APLIKACIJE IN INFORMACIJSKI SISTEMI

Mentor: Gašper Strniša, prof.

Kandidat: Miha Meglič

Kranj, april 2020

ZAHVALA

Zahvaljujem se mentorju, prof. Gašperju Strniši, za konzultacije pri pisanju seminarske naloge in prof. Dragici Debeljak za lektoriranje.
Prav tako se zahvaljujem očetu, Gregorju Megliču, za testiranje programske rešitve.

POVZETEK

Upravljanje z neko vrsto zaloge je vsakdanje opravilo, o katerem ne razmišljamo, dokler ga ne opravljamo na večjem obsegu – npr. zaloga v hladilniku je veliko bolj preprosta kot zaloga v podjetju ali industrijskem obratu. Zato se v primerih večjega obsega običajno obrnemo k tehnologiji.

Ustvaril sem spletno aplikacijo za vodenje skladišč/zaloge. V ozadju sistema je NoSQL podatkovna baza Firestore, vse skupaj pa je gostovano na Google Firebase. Druge uporabljene tehnologije so: Bootstrap, Webpack ter spletni jeziki (HTML, CSS in JavaScript).

Najprej določimo strukturo podatkov v podatkovni bazi, nato ustvarimo grafični vmesnik aplikacije. Nazadnje pa vse skupaj povežemo z logiko za procesiranje in upravljanje s podatki, nekaj le-teh se izvaja lokalno, nekaj pa na strežniku.

Za zaključek potrdimo uspešnost naše rešitve z beta testiranjem, da zagotovimo, da v kodi ni hujših napak ali varnostnih lukenj. Pri tem pa odkrijemo tudi nekaj pomanjkljivosti ter možnosti za nadaljnji razvoj.

KLJUČNE BESEDE:

- sistem upravljanja zaloge
- logistika
- Firebase
- spletna aplikacija

ABSTRACT

Managing inventory is a part of everyday life that we do not think about very much, while the scope of it remains small – e.g. the inventory of a fridge is much easier to handle than the inventory of an industrial operation. That is why we seek the help of technology in cases of a broader scope.

I created a web application for managing inventories/stock. The systems' backbone is Firestore, a NoSQL database, and everything is hosted on Google Firebase. Other technologies used are: Bootstrap, Webpack and web languages (HTML, CSS and JavaScript).

First, we determine how to structure the data within the database, next we create a graphical user interface and lastly we connect everything together with data processing and managing logic, some of which is executed locally, and some remotely on the server.

To wrap it all up, we validate our solution with beta testing, to ensure there are no severe errors or security leaks in our code. Doing that, we discover a few shortcomings and options for further development.

KEYWORDS:

- Inventory management system
- Logistics
- Firebase
- Web application

KAZALO

1. UVOD

Skoraj v vsaki vrsti industrije (ter celo doma) se srečujemo s tako ali drugačno obliko zaloge, npr. hrana v shrambi, gradbeni materiali v lopi ali svinčnik in papir na mizi.
V večini primerov moramo beležiti vhodne, izhodne in interne transakcije zaloge, da vemo, kdaj nam nečesa primanjkuje in kje se stvari nahajajo.
Ti podatki pa, posebej v svetu industrije, hitro prerastejo preprosto zapisovanje na papir oz. skladiščni karton. Poleg tega pa je iskanje in preverjanje podatkov na papirnih zapisih skorajda nemogoče pri večjih količinah arhivskih listin.

Podobno težavo sem opazil pri skladiščenju elektronskih komponent, saj je tako za proizvodnjo kot za hobi projekte potrebna zaloga materiala, da lahko proces teče nemoteno. Še večji problem pri domači zalogi pa je lokacija kosov. Proizvodnja ima običajno določen večji prostor, kjer se skladiščijo vsi potrebni kosi, doma pa so to večkrat kot ne samo sanje in hitro imamo več različnih prostorov z različnimi škatlicami ter omarami, ki so polne raznoraznih materialov, na katere hitro pozabimo. Zato je sitem za vodenje zaloge oz. IMS odlična rešitev.

Še vedno imamo pri vpisovanju kosov in zaloge ter prenašanju iz skladišča v skladišče precej dela, ampak ko želimo preveriti količino zaloge ali celo lokacijo zaloge, je preprost spletni vmesnik precej hitrejša metoda kot iskanje po hiši ter štetje, sploh ker se električne komponente in druge materiale običajno naroča v večjih količinah.

1.1. Logistika

Preden začnemo z opredelitvijo problema, pa je precej ključno, da razumemo termine oz. pojme, ki jih bomo uporabljali.

IDENT je šifra oz. identifikator skupine enakih kosov.
ŠIFRANT IDENTOV je tabela, v kateri hranimo idente in njihove parametre oz. vrednosti.
VPIS ZALOGE je dodajanje kosov v skladišče.
PRENOS ZALOGE je prenašanje kosov iz enega v drugo skladišče.
ODPIS ZALOGE je odstranjevanje kosov iz skladišča.

Te pojme bomo uporabljali skozi celoten proces načrtovanja in izdelave ter kasneje tudi v uporabniškem vmesniku, zato je ključno, da smo z njimi seznanjeni.

2. PREDSTAVITEV PROBLEMSKEGA STANJA

Domače skladišče običajno sestavlja več omar ter v primeru hrambe komponent tudi kar precej takih ali drugačnih predalčnikov, posod in škatel (Slika 1).

Podjetja, kot že omenjeno, ta problem rešujejo z informacijskimi sistemi (običajno neke vrste ERP), ki pa so večinoma neprimerni za domačo uporabo, saj so optimizirani za delo v industrijskem okolju in obsegajo veliko količino nastavitev, ki bi domačega uporabnika lahko hitro zmedla. Druga težava z njimi pa je, da pogosto zahtevajo strežnik, kjer se gostijo podatki, ki jih moramo sami priskrbeti ali pa plačevati gostovanje ali pa podatke shranjujejo samo na lokalni napravi, kar pomeni, da bi uporabnik moral uporabljati eno samo napravo ali pa podatke shranjevati na prenosni medij.

To je že kar precej razlogov proti uporabi poslovnega informacijskega sistema doma, še preden sploh upoštevamo vratolomne cene takih programov.

Slika 1: Primer skladišča komponent

3. NAČRTOVANJE REŠITVE

Kot smo prej opredelili, je ena glavnih težav centralizacija in sinhronizacija sistema, da na njem lahko dela več uporabnikov istočasno in z različnih lokacij. To pomeni, da je spletno okolje za naš namen optimalno, saj reši obe težavi ter nam omogoči, da za hrbtenico aplikacije uporabimo spletne platforme – specifično Firebase.

Sedaj bomo izdelavo aplikacije razdelili na nekaj faz.

Prva faza bo priprava delovnega okolja, v katerem bomo pisali in upravljali s kodo, kar bo vključevalo implementacijo orodja Webpack s prevajalnikom Babel. Webpack bo poskrbel za združevanje kode iz več JS skript v eno, kar ni podprto v starejših izdajah JS. Pri tem pa bo Babel prevedel JS v starejšo verzijo sintakse, kar pomeni, da bo stran delovala tudi v večini starejših brskalnikov. Poleg tega pa bo Babel tudi "minificiral[1]" in optimiziral JS kodo.

Druga faza pa bo določitev struktur, v katerih bomo hranili podatke. Čeprav je Firestore nestrukturirana podatkovna baza, moramo podatke vseeno urediti na nek logični način, da bo delo z njimi lažje. Za olajšanje postopka bomo v začetku vnesli nekaj podatkov, da presodimo, katere informacije so relevantne, in jih želimo zapisovati, ter katere ne. Ker imamo opravka predvsem z elektronskimi komponentami, bomo shranjevali veliko raznolikih parametrov, kjer nam bo bolj koristila NoSQL baza v primerjavi z SQL bazo.

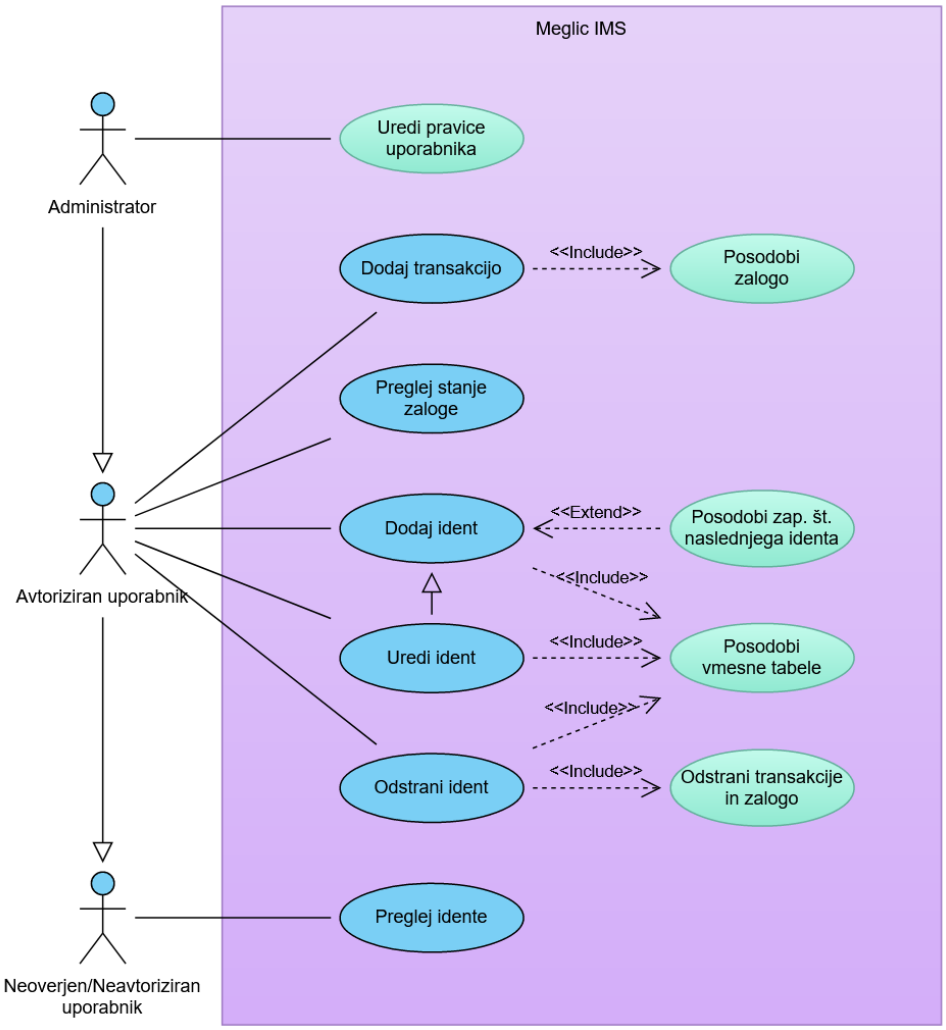
Tretja faza bo izdelava grafičnega vmesnika, ki bo sposoben na učinkovit način prikazati in iskati podatke ter ga dopolniti z načini vnosa in manipulacije podatkov.

