

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Žiga Marolt

**Spletni pregled in urejanje
raziskovalnih in drugih publikacij**

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Mira Trebar

Ljubljana, 2022

COPYRIGHT. Rezultati diplomske naloge so intelektualna lastnina avtorja in matične fakultete Univerze v Ljubljani. Za objavo in koriščenje rezultatov diplomske naloge je potrebno pisno privoljenje avtorja, fakultete ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil \LaTeX .

Kandidat: Žiga Marolt

Naslov: Spletni pregled in urejanje raziskovalnih in drugih publikacij

Vrsta naloge: Diplomaska naloga na visokošolskem strokovnem programu prve stopnje Računalništvo in informatika

Mentor: doc. dr. Mira Trebar

Opis:

Kandidat naj v diplomskem delu implementira spletno rešitev za vnos različnih publikacij in dodatnih vsebin z namenom urejanja in pregledovanja pomembnih informacij. Predlagana rešitev naj bo zasnovana tako, da imajo prijavljeni uporabniki dostop do podatkov, urednik skrbi za vnos in urejanje, administrator pa ima celovit nadzor nad delovanjem. Uporaba aplikacije naj bo predstavljena za objavljene publikacije in pomembne podatke s področja živil.

Title: Online review and editing of research and other publications

Description:

In the diploma thesis, the candidate should implement a web solution for inserting various publications and additional content to edit and review important information. The proposed solution should be designed for registered users to have access to data, the editor takes care of input and editing, and the administrator has comprehensive control over the operation. The use of the application should be presented for publications and important data in the field of food.

Zahvaljujem se mentorici doc. dr. Miri Trebar za uso pomoč in družini za podporo.

Kazalo

Povzetek

Abstract

1	Uvod	1
2	Razvoj aplikacije	3
2.1	Življenjski cikel razvoja	3
2.2	Pregled zahtev	5
2.3	Načrtovanje aplikacije	7
2.4	Spletne tehnologije	11
3	Implementacija	21
3.1	Zagon aplikacije v lokalnem okolju	21
3.2	Zasnova projekta	23
3.3	Komunikacija in dokumentacija	25
3.4	Avtentikacija in avtorizacija	30
3.5	Podatkovna baza	33
3.6	Čelni del aplikacije	39
3.7	Razvoj vnosa dinamičnih podatkov	41
4	Predstavitev aplikacije	47
4.1	Uvodna stran	47
4.2	Prijava uporabnika	48
4.3	Urejanje uporabnikov	54

4.4	Vnos publikacije	55
4.5	Prikaz publikacije	61
4.6	Iskanje publikacij	63
5	Sklepne ugotovitve	65
	Literatura	67

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application Programming Interface	aplikacijski programski vmesnik
CSS	Cascading Style Sheets	predloge za izgled spletnih strani
ER diagram	Entity Relationship Diagram	entitetno relacijski diagram
EB	Exabyte	Eksabajt
HTML	Hyper Text Markup Language	označevalni jezik za izdelavo spletnih strani
HTTP	Hyper Text Transfer Protocol	komunikacijski spletni protokol
IAAA	Identification, Authentication, Authorization and Accounting	varnostni koncepti: identifikacija, avtentikacija, avtorizacija in odgovornost
JSON	JavaScript Object Notation	objektna notacija za JavaScript
REST	Representational state transfer	predstavitveni prenos stanja
SQL	Structured Query Language	strukturiran poizvedovalni jezik
URL	Uniform Resource Locator	enolični krajevnik vira

Povzetek

Naslov: Spletni pregled in urejanje raziskovalnih in drugih publikacij

Avtor: Žiga Marolt

Dandanes se na spletu pojavlja vedno več raziskovalnih metod iz področja analize in napovedovanja dobe uporabnosti hitro pokvarljivih živil. Vsaka raziskovalna metoda lahko pri raziskovanju uporablja različne parametre, ki se razlikujejo glede na format, strukturo ali pa samo vrsto podatkov.

Namen te diplomske naloge je izdelati spletno aplikacijo s primerno podatkovno shemo za pregled in urejanje publikacij. Preko nje je mogoče dodajati, urejati, brisati in iskati po vnesenih metodah in drugih podatkih. Dostop do posameznih delov aplikacije je dovoljen le določenim uporabnikom z ustreznimi uporabniškimi pravicami. Predstavljen je celoten proces načrtovanja in implementacije aplikacije, ki je zasnovana kot mikrostoritev v programskem jeziku Go. Podane so zahteve, ki jih je potrebno upoštevati pri zasnovi arhitekture spletne aplikacije iz vidika varnosti podatkov, hitrosti razvoja in hitrosti aplikacije. Predstavljena je s primerom uporabe testnih podatkov za publikacije s področja hladne verige.

Ključne besede: spletna aplikacija, publikacije, Vue.js, docker, OpenApi.

Abstract

Title: Online review and editing of research and other publications

Author: Žiga Marolt

Nowadays, more and more research methods are appearing in the field of analysis and prediction of perishable foods shelf-life. Each research method may use different parameters in its research, which often vary according to format, structure or just the type of data.

The aim of this thesis is to develop a web-based application with a suitable data structure for reviewing and editing the publications. Through the application it is possible to add, edit, delete and search through the entered methods and other data. Access to individual parts of the application is restricted to specific users with the appropriate user rights. Presented is the complete design process and implementation of the application, which is designed as a microservice in the Go programming language. Given are requirements to be taken into account in the design of the web application architecture in terms of data security, development time and the speed of the application itself. The application is presented with an example of test data for publications in the field of cold chain.

Keywords: web application, publications, Vue.js, docker, OpenApi.

Poglavje 1

Uvod

Poleg splošnih informacij je eden od najpomembnejših podatkov, ki so na voljo za posamezna živila, predvsem za hitro pokvarljiva, zapis o roku uporabnosti. Podaja nam informacijo, koliko časa lahko neko živilo hranimo, da varnost in kakovost izdelka ostaneta v sprejemljivem območju. Ob tem morajo biti upoštevani specifični pogoji transporta in shranjevanja. Določitev roka uporabnosti živila temelji na predpostavki, da z živilom ustrezno ravnamo.

Iz dneva v dan nam je na voljo vedno več objavljenih raziskav iz področja ravnanja z živilom v preskrbovalni verigi. Raziskave so objavljene v različnih virih, naj si bo to članek na svetovnem spletu, v reviji ali pa v znanstveni knjigi. Raziskovalci, ki prebirajo raziskave pogosto porabijo veliko časa, da pridobijo zanesljive vire in iz njih uspejo pridobiti pomembne in zanesljive informacije.

