item se koristi tako što korisnik klikom na taj element može manipulirati tekстом tog elementa
9. Link item se koristi tako što korisnik klikne na naziv domene nakon čega će moći unijeti novi link, a klikom na "GO" otvoriti će se novi prozor s tom stranicom
10. Checkbox item je sličan text item-u samo što ima potvrdni okvir
11. Image item ima gumb za odabir slike i za učitavanje te slike. Slika će se prikazati nakon što se učita te će se također spremirati u bazu.
12. Container i Group elementi se mogu proširiti tako da se držanjem klika u donjem desnom kutu vuče miš. Elementi će se automatski proširiti ako ima prostora za njih
13. Dok se elementi vuku po ekranu, biti će označeno mjesto na koje korisnik može smjestiti taj element. U slučaju da nema mjesta za taj element a korisnik ga pusti onda se on vrati na početno mjesto