Četrta in zadnja razvojna faza pa je implementacija varnosti v obliki prijave in registracije v osebni račun, ki diktira nivo dostopa uporabnika v podatkovnem sistemu (Firestore).

Za zaključek pa sledi še testiranje izdelka. Ker se alfa testiranje dogaja na (skoraj) vsakem koraku razvoja, bomo takoj prešli na beta testiranje. Sistem je namenjen za domačo uporabo, zato ga bo testiral moj oče, ki ga bo poleg mene tudi uporabljal.

Da pa si bomo vse te korake lažje predstavljali, je uporabno orodje UML diagram primerov uporabe (Slika 2), iz katerega je razvidno, kako so povezane različne operacije in kdo jih (lahko) izvaja.

[1] proces odstranjevanja nepotrebnih ali redundantnih podatkov



Slika 2: UML diagram: primeri uporabe

Zaradi lažje berljivosti sem v diagramu za različno mesto izvajanja uporabil različne barve. Funkcije oz. primeri uporabe zelene barve se izvajajo v storitvah v ozadju (Firebase funkcije), modro obarvani primeri uporabe pa se izvajajo preko JavaScript lokalno v brskalniku in prenašajo podatke med uporabnikom in podatkovno bazo Firestore.

Lepo je razvidno tudi, da imamo tri nivoje dostopa, in sicer:

- neoverjeni ali neavtorizirani uporabnik → uporabnik, ki ali ni prijavljen ali ga administrator še ni avtoriziral, lahko samo pregleda idente v sistemu;
- avtorizirani uporabnik → uporabnik, ki je prijavljen in avtoriziran, lahko dodaja, ureja in odstranjuje idente ter pregleduje zalogo in dodaja transakcije;
- administrator → uporabnik, ki je prijavljen in ima status administratorja, lahko avtorizira ali dodeli administrativne pravice drugim uporabnikom.

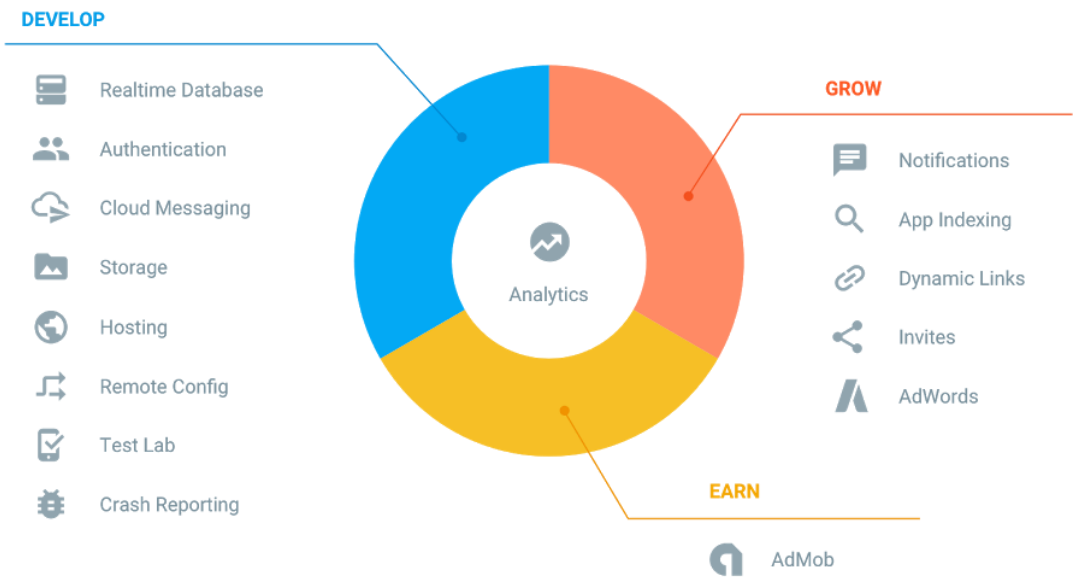
Administrativne pravice v sami aplikaciji nimajo velike vloge, poleg avtorizacije drugih uporabnikov, ampak omogočajo uporabniku neomejen dostop do podatkovne baze preko ukazne vrstice.

4. UPORABLJENE TEHNOLOGIJE

4.1. Google Firebase

Google Firebase je platforma za razvijanje mobilnih in spletnih aplikacij, ki razvijalcem ponuja širok set orodij in storitev. Le-te se delijo v štiri kategorije (Slika 3): **razvoj**, **kakovost**, **analitika** in **rast**. Kategorija razvoj je namenjena delu z vsemi podprtimi platformami, medtem ko so preostale tri predvsem usmerjene v razvijanje mobilnih aplikacij.

Kategoriji razvoj in kakovost ponujata orodja, ki pokrijejo zaledni sistem (tj. sistem, ki upravlja s surovimi podatki in je neviden uporabniku) ter distribucijo aplikacije, medtem ko analitika in rast pokrivata statistične analize občinstva in aktivnosti v aplikaciji ter se močno povezuje z Google Analytics.



Slika 3: Grafični prikaz storitev Firebase (Google, 2020)

Gradili bomo spletno aplikacijo, zato nas zanima samo razvojni del Firebase paketa, ki pokriva overitev uporabnika, podatkovne baze, shrambo datotek, spletno gostovanje in funkcije v oblaku.

Velik del upravljanja z razvojnim delom platforme pa se seveda dogaja na lokalni napravi, kjer razvijamo kodo in pripravljamo funkcije. Firebase zato ponuja CLI (znakovni uporabniški vmesnik), ki omogoča overjen in zanesljiv prenos podatkov med Firebase CDN (omrežje za distribucijo vsebin) in lokalno napravo. Poleg tega Firebase CLI omogoča tudi testiranje aplikacije na lokalnem sistemu.

Velika prednost uporabe platforme Firebase je močna spletna aplikacija, ki omogoča vpogled v stanje aplikacije, ter v primeru razvojne kategorije pregled, vnos in manipulacijo podatkov. Firebase svojo aplikacijo imenuje preprosto spletna konzola.

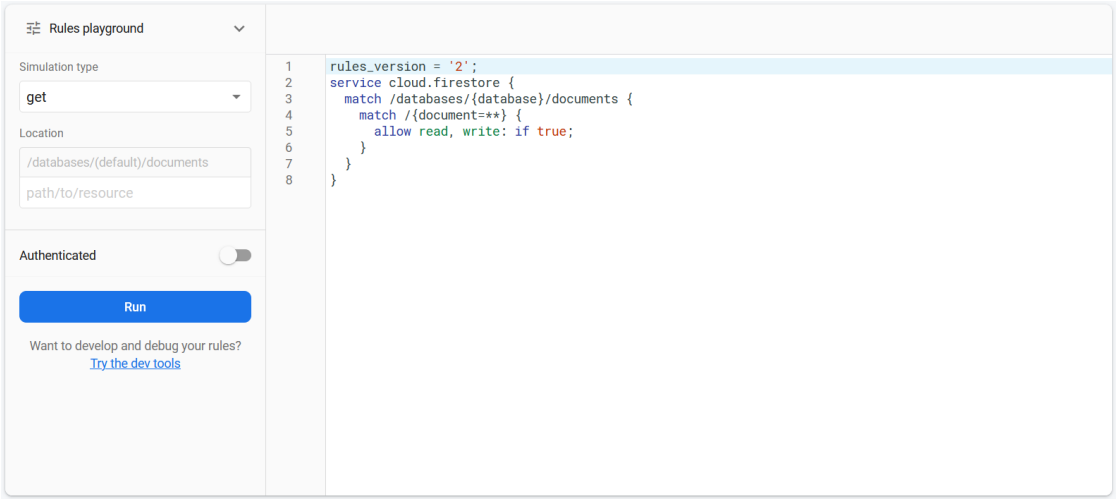
4.1.1. Overjanje uporabnika

Prva in verjetno najpomembnejša storitev za moderne spletne in mobilne aplikacije je overjanje identitete, ki uporabniku zagotovi, da so njegovi podatki varni in dosegljivi samo njemu oz. izbrani skupini uporabnikov.

Firebase nam za overjanje identitete ponuja sledeče opcije:

- klasična overitev z e-mailom in geslom,
- overitev preko telefona (SMS),
- overitev preko spletnih aplikacij, kot so Google, GitHub, Twitter, Microsoft …
- ter prijava anonimnega uporabnika oz. gosta.

Na podlagi overitvenega procesa lahko razvijalec nato določi pravice dostopa do podatkov (Slika 4). Firebase v modulih podatkovna baza in shramba ponuja nastavitev pravil za vsako od teh storitev posebej. Nastavitve omogočajo tudi takojšnje testiranje dostopa (Slika 4 – leva stran) tako neprijavljenega kot overjenega uporabnika.



Slika 4: Firebase pravila dostopa (Google, 2020)

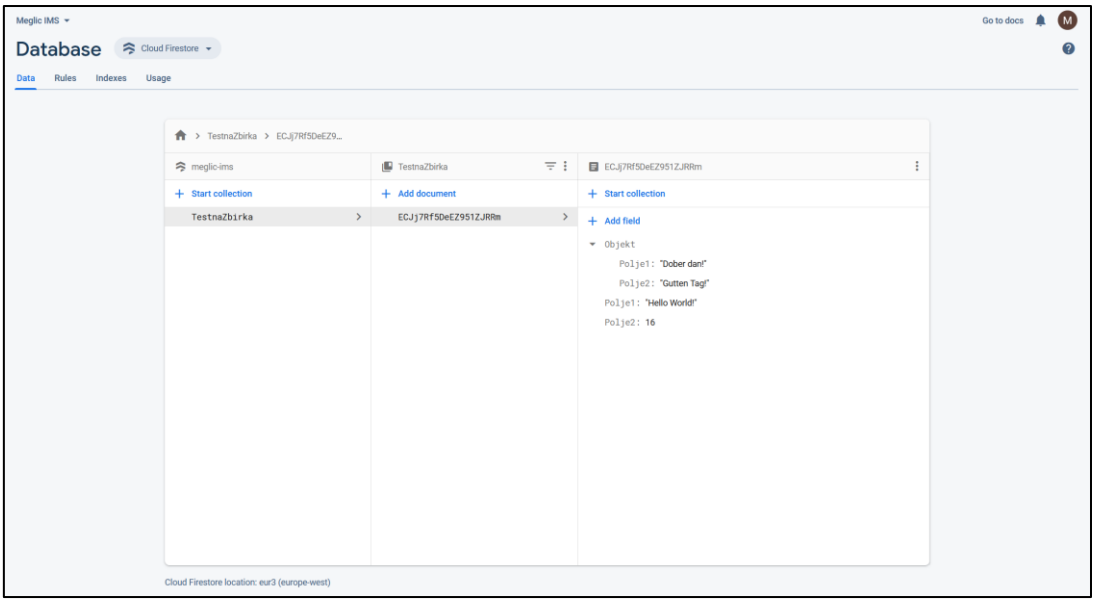
4.1.2. Podatkovna baza (NoSQL)

Naslednja pomembna storitev je podatkovna baza. Firebase nam v tem področju ponuja dve opciji – "Realtime Database" in "Cloud Firestore", ki sta si zelo podobni. V osnovi sta obe implementacija dokumentne NoSQL podatkovne baze. Ker pa je Firestore novejša, hitrejša in priporočena opcija, bomo od tu naprej raziskovali izključno funkcije Firestore.