Da bi poenostavili pridobivanje informacij, smo v diplomski nalogi zasnovali in implementirali aplikacijo za vnos in pregled raziskovalnih publikacij s področja analize in napovedovanja dobe uporabnosti hitro pokvarljivih živil. Aplikacija je zasnovana kot mikrostoritev. Za razvojno okolje je uporabljena Docker arhitektura, kar nam omogoča enostavno postavitve projekta, in pa tudi enostavno objavljjanje projekta v izbranem okolju.

Ker se struktura podatkov razlikuje od raziskave do raziskave in od vira

do vira, je pomembno, da podatkovno strukturo pravilno definiramo. Za dobro uporabniško izkušnjo mora biti vnos podatkov o raziskavah enostaven in dosleden, iskanje po le-teh pa mora biti hitro in uporabno. Definirali smo več vrst uporabnikov, ki imajo različne dostope do posameznih delov aplikacije.

V diplomski nalogi bomo predstavili kako smo pristopili k reševanju omenjenega problema, kakšen je bil proces dela in sam razvoj aplikacije. Na koncu bomo spoznali še končno rešitev in samo implementacijo.

Poglavje 2

Razvoj aplikacije

Pred izdelavo diplomskega dela smo najprej na spletu preverili ali obstajajo podobne rešitve na trgu. S tem smo razmislili ali je problem sploh smiseln in vreden razvoja. Med raziskavo smo odkrili nekaj orodij, ki so podobna naši aplikaciji. To so Food Safety Centre [1], Mendeley [2] in pa Zotero [3]. Omenjena orodja nam do neke mere olajšajo delo z organizacijo publikacij, vendar pa nobeno popolnoma ne ustreza našim zahtevam.

2.1 Življenjski cikel razvoja

Razvoj programske opreme je proces, ki je sestavljen iz vrste načrtovanih dejavnosti ali sprememb. Življenjski cikel razvoja je drugačen za vsak projekt, zato je potrebno izbrati primeren model, ki omogoča boljši pregled nad samim procesom razvoja programske opreme, kot tudi testiranja in definiranja le-te.

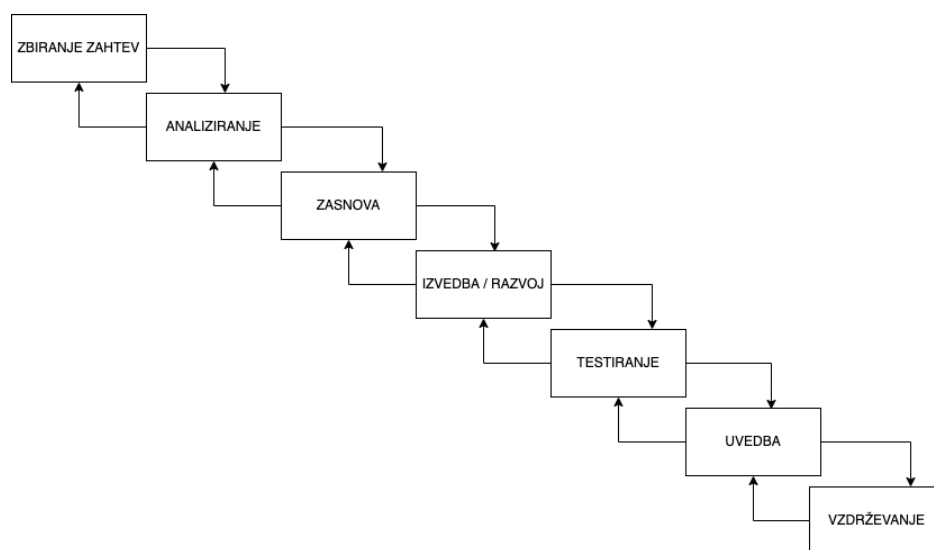
Za razvoj aplikacije smo uporabili „slapovni model“ (ang. waterfall model). Omenjena metodologija je linearni model, pri katerem napredek pretežno teče v eno smer navzdol skozi faze zbiranja potreb, analize, načrtovanja, razvoja, testiranja, uvajanja in vzdrževanja. Izraz je bil prvič uveden v dokumentu, ki ga je leta 1970 objavil dr. Winston W. Royce, in se še naprej uporablja v aplikacijah industrijskega oblikovanja [4].

Omenjeni model je definiran z zaporedjem posameznih faz, katerih število je lahko različno glede na izvedbo metodologije, giblje pa se med pet in sedem različnimi fazami (Slika 2.1). Izhod ene faze se uporablja kot vhod naslednje faze, torej mora biti vsaka faza zaključena, preden se lahko začne naslednja.

Opis posameznih faz:

1. **Zbiranje zahtev:** vse možne zahteve so zajete v dokumentih z opisom izdelka.
2. **Analiziranje:** pregled specifikacij in analiziranje le-teh. Z analiziranjem se definira poslovno logiko in pa tudi finančni plan projekta.
3. **Zasnova sistema:** glede na predhodno analizo se načrtuje arhitekturo programske opreme.
4. **Izvedba:** razvoj programske opreme v manjših enotah s funkcionalnim testiranjem.
5. **Integracija in testiranje:** integracija vsake enote, razvite v prejšnji fazi in po integraciji, katerim sledi testiranje celotnega sistema za morebitne napake.
6. **Uvedba sistema:** po opravljenih vseh funkcionalnih in nefunkcionalnih testiranjih izdelek deluje v proizvodnem okolju.
7. **Vzdrževanje:** odpravljanje težav in izdaja nove različice s popravki se izvaja po potrebi.

Zaradi medsebojne odvisnosti posameznih faz je model pri implementaciji zelo pregleden in jasen. Vendar ima nekaj pomanjkljivosti in je učinkovit le pri manjših projektih, kjer so zahteve zelo jasno definirane in kjer je predviden obseg dela manjši. Model ne dovoljuje spreminjanja specifikacij, oziroma je spreminjanje zahtev med samo implementacijo oteženo. Ko je izdelek v fazi testiranja, se je težko vrniti in spremeniti nekaj, kar je ostalo v fazi analize. To pomeni, da delujoč izdelek dobimo šele na koncu cikla omenjenega modela.



Slika 2.1: Vizualizacija sedmih faz slapovnega modela.