Firestore je dokumentna NoSQL baza, kar pomeni, da jo sestavlja več zbirk, v katerih so dokumenti z vrednostmi oz. zapisi ter včasih celo podzbirke.

Dobra funkcija Firestore baze je, da če namensko ne določimo imena dokumenta, ga storitev avtomatsko poimenuje z unikatnim nizom znakov, ter tako poskrbi, da dva dokumenta nimata istega imena.

Firestore pa ponuja tudi spletni vmesnik (Slika 5), ki omogoča ogled, dodajanje in manipulacijo podatkov.



Slika 5: Firestore spletni vmesnik (Google, 2020)

»Dokumentne NoSQL baze hranijo podatke v dokumentih, ki so podobni JSON (JavaScript Object Notation) objektom. Vsak dokument vsebuje pare polja in vrednosti. Vrednosti so lahko različnih tipov, kot npr. nizi, števila, logične vrednosti, seznami ali objekti, njihove strukture pa se običajno pokrivajo z objekti, s katerimi operira program. Zaradi pestre izbire podatkovnih tipov in močnih povpraševalnih jezikov so dokumentne podatkovne baze primerne za širok spekter aplikacij in uporabne kot podatkovne baze za splošno uporabo.« (Schaefer, 2020)

4.1.3. Shramba

Poleg hrambe podatkov v podatkovni bazi pa Firebase ponuja tudi shrambo, kjer lahko aplikacija naloži datoteke poljubnega formata ter jih hrani v oblaku. Ta funkcija omogoča razvijalcu hrambo statične vsebine aplikacije (npr. slike, dokumente ipd.), ki jo lahko spreminja preko klicev iz aplikacije same.

4.1.4. Spletno gostovanje

Firebase nam omogoča gostovanje naše aplikacije ali spletne strani na statičnem spletnem naslovu.

Pri nalaganju vsebine pa je obvezna datoteka *firebase.json*, ki opisuje konfiguracijo strežnika. V njej lahko navedemo in konfiguriramo:

- preusmeritve na druge naslove,
- dinamično (funkcije) in statično (gostovanje) vsebino,
- stran 404 (stran, ki se prikaže v primeru neveljavnega naslova),
- dokumente, ki jih želimo prenesti na CDN
- in veliko več.

4.1.5. Funkcije

Funkcije so neke vrste "dvorezni meč", ker jih lahko uporabimo za dinamično prikazovanje vsebine kot alternativno ali dopolnilno opcijo statični vsebini, ki jo prikazujemo s spletnim gostovanjem ali za procesiranje v oblaku, kar nam omogoča reakcije na vnos/spremembe podatkov v podatkovni bazi ali shrambi ter mnogo več.

Je zelo napreden in močan modul Firestore, ker lahko varno in neomejeno dostopa do podatkov, reagira nanje, jih manipulira in ureja, poleg tega pa je najbolj prilagodljiv del celotnega sistema, ker izvaja našo lastno kodo v oblaku.

Eden bonusov uporabe funkcij je izvajanje spletnih storitev, ki delujejo na osnovi Node.js okolja.

4.2. Bootstrap 4

Bootstrap 4 je odprtokodni paket programskih orodij za razvijanje odzivnih aplikacij s HTML, CSS in JavaScript. Paket omogoča hiter razvoj spletnih vmesnikov, ker vnaprej definira veliko slogovnih lastnosti projekta, kar pomeni, da je Bootstrap odlična odskočna deska za razvijanje spletnih strani ter aplikacij.

Prva verzija Bootstrapa je nastala znotraj Twitterja leta 2010 pod imenom "*Twitter Blueprint*". Nekaj mesecev po začetku razvoja pa je Twitter organiziral javni dogodek, na katerem se je z ogrodjem srečalo veliko novih razvijalcev. Ideja jim je bila zelo zanimiva, zato so jo takoj pričeli izboljševati in razvijati.
Skozi leta in z napredkom spletnih tehnologij se je Bootstrap razvil v odzivno ogrodje za kreacijo modernih spletnih strani in aplikacij, ki je trenutno na svoji četrti večji izdaji (s peto izdajo za vogalom).

Glavna prednost pri uporabi Bootstrapa je, da že v naprej definira velikosti zaslonov, pri katerih se spletna stran preoblikuje in prilagaja, določi barve in pisave vmesnika in implementira nekaj pogosto uporabljenih funkcij, kot npr. pogovorno okno za vnos podatkov. Poleg vsega tega pa tudi poskrbi, da so vse te funkcije kar najbolj kompatibilne s platformami na trgu.

4.3. NoSQL

Izraz "NoSQL podatkovna baza" se običajno nanaša na nerelativnostne baze, ki podatke shranjujejo v nestrukturiranih formatih, torej nasprotno bolj znanim SQL oz. entitetno-relacijskim podatkovnim bazam. (Schaefer, 2020)

Pogosto pride do zmote, da NoSQL baze ne morejo hraniti relacijskih podatkov, kar pa nikakor ni res, samo hranijo jih na drugačen način. Zaradi fleksibilne narave NoSQL je lahko oblikovanje relacij velikokrat celo lažje kot pri tradicionalnih SQL bazah.

V osnovi poznamo 4 različne tipe NoSQL baz:

- dokumentna shramba → hrani podatke v dokumentih podobnih JSON objektom;
- shramba ključ-vrednost → hrani podatke kot spremenljivke v parih ključ-vrednost (običajno uporabljajo hash[2] funkcijo za hitrejše iskanje);
- široko stolpčna shramba → hrani podatke v vrsticah z fleksibilnimi stolpci (podobno kot dvodimenzionalne shrambe ključ-vrednost);
- diagramska shramba → hrani podatke v gradnikih, poleg katerih zapišejo tudi relacijo (optimalne so za hranjenje izredno povezanih podatkov npr. socialno omrežje).
  (Schaefer, 2020)

4.4. HTML

»HTML je jezik, ki je namenjen izdelavi spletnih strani, tj. povezavi z drugimi dokumenti, prikazovanju in oblikovanju podatkov. *HyperText* je povezano besedilo, *Markup* pa je angleška beseda za označevanje. Z jezikom HTML torej označujemo in določamo lastnosti besedila.« (Mrhar, 2002)

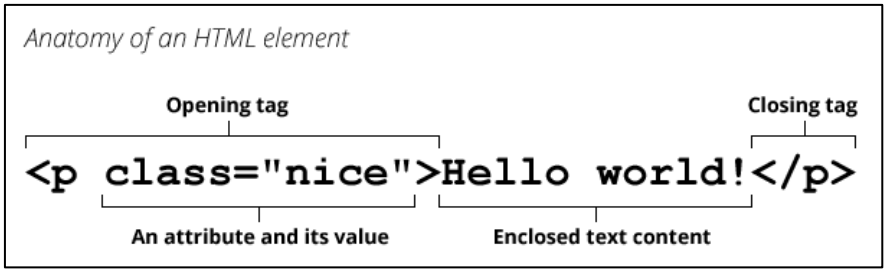
Razvoj se je začel leta 1990, ko je Tim Berners-Lee definiral koncept hiperteksta in ga formaliziral z označevalnim jezikom – HTML. Kmalu je jezik začela formalno navajati skupina IETF, ki je izdala verzijo 2.0. V prihodnjih leti pa je Tim ustanovil W3C (konzorcij za svetovni splet) ter prevzel razvoj jezika. Zaradi premajhnega financiranja se je razvoju verzije 5 pridružila še skupina WHATWG.
(Mozilla, 2020)

HTML5 pa je trenutno najnovejša iteracija standarda, ki definira HTML. Prinaša novo verzijo jezika in večji set modernih tehnologij, ki omogočajo izgradnjo najrazličnejših in predvsem močnejših spletnih strani in aplikacij.

4.4.1. Koncept in sintaksa

HTML dokument je strukturiran dokument elementov (Slika 6), ki jih omejujejo t.i. značke. Vsaka se začne in konča s trikotnim oklepajem (<>). Z nekaj izjemami, ki imajo samo eno značko. (Mozilla, 2020)

Znotraj prve značke običajno lahko določimo še nekaj izbirnih atributov, ki imajo lahko grafični in uporabni pomen (npr. razred, stil, povezava …) ali zgolj pomagajo razvijalcem pri delu z elementom (npr. identifikator, razred …).



Slika 6: Struktura HTML elementa (Mozilla, 2020)

[2] funkcija, ki arbitrarne podatke pretvori v vrednost fiksne dolžine

4.5. CSS

»Kratica CSS je nastala iz besedne zveze Cascading Style Sheet (kaskadni slogi ali slogi CSS). Gre za skriptni jezik, namenjen samo oblikovanju spletnih strani. S CSS oblikujemo prav vse: ozadja, besedilo, povezave, menije, tabele, obrobe … CSS določa, kako mora brskalnik prikazati posamezne elemente HTML.« (Kaltenekar, 2006)

Velja za eno izmed treh osnovnih spletnih tehnologij (skupaj s HTML in JS). Običajno je uporabljena v kombinaciji s HTML, a se lahko kombinira tudi z drugimi označevalnimi jeziki, kot npr. SVG in XML. (Mozilla, 2020)

4.5.1. Sintaksa

»CSS pravilo je set lastnosti, ki se navezuje na selektor. Spodaj je primer, ki naredi vsak HTML odstavek rumene barve s črnim ozadjem (Slika 7).« (Mozilla, 2020)

```
/* The selector "p" indicate that all paragraphs in the document will be affected by that rule */
p {
  /* The "color" property defines the text color, in this case yellow. */
  color: yellow;

  /* The "background-color" property defines the background color, in this case black. */
  background-color: black
}
```

Slika 7: Struktura CSS pravila (Mozilla, 2020)

Selektor je del CSS pravila, ki opiše elemente v dokumentu. Pri tem lahko uporabljamo imena elementov (npr. p, a, body …), identifikatorje, razrede, atribute ter katerokoli kombinacijo naštetih. Poleg teh opcij pa poznamo še pseudorazrede, ki omogočajo določanje pravil samo za specifično stanje elementa (npr. hover, checked …) ali območja pred in po elementu (z uporabo before in after). Nemogoče pa je našteti vse opcije, ker nove specifikacije izhajajo skoraj vsako leto.

4.6. JavaScript

»JavaScript je skriptni jezik, ki omogoča izboljšanje in posodobitev oblike, izdelovanje piškotkov in še veliko več. JavaScript je torej zelo uporaben, kar dokazuje tudi dejstvo, da je v spletu na milijone strani, ki vsebujejo JavaScript.« (Kaltenekar, 2006)

Je zelo lahkoten interpretiran jezik, ki ga dan danes podpirajo že vsi brskalniki, poleg tega pa ga uporabljajo tudi druga okolja, kot so Node.js, Apache CouchDB ipd. Vredno je omeniti tudi, da podpira objektno orientirano programiranje.

Razvoj se je začel z Brendanom Eichom pri Netscape in se je prvič pojavil v brskalniku Netscape Navigator leta 1995. Kmalu se je zanj pojavila podpora tudi v Internet Explorer pod imenom JScript. Leta 1996 pa je ECMA International prevzela standardizacijo jezika. Današnje implementacije JavaScript torej slonijo na ECMAScript standardu. (Mozilla, 2020)

4.6.1. Sintaksa

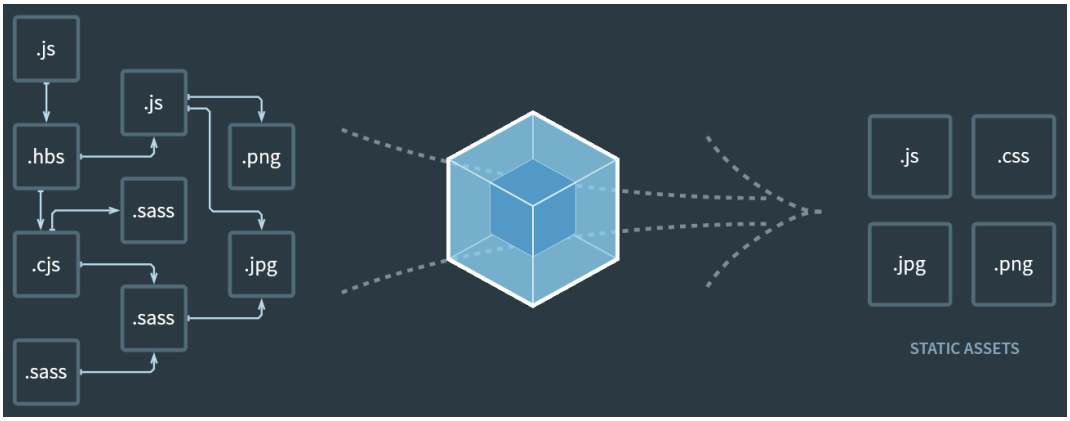
Sintaksa JavaScript oz. ECMAScript je opisana v (preko 700 strani dolgem) dokumentu, ki specificira oz. standardizira celotni jezik.
Dostopen je na naslovu: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf.

4.7. Webpack

Webpack je zelo nastavljiv in dinamičen upravitelj JS aplikacij (Slika 8). Njegova osnovna naloga je združevanje JS skript v snope oz. pakete, ki jih potem lahko uporabimo v HTML dokumentih. Nikakor pa to ni edina funkcija, ki jo ponuja, saj ima za seboj močno skupnost, ki vestno izdeluje nove in vzdržuje obstoječe vtičnike.

Ti vtičniki omogočajo tudi upravljanje z drugimi formati, kot npr. HTML, CSS, ter celo slikami in drugimi sredstvi.

Srce Webpack projekta je datoteka *webpack.config.js*, kjer razvijalec lahko spreminja parametre projekta ter opredeli in nastavi vtičnike. Nastavitve so zapisane v zelo lahko berljivem JSON formatu.



Slika 8: Predstavitev Webpack (Webpack, 2020)

4.7.1. Babel

Babel je vtičnik za Webpack (ter mnogo drugih sistemov), ki je v osnovi samo JS prevajalnik. Poseben in dobrodošel pa je zato, ker omogoča prevajanje JS napisanega po najnovejših ECMAScript[3] standardih v sintakso, ki jo razumejo brskalniki, ki še niso prevzeli ali implementirali teh standardov. Vredno je tudi omeniti, da podpira prevajanje TypeScript[4] kode, vkolikor jo projekt uporablja.

V konfiguraciji lahko določimo verzijo standarda, v katerega želimo prevajati, ali pa kar brskalnike in njihove verzije, ki jih želimo podpirati.

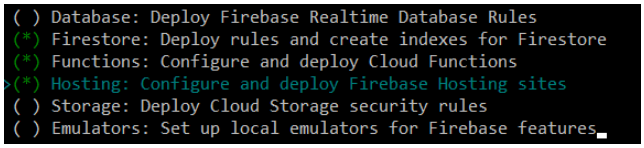
[3] standardizacija skriptnega jezika, ki je osnova za JavaScript
[4] odprtokodni jezik, ki gradi na osnovi JavaScript

5. IZDELAVA REŠITVE

5.1. Priprava delovnega okolja

Ker celoten projekt sloni na razvojni platformi Firebase, je prvi korak inicializacija projekta ter generiranje datotek, ki jih potrebuje Firebase. Za to uporabimo Firebase CLI, ki ga lahko prenesemo iz spletne dokumentacije Firebase (firebase.google.com/docs/cli). Pred začetkom pa se moramo prijaviti v Google račun z ukazom *firebase login*.

Proces na lokalnem sistemu začnemo z ukazom *firebase init* v ukazni vrstici. Program nam ponudi različne storitve, v našem primeru izberemo Firestore, funkcije in gostovanje (Slika 9). V nadaljevanju določimo še ime projekta ter se prijavimo v Google račun. Včasih inicializacija spodleti, zato se moramo prijaviti v spletni vmesnik in nastaviti lokacijo, kjer želimo gostovati.



Slika 9: Inicializacija storitev Firestore

Firebase nam je ustvaril vse potrebne dokumente, zato lahko nadaljujemo z inicializacijo Webpack. Za namestitev le-tega pa potrebujemo Node.js programsko orodje, s katerim dobimo upravitelja paketov **npm**. Node.js prenesemo z njihove spletne strani (nodejs.org). Z ukaznim pozivom zaženemo *npm init*, ki nas vodi skozi nastavitev npm projekta.

Sedaj smo končno pripravljeni za prenos in inštalacijo potrebnih Webpack in Babel paketov. Namestitev izvedemo z ukazom:

npm install webpack webpack-cli babel-loader @babel/core @babel/preset-env @babel/plugin-transform-runtime @babel/plugin-proposal-class-properties --save-dev

Ukaz namesti vse potrebne Webpack in Babel pakete, ki jih moramo sedaj konfigurirati. Za konfiguracijo pa samo uredimo datoteko webpack.config.js, v kateri se vse nastavitve prilagajajo v formatu JSON objekta (Slika 10). Sedaj smo pripravljeni, da pričnemo z razvojem.

```
module.exports = {
    mode: 'production',
    watch: true,
    watchOptions: {
        aggregateTimeout: 300,
        poll: 1000
    },
    entry: {
        main: './public/script/src/main.js',
        codebook: './public/script/src/codebook.js',
        storage: './public/script/src/storage.js',
    },
    output: {
        path: __dirname + '/public/script/bundle',
        filename: '[name].bundle.js',
    },
    module: {
        rules: [
            {
                test: /\.js?$/,
                loader: 'babel-loader',
                exclude: /node_modules/,
                options: {
                    presets: [
                        '@babel/preset-env',
                    ],
                    plugins: [
                        ['@babel/plugin-proposal-class-properties', {loose: true}],
                        ['@babel/plugin-transform-runtime'],
                    ]
                },
            },
        ]
    },
    resolve: {
        extensions: ['.js'],
    },
};
```

Slika 10: Konfiguracija Webpack

5.2. Določanje struktur v podatkovni bazi

Firestore je nestrukturirana podatkovna baza oz. NoSQL, kar pa ne pomeni, da lahko podatke samo stresamo vanjo brez okvirne strukture, ker bi nam to zelo otežilo delo z njimi. Zato bomo najprej vnesli nekaj podatkov, da vidimo, kakšna bi bila logična struktura.

Samo podatkovno bazo sem razdelil na pet zbirk, ki hranijo različne vrste dokumentov. Te zbirke so:

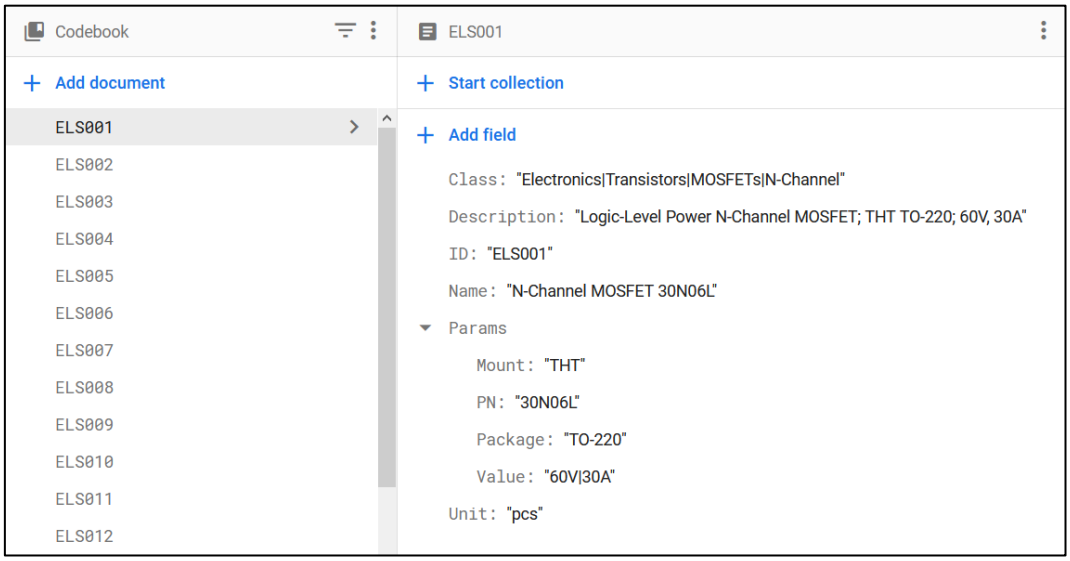
- "Codebook" oz. šifrant → hrani zapise identov;
- "Warehouse" oz. skladišča → hrani zapise skladišč;
- "Inventory" oz. zaloga → hrani zapise zaloge z referenco identa in skladišča;
- "InvTransactions" oz. transakcije → hrani zapise transakcij zaloge;
- "Settings" oz. nastavitve → hrani nastavitve ter vmesne tabele.

Sedaj, ko imamo določeno vrhnjo strukturo baze, pa nas čaka še določitev oblike dokumenta v posamezni zbirki. Za prve štiri zbirke je oblika dokumenta zelo uniformna, ker dokumenti hranijo enaka polja, samo drugačne vrednosti (ta način hrambe zelo spominja na SQL bazo), v zbirki nastavitev pa se bodo v dokumentih hranile najrazličnejše vrednosti.

Strukture bomo določili tako, da bomo preko Firebase spletnega vmesnika vnesli podatke za nekaj identov ter tako zagotovili, da hranimo vse potrebne podatke.