2.2 Pregled zahtev

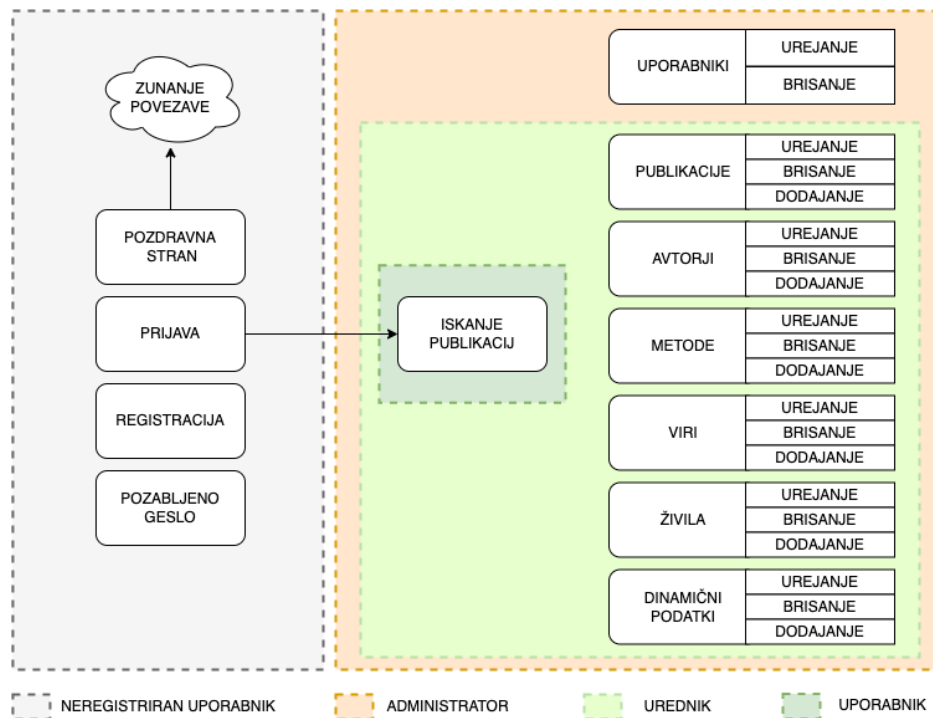
Cilj izdelave spletne aplikacije je agregirati podatke iz različnih virov, kot so knjiga, spletni vir ali pa revija. Vnašanje omenjenih podatkov iz nedefiniranih virov je težko in zamudno opravilo. Aplikacija mora biti razvita z namenom, da je uporabniku, ki publikacije vnaša v sistem, delo olajšano.

Aplikacija naj omogoča vnos novih in urejanje obstoječih publikacij. Vsak vnos mora biti pred objavo potrjen s strani administratorja. vnesene publikacije naj bodo primerno prikazane končnemu uporabniku, z možnostjo iskanja po posameznih zapisih.

Aplikacija mora biti pregledna in enostavna za uporabo. Podpirati mora registracijo novih uporabnikov v aplikacijo in omejiti dostop do določene vsebine glede na vlogo uporabnika. Dostop do posameznih strani naj bo omogočen le prijavljenim uporabnikom z uporabniškim imenom in geslom ter primernim dostopom (Slika 2.2). Definirani uporabniki so:

- **Uporabnik:** omogočen je pregled objavljenih publikacij in pripadajočih vsebin.

- **Urednik:** lahko ureja publikacije, vsebine ter parametre za posamezno publikacijo.
- **Administrator:** ima polni nadzor aplikacije.



Slika 2.2: Pregled dostopa do posameznih strani in akcij glede na vlogo uporabnika.

Zaledni sistem naj bo zasnovan tako, da ga bo mogoče nadgraditi in kasneje uporabiti v mobilni aplikaciji. Uporabniški vmesnik aplikacije naj bo v angleščini, komunikacija med čelnim in zalednim sistemom pa naj bo ustrezno dokumentirana.

2.3 Načrtovanje aplikacije

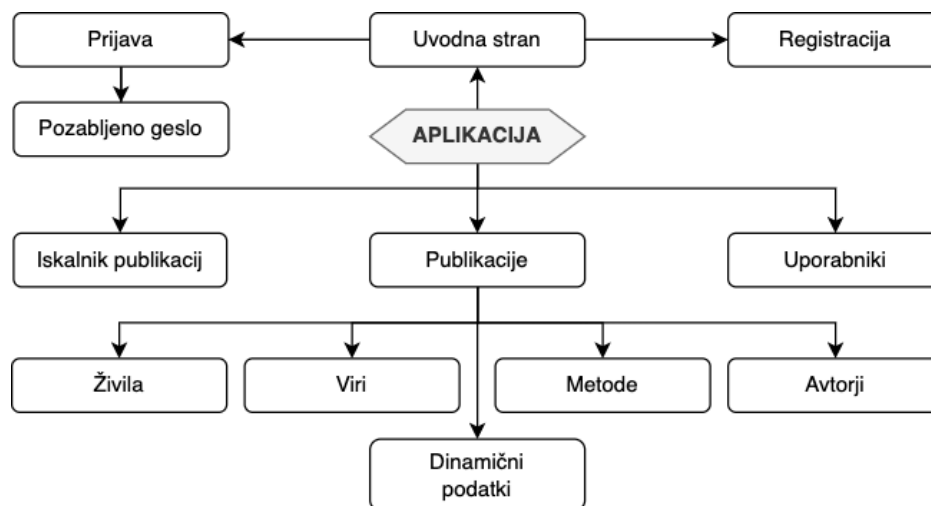
Na podlagi zahtev smo začeli z načrtovanjem aplikacije. Najprej smo popisali vse funkcionalnosti, katere mora aplikacija zajemati in razdelali strukturo uporabniškega vmesnika. Glede na popisane funkcionalnosti smo se odločili kako bo projekt postavljen in definirali arhitekturo aplikacije.

Med načrtovanjem smo se odločili tudi kateri programski jezik bomo uporabili. Definirali smo, kako bo potekala komunikacija med posameznimi internimi storitvami in komunikacijo med zalednim in čelnim delom.

V nadaljevanju bomo spoznali posamezne tehnologije in metode, uporabljene pri razvoju aplikacije in čemu je katera tehnologija namenjena.

2.3.1 Funkcionalnosti in njihove povezave

Aplikacija je sestavljena iz sklopov strani, do katerih lahko dostopamo iz različnih delov aplikacije. Za lažjo predstavitev so na sliki 2.3 prikazane povezave med posameznimi stranmi, med katerimi se uporabnik lahko premika. Opisi zahtev za posamezne strani so naslednji:



Slika 2.3: Zemljevid aplikacije (ang. sitemap).

Registracija: ustvarjanje novega uporabnika

Prijava: obrazec za overjanje uporabnika

Pozabljeno geslo: nastavitev novega gesla uporabniku

Uporabniki: seznam vseh uporabnikov

- urejanje: obrazec za urejanje uporabnika
- brisanje: deaktiviranje uporabnika

Publikacije: seznam vnesenih publikacij

- urejanje: obrazec za urejanje obstoječe publikacije
- brisanje: odstranitev publikacije
- dodajanje: obrazec za kreiranje nove publikacije
- iskanje: iskanje po naslovu
- prenos datoteke: prenos pripete datoteke k publikaciji

Metode: seznam vseh vnesenih metod

- urejanje: obrazec za urejanje obstoječe metode
- brisanje: odstranitev metode
- dodajanje: obrazec za kreiranje nove metode

Viri: seznam vseh dodanih virov

- urejanje: obrazec za urejanje obstoječega vira
- brisanje: odstranitev vira
- dodajanje: obrazec za kreiranje novega vira

Živila: seznam vseh dodanih živil

- urejanje: obrazec za urejanje obstoječega živila
- brisanje: odstranitev živila

- dodajanje: obrazec za kreiranje novega živila

Avtorji: seznam vseh dodanih avtorjev

- urejanje: obrazec za urejanje obstoječega avtorja
- brisanje: odstranitev avtorja
- dodajanje: obrazec za kreiranje novega avtorja

Dinamični podatki: seznam vseh definiranih dinamičnih podatkov

- urejanje: obrazec za urejanje obstoječih podatkov
- brisanje: odstranitev podatka
- dodajanje: obrazec za kreiranje vnosa novih podatkov

Iskalnik: stran za prikaz in iskanje publikacij

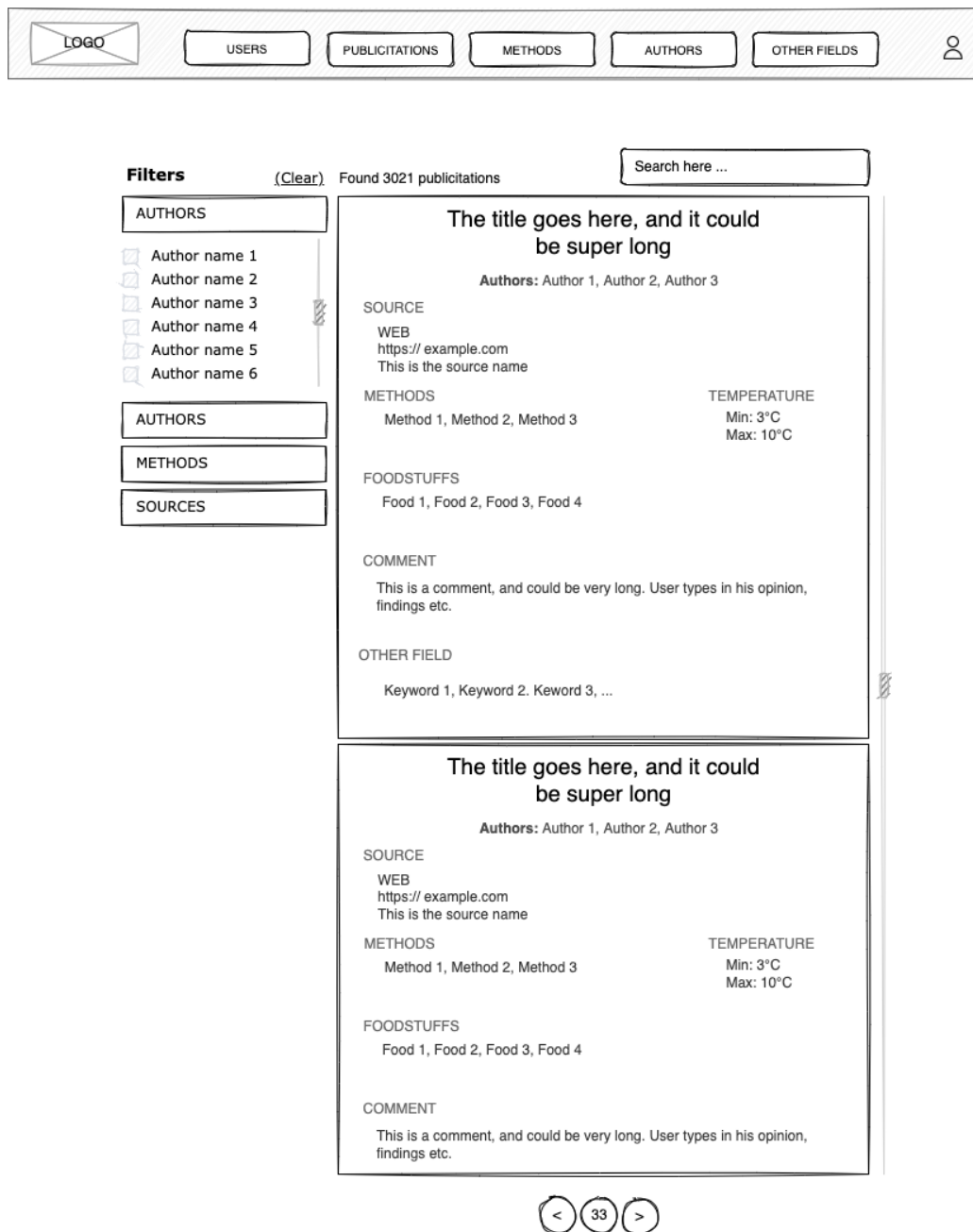
2.3.2 Postavitev spletne strani

Osnovno postavitev posameznih strani smo najprej skicirali na list papirja. Odločili smo se za postavitev, ki se v spletnih aplikacijah zelo pogosto uporablja in je enostavna in pregledna.

Na vrhu strani se nahaja navigacija, katere namen je premikanje med posameznimi sklopi. Na skrajnem levem delu se nahaja ime aplikacije „Safe Food“. Na skrajnem desnem delu se nahaja gumb za nadzor nad uporabniškim delom. Na sredini navigacije pa se nahajajo povezave do posameznih sklopov aplikacije.

Na strani, kjer uporabnik pregleduje in išče publikacije, smo se odločili, da stran razdelimo na dva dela. Na levem delu strani se nahaja nabor filtrov, na desnem delu pa prikaz publikacij (Slika 2.4).

Vse strani uporabljajo enak koncept. Za enostavno premikanje med stranmi se na dnu nahaja paginacija. V primeru napake pri vnosu podatkov se ta prikaže poleg vnosnega polja. V primeru generalne napake, kot je na primer neuspešno poslan zahtevak na zaledni sistem, pa se ta izpiše na dnu strani v posebnem oknu.