5.2.1. Struktura dokumentov šifranta

Osnovni podatki identa (Slika 11) so identifikator, naziv ter opis. Poleg teh pa bomo hranili tudi klasifikacijo, številno enoto in seznam parametrov, v katerega bomo zapisali karakteristike komponent ipd.



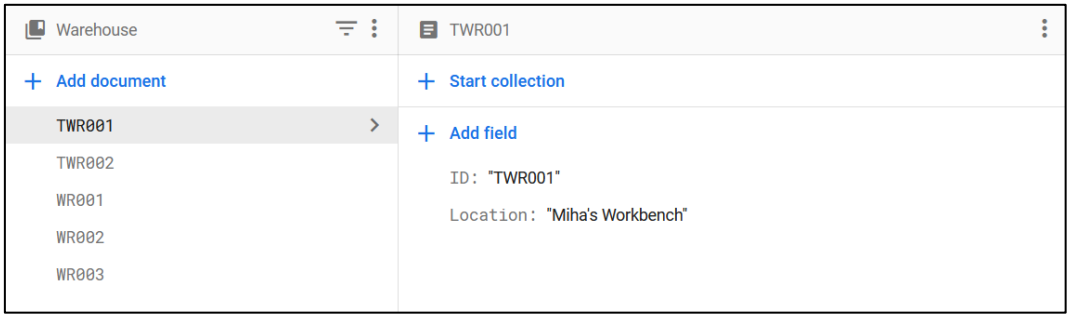
Slika 11: Struktura identa

Odločil sem se za nenumerične identifikatorje, ker bodo tako lažje razumljivi uporabniku, ki operira z njimi. Primer takega identa je *ELS001*, v katerem je prva polovica okrajšava za "Electronic Semiconductor", druga polovica pa je zaporedna številka.

Ker bo parametre (z izjemo identifikatorja) nastavljal uporabnik, jih bomo hranili v obliki besedila.

5.2.2. Struktura dokumentov skladišča

O skladišču (Slika 12) bomo hranili zelo malo podatkov, saj nas v resnici zanimata samo dva parametra – identifikator in lokacija. Ponovno uporabljam nenumerični identifikator.

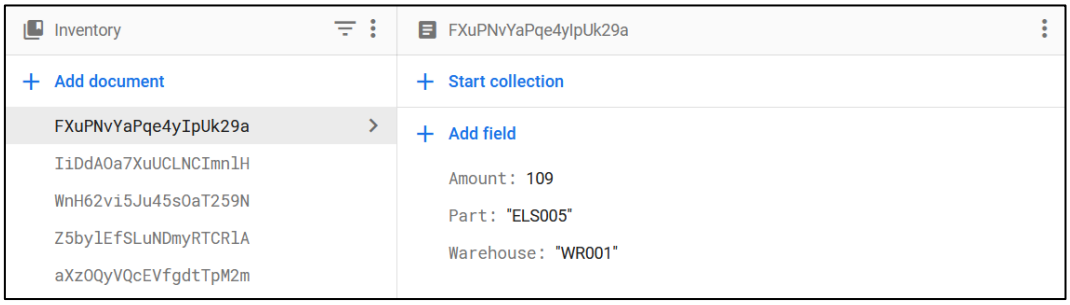


Slika 12: Struktura skladišča

5.2.3. Struktura dokumentov zaloge

Pri dokumentu zaloge (Slika 13) je glavni podatek količina, ki jo bomo hranili kot število. Tu nam ponovno naredi korak nasproti Firestore, saj ne ločuje numeričnih podatkov na cela in realna števila.

Da pa nam bo podatek količine koristil, potrebujemo še identifikator identa in skladišča, v katerem je ident hranjen.

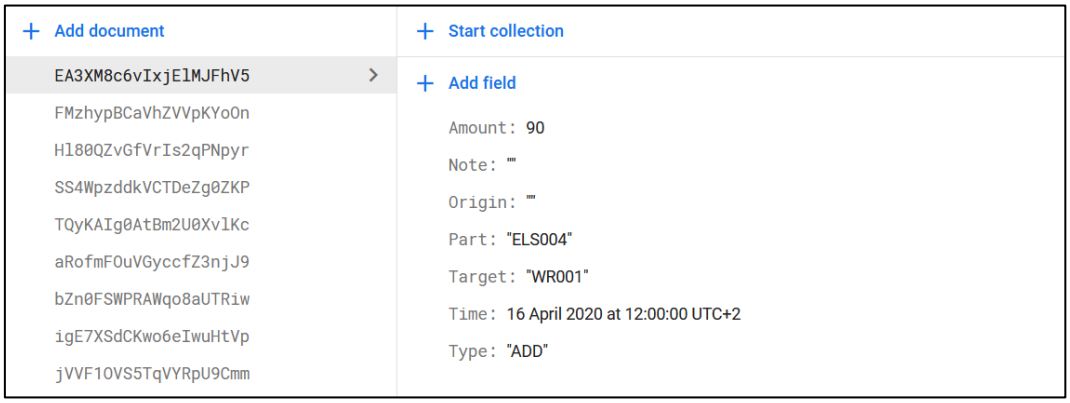


Slika 13: Struktura zaloge

Za razliko od prejšnjih dveh zbirk tu izkoriščamo značilnost Firestore, ki nam avtomatsko poimenuje dokument z unikatnim nizom znakov. V tem primeru namreč ne bomo iskali dokumentov po identifikatorju, saj je lahko en ident hranjen v več različnih skladiščih in eno skladišče lahko vsebuje več različnih identov. Pri iskanju dokumentov bomo zato uporabili povpraševalne ukaze.

5.2.4. Struktura dokumentov transakcij

Transakcija (Slika 14) je ena obširnejših struktur, saj nas zanima veliko spremenljivk, kot so: tip transakcije, ident, izvorno skladišče, ciljno skladišče, količina, čas transakcije in opombe uporabnika.



Slika 14: Struktura transakcije

Ponovno bomo uporabili avtomatsko poimenovanje dokumentov, saj se bodo tako skladišča kot identi ponavljali.

5.2.5. Zbirka nastavitev

V zbirki nastavitev bomo hranili nastavitve, ki jih lahko ureja administrator, ter vmesne tabele, s katerimi bodo upravljale Firebase funkcije in občasno administrator.

Pripravili bomo tri dokumente:

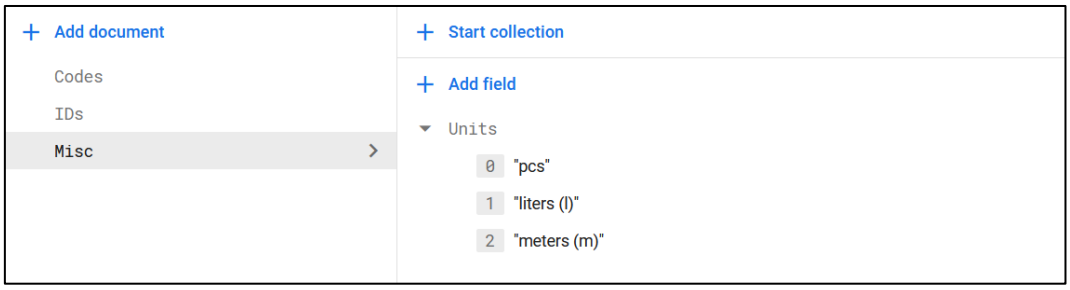
- "Codes" → vmesna tabela, ki jo sestavljajo pari identifikatorja in naziva identa;
- "IDs" → nastavitev nenumeričnih identov, sestavljena iz besednega dela in naslednjega odprtega zaporednega števila;
- "Misc" → ostale nastavitve oz. konstante, ki se bodo uporabljale v grafičnem vmesniku.

S prvim dokumentom bodo operirale funkcije, zato nas zaenkrat zanima samo struktura → par identifikatorja in naziva identa. Namen dokumenta je, da omogočimo branje osnovnih informacij vseh identov brez obremenjevanja podatkovne baze oz. z minimalnim številom branj. To je posebej pomembno zato, ker bodo dokument brali spustni meniji ipd.

V dokumentu "IDs" bo administrator lahko dodal besedne dele identifikatorjev ter spreminjal začetno zaporedno število. Struktura je zelo preprosta (Slika 15), za vnos novega besednega dela identifikatorja pa administrator doda novo polje z numerično vrednostjo 1 oz. željenega začetnega zaporednega števila.

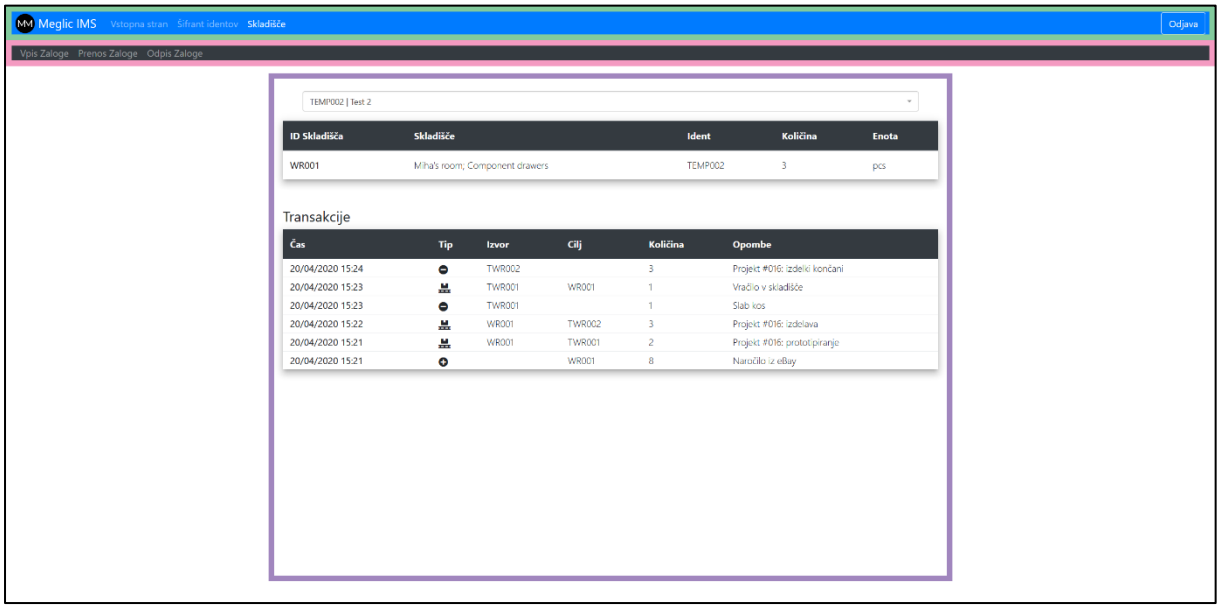
Slika 15: Struktura dokumenta "IDs"

V zadnjem dokumentu (Slika 16) pa bomo hranili seznam številnih enot, ki so na voljo uporabniku. Ker bodo enote hranjene v seznamu, lahko kasneje ta dokument uporabimo za hrambo drugih statičnih nastavitev v ločenih seznamih.