Slika 2.4: Skica postavitve strani s publikacijami (ang. wireframe).

2.4 Spletne tehnologije

2.4.1 Nadzor različic

Orodje, ki upravlja in sledi različnim verzijam programske kode ali drugih podatkov, je poznano kot sistem za verzioniranje podatkov. V angleščini poznamo več izrazov, ki se navezujejo na omenjeno orodje, to so Version Control System (VCS), Source Code Manager (SCM) in Revision Control System (RCS). Nadzor različic, znan tudi kot nadzor vira, je praksa sledenja in upravljanja sprememb programske kode. Sistem običajno hrani kodo na dogovorjeni lokaciji na centralnem strežniku (primer GitHub). Razvijalci delajo na svojih lokalnih kopijah izvirne kode, ki jih pridobijo iz centralnega repozitorija. Svoje spremembe pošiljajo na centralni strežnik, pri čemer sistem omogoča tudi reševanje konfliktov, ko dva ali več razvijalcev poskuša poslati na strežnik spremembe istega dela posamične datoteke. Sistem omogoča obnavljanje stanja izvirne kode iz poljubne verzije v preteklosti.

V našem projektu smo se odločili uporabiti sistem za verzioniranje kode Git. Je distribuiran sistem s poudarkom na hitrosti, integriteti podatkov in podpira vzporedne nelinearne tokove dela. Predstavljen je bil leta 2005, za potrebo razvoja Linuxovega jedra iz strani Linux razvijalcev [5] in je postal najbolj razširjen sistem na tem področju. Git repozitorij gostujemo na brezplačnem ponudniku omenjene storitve - GitHub (Slika 2.5). Poleg brezplačnega gostovanja ponuja tudi vrsto drugih rešitev, kot je na primer beleženje dela (ang. task management), neprekinjeno integracijo (ang. Continuous Integration - CI) in vrsto drugih integracij.

2.4.2 Infrastruktura

Uporaba programske opreme je zapletena. Pred namestitvijo je potrebno razmisliti, kateri operacijski sistem se bo uporabljal, katera so orodja, ki jih programska oprema potrebuje in še vrsto drugih vprašanj. Večina računalnikov že ima nameščene in zagnane aplikacije, ki so odvisne od drugih aplikacij.

The screenshot shows a GitHub repository page. At the top, it displays the repository name 'Ziga Marolt dynamic field validation updates', the commit count '71 commits', and the last commit date 'on 10 Dec 2021'. Below this is a table of files and folders with their commit dates. The right sidebar contains sections for 'About', 'Releases', 'Packages', and 'Languages'.

File/Folder	Description	Commit Date
.github	playing with github ci pipelines	12 months ago
docker	bump golang version to 1.17.0	1 month ago
internal	dynamic field validation updates	1 month ago
scripts	added OpenApi build scripts	1 month ago
specs	dynamic field validation updates	1 month ago
web	design update	1 month ago
.editorconfig	Basic project structure init	13 months ago
.env	docker configuration updates	1 month ago
.gitignore	Basic project structure init	13 months ago
Makefile	docker configuration updates	1 month ago
README.md	Basic project structure init	13 months ago
docker-compose.yml	docker configuration updates	1 month ago
docker.env	docker configuration updates	1 month ago

About
No description, website, or topics provided.

Releases
No releases published
[Create a new release](#)

Packages
No packages published
[Publish your first package](#)

Languages

Language	Percentage
Go	56.7%
JavaScript	11.7%
Shell	0.9%
Other	0.4%
Vue	27.8%
Makefile	1.8%
HTML	0.7%

Slika 2.5: Prikaz GitHub repozitorija.

V primeru, da nekatere aplikacije med seboj niso kompatibilne, lahko pride do težav, ki jih ni enostavno odpraviti. Stvari postanejo bolj zapletene, če si aplikacije med seboj delijo skupne vire.

Na našem projektu smo se hoteli izogniti nevšečnostim s postavljanjem projekta, zato smo se odločili uporabiti orodje za izoliranje okolja. Virtualizacija je postopek, ki strojni ali programski vir preslika v navidezni vir, tega pa potem odjemalec koristi kot pravi vir. Pomembna prednost uporabe virtualizacije je prenosljivost. Z njo lahko dosežemo enake pogoje za izvajanje programske opreme na različnih strojnih opremah [6].

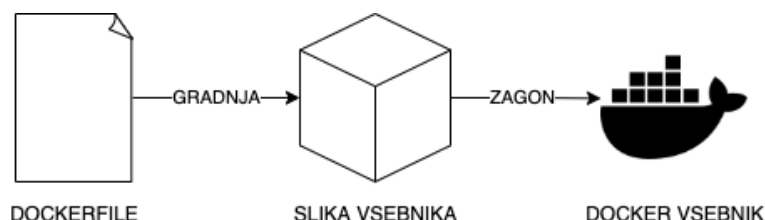
Orodje Docker zagotavlja tako imenovano abstrakcijo, ki nudi enako okolje vsem razvijalcem, ki razvijajo na istem projektu, ne glede na to, kateri operacijski sistem uporabljajo. Vsebniki (ang. containers) uporabljajo jedra operacijskega sistema in se obnašajo kot navadni programi, ki lahko uporabljajo vse sistemske vire, vendar omejijo dostop aplikacije samo na prostor v vsebnikih.

Docker

Docker je trenutno najbolj priljubljena rešitev vsebnika. Ponuja številne funkcije in je podprt s strani ostalih sistemov, kot so na primer orodja za orkestracijo (ang. orchestration). V osnovi gre za izolacijo procesov in virtualizacijo virov, do katerih procesi dostopajo. Omogoča izoliranje aplikacije od infrastrukture, kar olajša hitro dostavljanje programske opreme [7].

Docker je odvisen od jedra Linuxa, kar pomeni, da ne deluje v sistemu Windows ali macOS. V tem primeru je potreben zagon v virtualnem stroju, ki vsebuje jedro Linuxa [6].

Konfiguriranje slike vsebnika zahteva ustvarjanje konfiguracijske datoteke. Ta je ponavadi znotraj projekta in se imenuje „Dockerfile“. To je tekstovna datoteka, ki opisuje zahteve posameznega programa, kot so osnovna slika za izvajanje (npr. slika ki omogoča izvajanje Go programov), ukazi za zagon (npr. namestitvev), vrata na katerih bo aplikacija poslušala in tako naprej (Slika 2.6).



Slika 2.6: Prikaz gradnje Docker vsebnikov.