Slika 16: Struktura dokumenta "Misc"

5.3. Izdelava grafičnega vmesnika



Slika 17: Dispozicija vmesnika

Vmesnik bomo razdelili na tri glavne dele (Slika 17) – navigacijsko vrstico ( ), akcijsko vrstico ( ) in telo strani ( ).

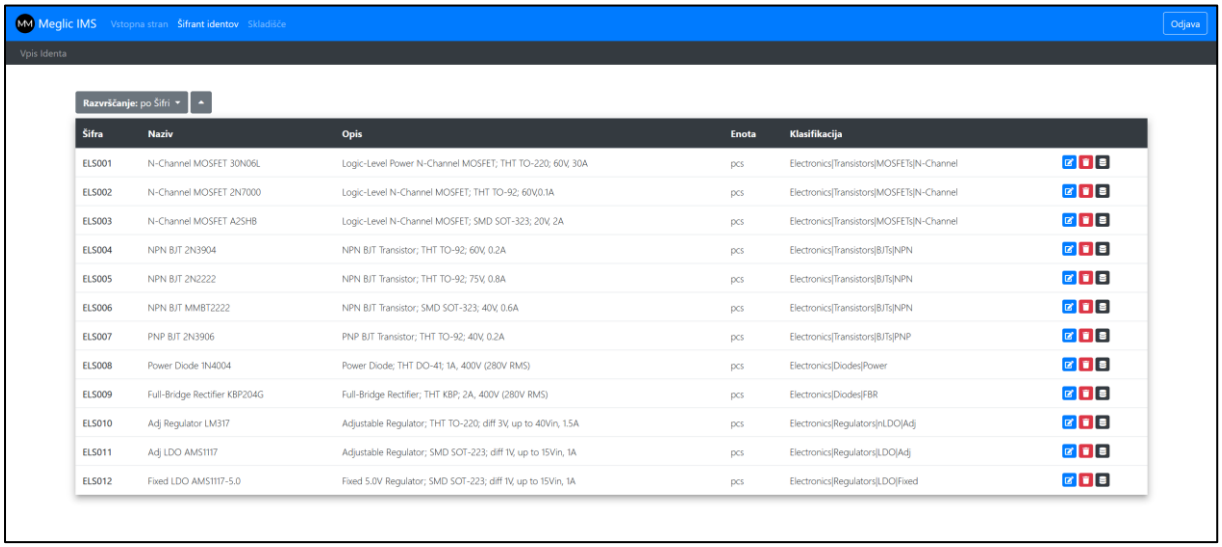
Z navigacijsko vrstico bomo omogočali uporabniku premikanje po aplikaciji, na desni rob pa bomo dodali gumb za prijavo/odjavo iz aplikacije, poleg pa izpis trenutnega uporabnika. Ker uporabljamo Bootstrap, pa bo navigacijska vrstica na mobilnih napravah stisnjena v spustni meni.

V akcijsko vrstico bomo dodali gumbe za funkcije, ki jih potrebuje uporabnik, kot so dodajanje, brisanje in urejanje vnosov. Te akcije bodo na voljo samo prijavljenim uporabnikom, ki imajo pravice v Firestore tabeli, zato bomo njihovo vidljivost spreminjali z JavaScript funkcijami. Poleg tega pa jo želimo skriti uporabnikom na napravah manjših od tabličnega računalnika, ker bodo okna, ki jih odprejo prenerodna za uporabo na mobilnih telefonih. To bomo realizirali z Bootstrap razredi, ki se odzivajo na velikost zaslona naprave.

Telo strani pa je namenjeno prikazovanju podatkov, kot so različne tabele identov, zaloge ipd. Glavno pri teh tabelah je, da se aktivno posodabljajo s podatki v podatkovni bazi ter uporabniku podatke posredujejo čim bolj nazorno.

5.3.1. Šifrant identov

Središče okna "Šifrant identov" (Slika 18) je tabela, ki prikazuje osnovne podatke o vsakem identu v željenem vrstnem redu, ter prijavljenim uporabnikom omogoča urejanje, brisanje in vpogled zaloge za vnesene elemente.



Slika 18: Šifrant identov

Nad tabelo smo implementirali spustni meni, ki omogoča nastavitev različnih vrstnih redov razvrščanja. Poleg njega pa še gumb, ki spreminja vrstni red med padajočim in naraščajočim.

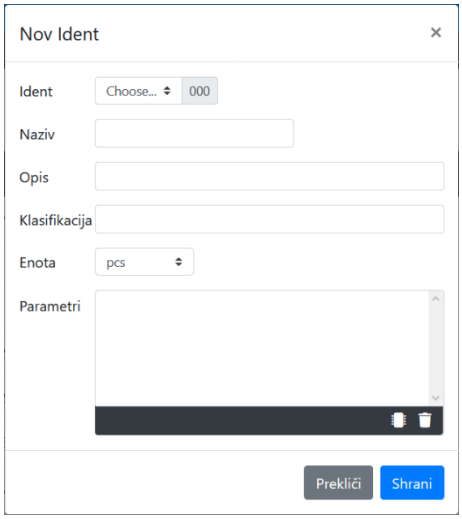
V osnovi tabela naloži 20 elementov, da zmanjšamo število branj, ki jih izvedemo iz podatkovne baze. Ko pa se uporabnik približa dnu naloženih vrstic, jih koda v ozadju iz baze vrne še dodatnih 20, dokler ne prikaže vseh podatkov.

Vsaka vrstica v tabeli ima tri gumbe, ki omogočajo urejanje, brisanje in pregled zaloge. Gumb za urejanje odpre okno z vsemi podatki identa, ki jih uporabnik lahko prilagaja ter na koncu shrani. Vmesnik je enak vmesniku za dodajanje identov. Gumb za izbris identa odpre potrditveno okno kot varovalo, da uporabnik ne izbriše identa ponesreči. Zadnji gumb pa uporabnika preprosto preusmeri na zavihek "Skladišče" ter mu prikaže zalogo za ident.

Nedvomno najpomembnejši pa je gumb v akcijski vrstici, ki odpre okno za dodajanje oz. vpis identa (Slika 19). V oknu lahko uporabnik iz spustnega menija najprej izbere besedni del identifikatorja, kar posodobi numerični del z naslednjo zaporedno številko.

V osrednjih poljih mora uporabnik navesti zahtevane parametre identa.

Na dnu okna pa je še polje parametrov, namenjeno predvsem parametrom elektro komponent. S klikom na ikono integriranega vezja se odpre vnaprej določen set parametov, ki jih lahko uporabnik po potrebi tudi odstrani.

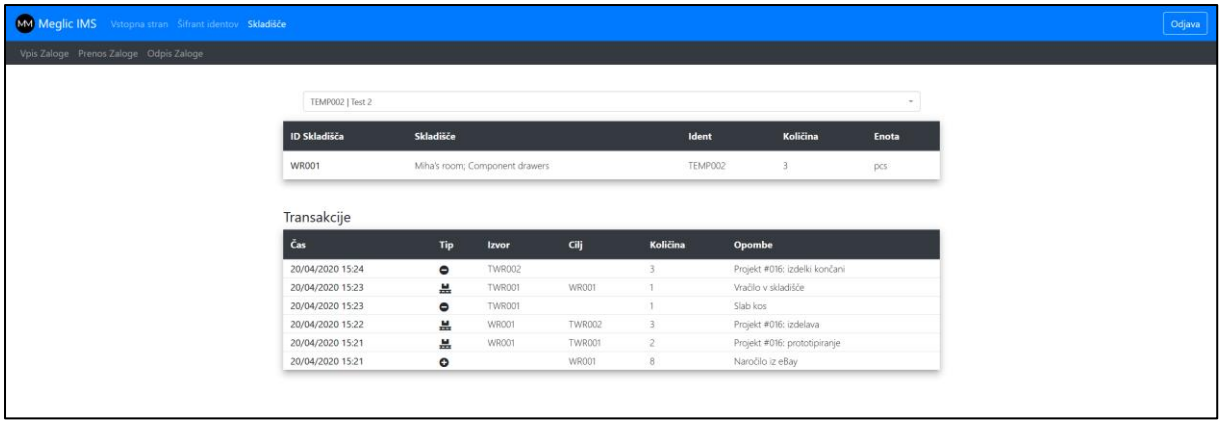


Slika 19: Okno za dodajanje identa

5.3.2. Skladišče

V središču okna "Skladišče" (Slika 20) je spustni meni, kjer lahko uporabnik izbere ident, za katerega se zanima, kar avtomatsko posodobi spodnji tabeli.

Prva tabela prikazuje zalogo izbranega identa v skladiščih (če je zaloga v danem skladišču nič, tega vnosa ne prikaže), druga tabela pa zadnjih 15 transakcij identa.



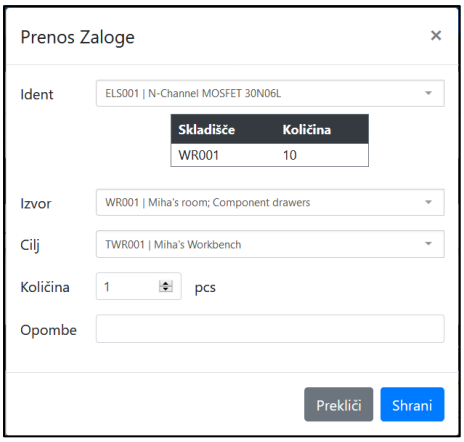
Slika 20: Skladišče

Tabela transakcij, kot že omenjeno, navaja zadnjih 15 transakcij, ki so kronološko urejene v padajočem vrstnem redu. Tip transakcije pa je prikazan z eno izmed treh ikon. Ta tabela uporabniku omogoča vpogled v nedavne transakcije identa ter, v primeru napake pri zalogi, odkrivanje napake v vpisih.

Ponovno pa telo strani spremljajo akcijski gumbi. Ti omogočajo vpis, prenos in odpis zaloge. V osnovi in delovanju so si zelo podobni, zato si podrobneje poglejmo samo najkompleksnejši obrazec – prenos zaloge.