Vsebnik je skupina procesov v operacijskem sistemu z jedrom Linux, ki upravlja računske vire z nadzornimi skupinami in zagotavlja izolacijo virov z uporabo imenskih prostorov. Ob zagonu vsebnika se uporabi vsebniška slika (ang. container image), v kateri so zapisane informacije o izvajanju programske opreme in preko katere Docker tudi priklopi nov, prazen bralno-pisalni datotečni sloj. Slika se ustvari s postopkom gradnje, ki je zapisan v datoteki *Docker file*.

V kodi 2.1 je prikazan primer grajenja slike, ki smo jo uporabili pri razvoju

zalednega dela naše aplikacije.

```

1 $ docker build .
2 [+] Building 69.1s (9/9)                                FINISHED
3 => [internal] load build definition from Dockerfile      0.1s
4 => => transferring dockerfile: 180B                      0.0s
5 => [internal] load .dockerignore                        0.0s
6 => => transferring context: 2B                          0.0s
7 => [internal] load metadata for golang:1.17             2.7s
8 => [1/2] FROM golang:1.17@sha256:39953c7c7              60.0s
9 => => resolve golang:1.17@sha256:39953c7c7              0.0s
10 => => sha256:0e29546d54 54.92MB / 54.92MB              27.2s
11 => => extracting sha256:0e29546d541                   4.5s
12 => => sha256:6432567af305a4304605a3 154B / 154B      34.4s
13 => [internal] load build context                       0.0s
14 => => transferring context: 121B                      0.0s
15 => [2/2] COPY start.sh /                             0.0s
16 => exporting to image                                  0.2s
17 => => exporting layers                                  0.2s
18 => => writing image sha256:96d781a9030                 0.0s

```

Koda 2.1: Grajenje slike Docker vsebnika.

2.4.3 Spletni strežnik

Spletni strežnik (ang. Web Server) je računalniški sistem, ki obdeluje zahteve preko protokola HTTP. Izraz spletni strežnik se lahko nanaša na celoten sistem ali posebej na programsko opremo, ki sprejema in nadzira zahteve HTTP. Primarna funkcija spletnega strežnika je shranjevanje, obdelava in pošiljanje spletnih strani odjemalcem. Predložene strani so najpogostejše dokumenti HTML, ki poleg besedilne vsebine vključujejo tudi slike, CSS predloge in skripte.

Glavna naloga spletnega strežnika je prikazovanje vsebine spletne strani. Če spletni strežnik ni izpostavljen javnosti in se uporablja interno, se imenuje „intranetni strežnik“. Strojna oprema spletnega strežnika je povezana z internetom in omogoča izmenjavo podatkov z drugimi povezanimi napra-

vami, medtem ko programska oprema spletnega strežnika nadzoruje, kako uporabnik dostopa do gostiteljskih datotek.

Do programske opreme spletnega strežnika se dostopa preko domenskih imen spletnih mest in zagotavlja dostavo vsebine spletnega mesta uporabniku, ki je poslal zahtevek. Strežnik HTTP lahko razume zahteve HTTP in povezave z enoličnimi krajevnički virov (URL-ji). Kot strojna oprema je spletni strežnik računalnik, ki shranjuje programsko opremo spletnega strežnika in druge datoteke, povezane s spletnim mestom, kot so dokumenti HTML, slike in datoteke JavaScript.

2.4.4 Programski jezik Go

Go je odprtokodni programski jezik, razvit iz strani ameriške korporacije Google. Pobudniki razvoja so bili Robert Griesemer, Ken Thomson in Rob Pike. Sprva je bil uporabljen le za interno uporabo, leta 2009 pa so ga ponudili širši publiki kot odprtokodni programski jezik.

Kljub temu, da je programski jezik Go namenjen splošni uporabi, je njegova primarna uporaba namenjena pisanju sistemskih orodij, spletnih storitev in programov, ki veliko komunicirajo preko spletnega omrežja. Zaradi njegove enostavnosti in pristopov je primeren tudi za učenje prvega programskega jezika. Definiranih, oziroma rezerviranih je le 25 ključnih besed, kar pomeni, da si je jezik veliko lažje zapomniti in se je potrebno naučiti le konceptov.

Čeprav Go ni objektno usmerjen programski jezik, so njegovi vmesniki zelo vsestranski in omogočajo posnemanje nekaterih zmožnosti objektno usmerjenih jezikov, kot so polimorfizem, enkapsulacija in sestava.

Go ima tudi zmožnosti sočasnosti z uporabo preprostega modela sočasnosti, ki se izvaja z uporabo go-rutin in kanalov. Go upravlja niti operacijskega sistema namesto nas in ima zmogljiv izvajalni čas. To omogoča ustvarjanje lahkih delovnih enot (ang. goroutine), ki med seboj komunicirajo s pomočjo kanalov.

2.4.5 Mikrostoritev

Arhitekturni koncept mikrostoritev je pristop k razvoju ene same aplikacije kot zbirke majhnih storitev, od katerih vsaka deluje v svojem procesu in komunicira z enostavnimi mehanizmi, kot je na primer HTTP API. Te storitve so zgrajene na podlagi poslovnih zmogljivosti in jih je mogoče neodvisno uvesti s popolnoma avtomatiziranimi stroji za uvajanje. Obstaja minimalno centralizirano upravljanje teh storitev, ki so lahko napisane v različnih programskih jezikih in uporabljajo različne tehnologije za shranjevanje podatkov [8].

2.4.6 Podatkovna baza

Za shranjevanje podatkov smo se odločili za relacijsko podatkovno bazo PostgreSQL. Relacijska podatkovna baza (ang. Relational Database Management System - RDBMS) uporablja podatkovno strukturo, ki nam omogoča identifikacijo in dostop do podatkov preko relacije med dvema entitetama. Podatke v relacijski podatkovni bazi si lahko predstavljamo kot tabele s stolpci in vrsticami [9].

PostgreSQL

PostgreSQL je zmogljiv, odprtokoden, objektno-relacijski sistem baz podatkov, ki uporablja in razširja jezik SQL v kombinaciji s številnimi funkcijami, ki varno shranjujejo in spreminjajo podatke. Začetki PostgreSQL segajo v leto 1986 kot del projekta POSTGRES na kalifornijski univerzi v Berkeleyju in ima več kot 30 let aktivnega razvoja na osnovni platformi.

PostgreSQL si je prislužil močan sloves s svojo dokazano arhitekturo, zanesljivostjo, celovitostjo podatkov, robustnim naborom funkcij, razširljivostjo in predanostjo skupnosti odprte kode, ki stoji za programsko opremo, da dosledno zagotavlja zmogljive in inovativne rešitve. PostgreSQL deluje na vseh večjih operacijskih sistemih in ima zmogljive dodatke, kot je na primer priljubljena razširitev geoprostorske baze podatkov PostGIS (ang. Geographic

Information System) [10].

UUID

Za identifikacijo naših entitet smo uporabili enoličen univerzalen identifikator - UUID (ang. Universally Unique Identifier). To je standardna identifikacijska koda, ki se uporablja v postopku izdelave programske opreme. Uporablja se za generiranje univerzalnih unikatnih identifikatorjev, ki omogočajo prepoznavanje in razlikovanje predmeta znotraj sistema ali istega predmeta v različnih kontekstih.

UUID je sestavljen iz 128 bitov (32 znakov). Vsak znak je predstavljen v šestnajstičnem (heksadecimalnem) zapisu, kar pomeni, da je znak lahko število od 0 do 9, ali pa črka od a do f [11].

Glede na ime bi lahko sklepali, da je UUID unikatno glede na čas in prostor. Vendar je končno število vseh kombinacij UUID-jev $n = 2^{122}$. To pomeni, da je verjetnost, da naletimo na dva enako generirana identifikatorja, zelo nizka. Pa vendar, če generiramo množico UUID-jev (r), kjer je število generiranih identifikatorjev večje kot največje število vseh mogočih vrednosti ($r > n$), morajo v množici obstajati duplikati. Verjetnost, da se pojavijo duplikati je mogoče natančno izračunati na podlagi rojstnodnevnega paradoksa [12], ki ga je leta 1932 predstavil Von Mises [13].

Izrek 2.1 *Verjetnost unikatno generiranega UUID-ja*

$$\frac{n!}{n^r(n-r)!} \quad (2.1)$$

Dokaz. Število načinov, da nimamo dvojnikov je $n * (n - 1) * (n - 2) * \dots * (n - (r - 1))$. Kar pomeni, da je lahko prvi UUID katerakoli kombinacija od n možnosti, drugi je lahko katera koli kombinacija od n , razen prvega ($n - 1$), in tako naprej ($n - 2$)... Skupno število načinov za generiranje r UUID-jev je torej n^r , saj ima vsak, od r UUID-jev n različnih kombinacij. S tem je dokaz Izreka 2.1 zaključen. \square

Če nadaljujemo računanje na podlagi rojstnodnevnega paradoksa, pridemo do rešitve, ki nam pove, da je verjetnost resnično majhna. Da se

pojavi duplikat, bi morali 85 zaporednih let vsako sekundo generirati milijardo vrednosti. Datoteka z generiranimi UUID-ji bi bila na koncu velika $45EB$ (ang. Exabyte) ($45kb * 1000^6$) [14].

Minio

Minio je samostojna rešitev za izdelavo lastne hrambe objektov. Je alternativa za bolj poznani Amazonovi storitvi AWS S3.

Programska oprema Minio je na voljo kot preprost binarni dokument in celo uradna dokumentacija kaže, da ga uporabljajo na tak način, namesto upravitelja paketov. Obstajajo tudi Dockerjeve slike, katere je mogoče uporabiti za zagon Minio strežnika kot Docker vsebnik.

Minio je bolj primeren za shranjevanje nestrukturiranih podatkov kot so fotografije, videoposnetki, dnevniške datoteke, varnostne kopije in slike vsebnikov / VM. Velikost predmeta se lahko giblje od nekaj KB do največ 5 TB. Storitev uporabljamo za potrebe po shranjevanju datotek, ki jih urednik lahko pripne zraven publikacije.

Firestore

Firestore je BaaS (Backend as a Service) storitev, ki jo ponuja podjetje Google. Z orodji, ki jih zajema, je programerjem olajšano delo pri razvoju in pa tudi pri vzdrževanju aplikacije. Orodja, ki jih nudijo, so orodje za avtentikacijo, testiranje, za obveščanje strank in ostala infrastrukturna orodja kot na primer podatkovna baza in gostovanje [15].

Za potrebe naše aplikacije smo uporabili Firestore avtentikacijo. Avtentikacija temelji na žetonih in zagotavlja izključene integracije z najpogostejšimi ponudniki, kot so Google, Facebook, Twitter in ostali. Omogoča nam uporabo zahtevkov po meri, ki jih bomo izkoristili za izgradnjo prilagodljivega API-ja, ki temelji na vlogah. V zahteve lahko nastavimo katero koli vrednost JSON (npr. { "vloga": 'administrator' } ali { "vloga": 'urednik' }). Nastavljeni zahtevki so zapisani v žetonu, ki ga generira avtentikacijska storitev.

Brezplačni plan Firebase omogoča kreiranje neomejenega števila uporabnikov. Omejeno je le število registriranih in izbranih uporabnikov v časovnem obdobju [16]:

- **Število registriranih uporabnikov** - neomejeno
- **Hitrost brisanja uporabnikov** - 10 uporabnikov/sekundo
- **Hitrost kreiranja uporabnikov** - 100 uporabnikov/IP/uro

2.4.7 Vue.js

Vue je odprtokodno, progresivno JavaScript ogrodje (ang. framework), namenjeno izdelavi uporabniških vmesnikov in enostranskih aplikacij.

Vue.js ima postopoma prilagodljivo arhitekturo, ki se osredotoča na deklarativno upodabljanje in sestavo komponent. Jedrna knjižnica je osredotočena samo na vizualni sloj. Napredne funkcije, potrebne za zapletene aplikacije, kot so usmerjanje, upravljanje stanja in orodja za gradnjo, so na voljo prek uradno vzdrževanih podpornih knjižnic in paketov [17].

2.4.8 Tailwind

Tailwind je ogrodje, ki ponuja CSS gradnike za lažjo in hitrejšo izdelavo spletnih aplikacij. Temelji na slogovnem jeziku CSS oziroma kaskadnih stilskih podlogah (ang. cascading style sheets). Vsebuje številne elemente, katere redno uporabljamo za oblikovanje HTML gradnikov, kot so osnovna razdelitev strani, posamezni gumbi, spustni meniji in ostali gradniki s katerimi oblikujemo stran. Ogrodje samo po sebi poskrbi za prilagodljiv izgled na različnih napravah, kar ponavadi pri razvoju vzame veliko časa.

Ogrodje je napisano s pomočjo *PostCSS* orodja, kar nam omogoča enostavno konfiguriranje posameznih slogovnih elementov.

Poglavje 3

Implementacija

V tem poglavju bomo spoznali vpeljavo spletnih in drugih tehnologij ter si pogledali kako je zgrajen strežniški del in nato še uporabniški del naše aplikacije. Strežniški del skrbi za obdelavo podatkov, uporabniški del pa uporabniku omogoča enostaven vnos in prikaz podatkov.