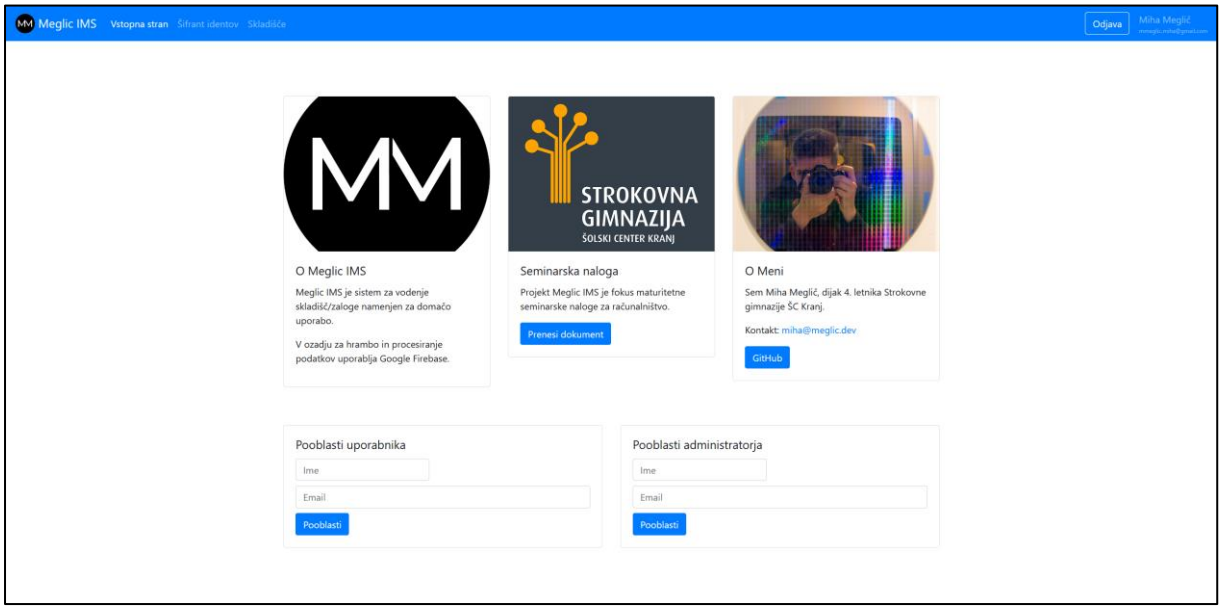
Okna za delo z zalogami (Slika 21) najprej od uporabnika v spustnem meniju zahtevajo izbor identa, kar v obrazcih za prenos in odpis zaloge spremeni polja, ki so na voljo v spustnem meniju skladišč. Ta funkcija je nujna, saj ne moraš odstraniti kosov iz lokacije, kjer kosov ni. Zato se takoj, ko uporabnik izbere skladišče, spremeni tudi zgornja meja polja za vnos količine.

V polje opombe uporabnik lahko navede podatke za osebno evidenco in/ali jasnejšo reprezentacijo v tabeli transakcij.



Slika 21: Okno za prenos zaloge

5.3.3. Vstopna stran



Slika 22: Vstopna stran

Primarna funkcija vstopne strani (Slika 22) je, da uporabniku poda osnovne informacije o projektu. Za uporabnike z dodeljenimi pravicami administratorja pa se na dnu prikažeta še obrazca za opravljanje pravic uporabnikov.

5.3.4. Uvoz Firebase, Bootstrap in drugih knjižnic

Na vsakem HTML dokumentu potrebujemo nekaj osnovnih JavaScript in CSS knjižnic, ki jih uvozimo z <link> in <script> elementi (Slika 23). Knjižnice uvažamo s CDN naslovov, ker bi lokalna shramba dodatno obtežila oz. povečala obsežnost aplikacije.

```
<!-- Firebase imports -->
<script defer src="/__/firebase/7.8.2/firebase-app.js"></script>
<script defer src="/__/firebase/7.8.2/firebase-auth.js"></script>
<script defer src="/__/firebase/7.8.2/firebase-firestore.js"></script>
<script defer src="/__/firebase/init.js"></script>

<!-- Bootstrap imports; includes jQuerry and Popper -->
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
      integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
        integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
        integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"
        integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

<!-- Select2 import -->
<link href="https://cdn.jsdelivr.net/npm/select2@4.0.13/dist/css/select2.min.css" rel="stylesheet"/>
<link href="style/select2-bootstrap.min.css" rel="stylesheet"/>
<script src="https://cdn.jsdelivr.net/npm/select2@4.0.13/dist/js/select2.min.js"></script>

<!-- Import icons -->
<link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">
<script src="https://kit.fontawesome.com/fc95ba0b49.js" crossorigin="anonymous"></script>
```

Slika 23: Uvoz knjižnic

Ta korak se ponovi v glavah vseh straneh, saj knjižnice uporabljamo v celotni aplikaciji. Nekatere za oblikovanje strani, druge pa za implementacijo funkcionalnosti.

Ker pa bomo lastno JavaScript in CSS kodo pisali v ločene dokumente, na vsaki strani uvozimo tudi primerne lokalne skripte in stile.

5.4. Procesiranje podatkov v ozadju

Tabeli, v kateri lahko uporabnik vnaša in ureja podatke, sta šifrant in transakcije. Seveda pa se tam obdelava podatkov ne zaključi. Kot je omenjeno pri določanju struktur, se nekatere vmesne tabele posodabljajo z uporabo Firebase funkcij (Slika 24). To pa je še posebej enostavno, ker je ena od opcij za sprožitev funkcije kar manipulacija dokumentov v zbirki.

Dokument "Codes" se mora posodobiti vsakokrat, ko pride do spremembe naziva identa, zato za spremembe poslušajo tri različne funkcije. Vsaka pa spremlja svoj dogodek, ena dodajanje (Slika 25), druga urejanje in zadnja brisanje iz tabele. Poleg tega prva funkcija posodobi tudi dokument "IDs", kjer poveča zaporedno število naslednjega identa. Pri tem funkcija vedno preveri, da je vneseni identifikator res zaporeden, saj ima administrator možnost ročnega vnašanja, kar pa ne sme povečati zaporednega števila.

Function	Trigger	Region	Runtime	Memory	Timeout
itemAdded	document.create Codebook/{id}	europe-west3	Node.js 8	256 MB	60s
itemModified	document.update Codebook/{id}	europe-west3	Node.js 8	256 MB	60s
itemRemoved	document.delete Codebook/{id}	europe-west3	Node.js 8	256 MB	60s
transactionAdded	document.create InvTransactions/{transaction}	europe-west3	Node.js 8	256 MB	60s

Slika 24: Firebase funkcije

Poleg tega pa uporabniki pri vpisovanju transakcij zbirke zaloge nikoli ne urejajo direktno, ker z uporabo vmesnih transakcij lahko hranimo zgodovino sprememb identa. Zato se funkcija za vpis spremembe zaloge sproži vsakokrat, ko dodamo zapis v zbirko transakcij.

```
exports.itemAdded = functions.region('europe-west3').firestore
    .document('Codebook/{id}')
    .onCreate(doc => {
        const entry = doc.data();
        const id = entry['ID'];
        const idClass = id.substr(0, id.length - 3);
        const idsRef = admin.firestore().collection('Settings').doc('IDs');
        const codesRef = admin.firestore().collection('Settings').doc('Codes');

        let idsData = {};
        idsData[idClass] = admin.firestore.FieldValue.increment(1);
        let codesData = {};
        codesData[id] = entry['Name'];

        return idsRef.get().then(doc => {
            if (doc.data()[idClass] === parseInt(id.substr(id.length - 3))) {
                return idsRef.update(idsData);
            } else {
                console.warn('Item number out of order:', entry);
                return Promise.resolve('Item number out of order');
            }
        }).then(() => {
            return codesRef.update(codesData);
        }).then(() => {
            console.log('Item successfully added:', entry);
            return Promise.resolve('Item add successful');
        }).catch(err => {
            console.error('Item add failed:', entry, 'Error:', err);
            return Promise.reject('Item add failed');
        });
    });
```

Slika 25: Firebase funkcija - dodajanje identa

5.5. Varnost podatkov

Kot je razvidno iz vmesnika, je za urejanje podatkov potrebna prijava, ki v ozadju komunicira s Firebase modulom za overitev in deluje na principu žetonov.

Ko uporabnik klikne gumb prijavi, JavaScript skripta v ozadju preko zavarovane TLS povezave pošlje uporabnikov mail in geslo na Firebase strežnik ter nazaj prejme žeton, ki je potem pripet vsakemu klicu na podatkovno bazo ali kamorkoli drugam znotraj Firebase platforme. V primeru, da uporabnik žeton izgubi ali se odjavi, Firebase sproži klic na funkcijo, ki jo definira razvijalec. V našem primeru le-ta skrije gumbe, za katere je potrebna overitev.

Seveda pa skrivanje gumbov ne zagotovi varnosti, zato Firestore in druge Firebase storitve ponujajo set varnostnih pravil, kjer razvijalec definira zahteve, ki jih mora izpolnjevati prošnja za podatke, preden se storitev odzove.

Takole izgledajo naše varnostne nastavitve za Firestore (Slika 26):

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /Codebook/{ident} {
      allow read;
      allow write: if request.auth.token.authorized == true;
    }
    match /Settings/{setting} {
      allow read;
    }
    match /Inventory/{entry} {
      allow read: if request.auth.token.authorized == true;
    }
    match /Warehouse/{entry} {
      allow read: if request.auth.token.authorized == true;
    }
    match /InvTransactions/{transaction} {
      allow create, read: if request.auth.token.authorized == true;
    }

    match /{document=**} {
      allow read, write: if request.auth.token.admin == true;
    }
  }
}
```

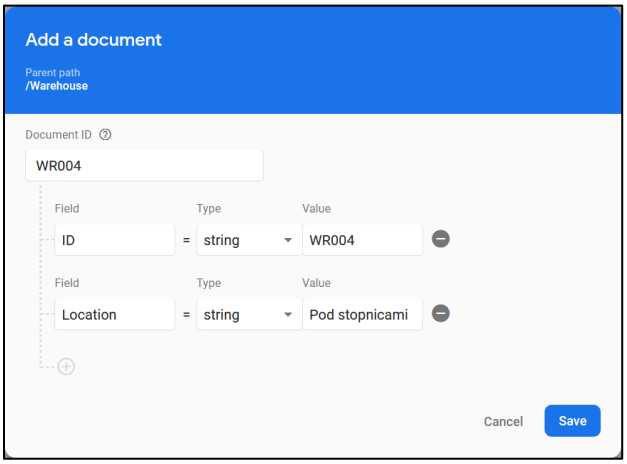
Slika 26: Varnostna pravila za Firestore

Varnostna pravila zahtevajo, da je uporabnik avtoriziran, kar pomeni, da mora uporabnika ustvariti in pooblastiti administrator.

Če pa pogledamo zadnje pravilo, vidimo, da uporabnikom s statusom administratorja ne omejujemo branja ali pisanja v nobeni tabeli, kar pomeni, da administrator z zadostnim poznavanjem sistema lahko podatke vnaša in ureja preko konzole, če je to potrebno.

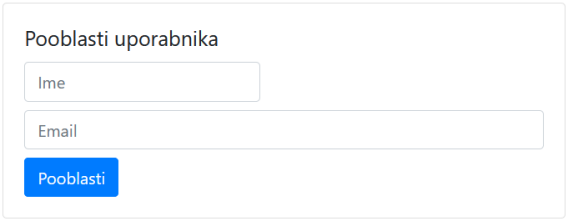
5.6. Administracija sistema

Za administracijo so večinoma potrebne spremembe v Firestore bazi, zato večina tega dela poteka na Firebase spletnem vmesniku. Preko tega poteka dodajanje besednega dela identifikatorja, prav tako pa se tam dodajajo skladišča (Slika 27) in urejajo številne enote.



Slika 27: Dodajanje skladišča

Dodajanje uporabnika pa je postopek v dveh korakih. Najprej v Firebase modulu za overitev dodamo uporabnika. Če želimo, da sam nastavi geslo, uporabimo funkcijo "Reset password", kar pošlje uporabniku pošto s povezavo za ponastavitev gesla. Nato pa se z administrativnim uporabnikom prijavimo v aplikacijo ter na prvi strani vpišemo ime in e-mail uporabnika v polje "Pooblasti uporabnika" (Slika 28).



Slika 28: Polje za pooblastitev uporabnika

6. BETA TESTIRANJE

Pri testiranju aplikacij so najpogosteje uporabljene metode črne, bele in sive škatle. Vse tri so metode beta testiranja, kar pomeni, da pri tem ne sodeluje razvijalec kode. Razlikujejo pa se v količini informacij o programu, ki jih podamo testerju.

Aplikacija je namenjena za interno/domačo uporabo, zato sem za beta testerja prosil očeta - Gregorja Megliča. Kot vodja tehnologije in razvojni inženir ima precej izkušenj z uporabo ERP sistemov, ki so veliko bolj obsežni kot samo sistem za vodenje zaloge.

Kot prvi uporabnik je o rešitvi povedal: »Zelo pregleden in intuitiven GUI. Meniji so tam, kjer jih pričakuješ. Ravno tako okna za vpis identa in skladiščne transakcije. Prijava deluje pravilno. Preveril sem delovanje vseh funkcij, vpis identa, urejanje identa, brisanje identa in skladiščne transakcije. Vse delujejo kot pričakovano.«

6.1. Testiranje črne škatle

»Testiranje črne škatle pomeni, da o notranjem delovanju aplikacijske rešitve ne vemo ničesar. Zato preverjamo funkcionalnosti samo preko vmesnika. S postopkom testiranja črne škatle torej preverjamo, da določeni vhodi v škatlo (vhodni podatki) rezultirajo v pričakovanih izhodih (rezultatih) iz nje. Ker je notranje delovanje oziroma implementacija aplikacijske rešitve pri tej metodi neznana, v tem primeru zato preverjamo skladnost z začetno specifikacijo rešitve (regularnost poslovnih procesov). Torej preverjamo delovanje na višjem funkcionalnem nivoju, pri čemer podrobnosti delovanja niso znane.« (Koritnik, 2010)

Testiranje črne škatle je izvedel oče, predlagal je naslednje izboljšave:

- iskalnik po šifrantu identov → ko bo šifrant precej večji, zna biti iskanje identov precej zamudno;
- pogled zaloge po skladiščih → trenutno je na voljo samo prikaz po identih;
- kosovnice/recepti → seznami količin za pripravo danega izdelka.

Poleg teh predlogov pa ni našel drugih pomanjkljivosti ali napak.

6.2. Testiranje bele škatle

»Od metode testiranja črne škatle se ta razlikuje v tem, da so testerju znani in tudi dostopni notranji algoritmi in delovanje aplikacijske rešitve. Rezultat tega védenja so podrobnejši testni načrti, ki testirajo regularnost internega delovanja programov. … Pri testiranje bele škatle razvijalec napiše testne programe, ki enote testirajo pravilnost na več različnih načinov.« (Koritnik, 2010)

Ker se testiranje bele škatle zanaša na intimno poznavanje izvorne kode ter sistema, na katerem deluje, sem prelomil pravilo beta testiranja ter nekaj scenarijev testiral sam. Že od vpeljave funkcij za urejanje podatkovne baze se mi je zazdevalo, da je to mogoča točka napake, a je takrat nisem imel časa testirati.

Napisal sem kratko skripto, ki je v hitri sekvenci dodala 10 novih identov, ter preveril, kaj se dogaja s števcem naslednjega zaporednega števila v podatkovni bazi. Po 10 sekvenčnih vnosih bi morala biti vrednost števca za deset večja (torej 11, če smo začeli z 1), a vrednost je bila 3. Moj sum je še dodatno podkrepil izpis iz Firestore funkcij (Slika 29).

Poskus sem izvedel še nekajkrat, da sem se prepričal, da je ponovljiv. Najboljša rešitev je, da celotni proces dodajanja izvedem v Firestore funkciji, ki jo lahko pokliče uporabnik. Tako lahko namesto več klicev na bazo izvedemo celoten postopek z Firestore transakcijo, ki poskrbi, da so podatki pravilni, preden jih vnese in posodobi.

Slika 29: Izpis napak za funkcijo itemAdded (med testiranjem)

6.3. Testiranje sive škatle

»Metoda testiranja sive škatle je kombinacija prejšnjih metod črne in bele škatle. Testiranje aplikacijske rešitve poteka na nivoju uporabnika oziroma uporabniškega vmesnika (enako kot v primeru črne škatle), testni procesi pa so prilagojeni védenju notranjega delovanja notranjih funkcionalnosti (bela škatla).« (Koritnik, 2010)

Testiranje sive škatle je v osnovi testiranje, ki ga vsak razvijalec opravlja sproti, ko implementira aplikacijo, zato se veliko napak odstrani spotoma. Poleg tega pa sem ponovno prosil za pomoč očeta. Ker ni seznanjen s programskim jezikom JavaScript, sem mu kodo na kratko razložil.

V testiranju je odkril samo eno napako. Pri vnosu identa funkcija prebere zaporedno število identa iz HTML elementa, kamor ga je prej druga funkcija nastavila. To pomeni, da je z orodji modernih brskalnikov lahko spremenil vrednost HTML elementa in funkcija je dodala ident s to vrednostjo. Rešitev tega problema pa že kar sovpada z rešitvijo prejšnjega. Če bi dodajanje identa prestavili v Firestore funkcijo ter uporabili transakcijo, bi lahko zaporedno število pridobili kar direktno iz podatkovne baze in tako zagotovili pravilnost podatka.

7. ZAKLJUČEK

Sistem za vodenje skladišča dosega željene kriterije. Uporabniki lahko v njem upravljajo z identi ter zalogo, kar sta primarni nalogi. To pa še zdaleč ne pomeni, da je razvoj zaključen.

Kot smo ugotovili v beta testiranju, aplikacija ni popolna, kadar jo izpostavimo hitremu vpisovanju podatkov (6.2. Testiranje bele škatle), ter nima zadostne zaščite pri vpisovanju podatkov (6.3. Testiranje sive škatle). Prvi popravki bi bili zato premikanje teh operacij v spletne funkcije ter implementiranje s pomočjo Firebase transakcij, kar bi imelo postranski efekt manjše količine kode na strani uporabnika.

Videli smo, da rezultat beta testiranja (sploh v primeru bele škatle) lahko poda tudi nekaj zelo dobrih predlogov za nove funkcije. Te se v prihodnosti zagotovo splača implementirati, saj izboljšujejo uporabnikovo izkušnjo ter pospešijo delo z aplikacijo.

Večina administracije še vedno poteka v Firestore spletnem vmesniku (npr. dodajanje skladišč, uporabnikov ipd.), zato bi bil logični naslednji korak dodajanje zavihka za administracijo, ki je dostopen samo uporabnikom s statusom administratorja. Tam lahko dodajamo uporabnike, skladišča in besedne dele identov. Seveda pa je to odlična priložnost za kakšne dodatne funkcije, kot npr. nadaljnja razdelitev vlog po pravicah za posamezno zbirko ali posamezno akcijo (tj. dodajanje, urejanje, brisanje in branje).

Celotna logika aplikacije je napisana v programskem jeziku JavaScript, ki je primarni jezik spletnih aplikacij, ampak pri takem obsegu hitro pridejo do izraza njegove pomanjkljivosti. JavaScript v deklaraciji spremenljivk in argumentov ne zahteva oz. ne podpira tipov, kar lahko hitro postane problem, saj razvojno okolje težje zazna napako v podanih argumentih. To lahko popravimo na dva različna načina. Prvi je uporaba JSDoc označevalnega formata, ki ga razume večina prevajalnikov ter integriranih razvojnih okolij, a zahteva od razvijalca več časa. Drugi način pa je prehod na programski jezik TypeScript, ki osnovi JavaScript doda eksplicitne tipe spremenljivk, na koncu pa se vseeno prevede nazaj v JavaScript, ki ga brskalnik razume. Prevajanje bi bilo lahko avtomatsko urejeno z Webpack.

Zahtevnejša nadgradnja pa bi bil prenos celotne aplikacije v enega izmed programskih ogrodij, kot so Angular, React ali Vue.js, ki omogočajo večjo generalizacijo kode na posamezne elemente (npr. tabela bi bila svoj element, pojavno okno svoj element …), ki so povezani s kodo, potrebno za njihovo delovanje.

Bolj ko gledamo rezultat, bolj očitno postaja, da se programerjevo delo nikoli ne konča.

VIRI IN LITERATURA

Google. (april 2020). *Firebase Documentation*. Pridobljeno iz Google Firebase: https://firebase.google.com/docs

Kaltenekar, M. (2006). *Oblikovanje spletnih strani : HTML, CSS in JavaScript : Hitri vodnik.* Ljubljana: Založba Pasadena.

Koritnik, R. (2010). *Metode za testiranje programskih aplikacij in njihova tržna razširjenost.* Ljubljana: [Koritnik R.].

Mozilla. (april 2020). *CSS*. Pridobljeno iz Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Glossary/CSS

Mozilla. (april 2020). *HTML*. Pridobljeno iz Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Glossary/HTML

Mozilla. (april 2020). *JavaScript*. Pridobljeno iz Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Glossary/JavaScript

Mrhar, P. (2002). *XHTML 1.1 in slogi CSS2.* Šempeter pri Gorici: Flamingo Založba.

Schaefer, L. (2020, april). *NoSQL Databases Explained*. Retrieved from MongoDB: https://www.mongodb.com/nosql-explained

Webpack. (april 2020). *Webpack Docs*. Pridobljeno iz Webpack: https://webpack.js.org/concepts/

POJMOVNIK

IDENT	šifra oz. identifikator skupine enakih kosov
ŠIFRANT IDENTOV	tabela, v kateri hranimo idente in njihove parametre
VPIS ZALOGE	dodajanje kosov v skladišče
PRENOS ZALOGE	prenašanje kosov iz enega v drugo skladišče
ODPIS ZALOGE	odstranjevanje kosov iz skladišča
KOSOVNICA	popis materiala potrebnega za izdelavo izdelka

KRATICE IN AKRONIMI

HTML	HyperText Markup Language
IETF	The Internet Engineering Task Force
CLI	Command Line Interface
CDN	Content Distribution Network
CSS	Cascading Style Sheet
JS	JavaScript
CDN	Content Delivery Network
IMS	Inventory Management Software
ERP	Enterprise Resource Planning
UML	Unified Modeling Language

KAZALO SLIK