Najprej bomo opisali kako smo projekt zasnovali in vpeljali različne arhitekturne koncepte in vzorce. Nato bomo opisali kako poteka komunikacija med uporabniškim in strežniškim delom aplikacije. Na koncu pa bomo spoznali še kako so posamezni podatki predstavljeni in shranjeni v ustreznih podatkovnih bazah.

3.1 Zagon aplikacije v lokalnem okolju

Postavitev projekta je lahko včasih zelo zamudno delo, s katerim razvijalec izgubi veliko nepotrebnega časa. Za zagon projekta mora razvijalec imeti nameščena orodja `Makefile`, `Docker` (2.4.2) in `Git` (2.4.1). Z uporabo vsebnikov smo ustvarili okolje, katerega je enostavno zagnati ne glede na operacijski sistem, kjer želimo aplikacijo zaganjati.

Za prenos projekta na svoj računalnik mora razvijalec imeti dostop do `Git` repozitorija, katerega gostimo na platformi `GitHub`. Z ukazom (Koda 3.1), ki ga poženemo v ukazni vrstici poskrbimo, da so vse datoteke iz `GitHub`

repozitorija uspešno prenesene v direktorij na računalniku.

```
1 $ git clone git@github.com:marolt/diploma.git
```

Koda 3.1: Ukaz za prenos datotek iz GitHub repozitorija.

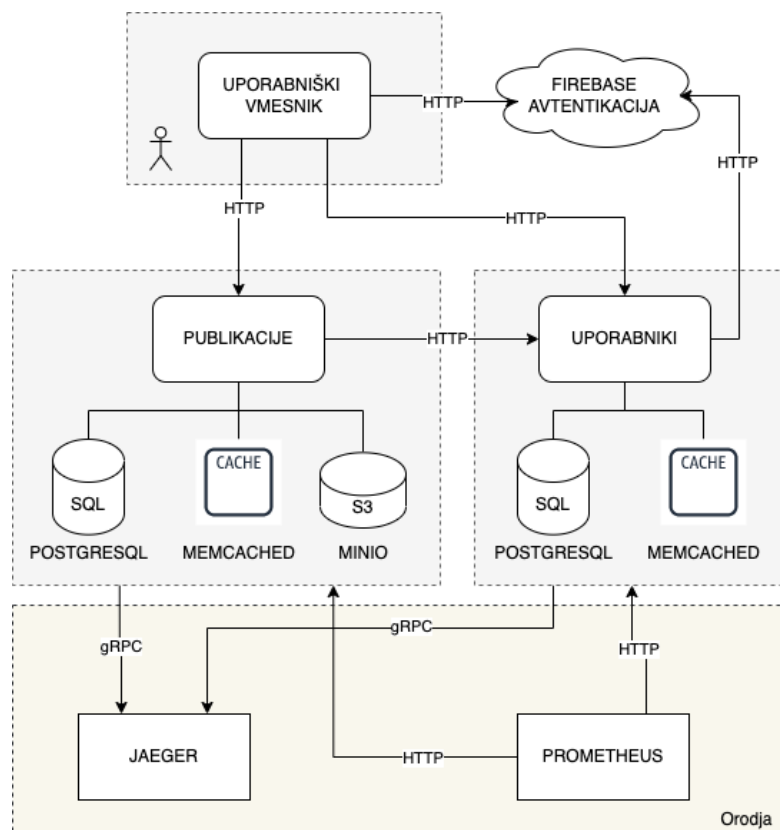
Da je postavitve projekta enostavna, smo ukaze, ki so potrebni za zagon projekta, ovili z orodjem **Makefile**. S poganjanjem ukaza **make start** se izvede zaporedje ukazov, ki poskrbijo, da so vsi potrebni vsebniki dosegljivi. Poskrbi, da sta koda odjemalca in koda strežnika generirani na podlagi **OpenApi** specifikacij, in da so vse zunanje knjižnice, ki jih aplikacija potrebuje pri delovanju, uspešno nameščene. Vsebniki so nastavljeni tako, da poslušajo spremembe v kodi, kar pomeni, da je vsaka sprememba takoj razvidna v aplikaciji. Ko se zagnan ukaz uspešno izvede, nam je aplikacija na voljo na spletnem naslovu **http://localhost**.

```
1 $ docker-compose ps
2
3 Name                Command                State Ports
4 -----
5 d_web_1              /run.sh               Up    :80->80/tcp
6 d_app_1              reflex -c /reflex.conf Up    :3002->80/tcp
7 d_users_1            reflex -c /reflex.conf Up    :3001->80/tcp
8 d_firestore_1        dockerize -template=... Up    :9099->9099/tcp
9                      8787/tcp
10 d_minio_1            /usr/bin/docker-entry... Up    9000/tcp
11 d_memcached_1        docker-entrypoint.sh ... Up    11211/tcp
12 d_postgres_1         docker-entrypoint.sh ... Up    :5433->5432/tcp
13 d_trace_1            /go/bin/all-in-one-linux Up    14250/tcp
14                      :14268->14268/tcp
15                      :16686->16686/tcp
16 d_teus_1             /bin/prometheus --con... Up    9090/tcp
```

Koda 3.2: Prikaz zagnanih vsebnikov.

Z ukazom **docker-compose ps** (Koda 3.2) lahko razberemo, da se ob postavitvi projekta zažene devet vsebnikov. Ti poskrbijo za streženje vseh potrebnih datotek za popolno delovanje čelnega in zalednega dela aplikacije.

Za lažjo predstavitev celotnega projekta ali zgradbe aplikacije je diagram celotne arhitekture prikazan na sliki 3.1.



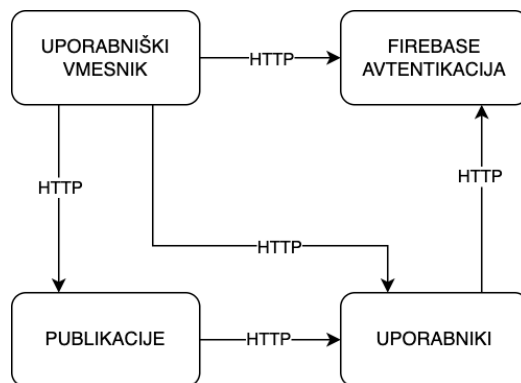
Slika 3.1: Arhitektura projekta predstavljena z diagramom.

3.2 Zasnova projekta

Arhitekturna zasnova močno vpliva na delovanje in razvoj aplikacije. Vsak arhitekturni koncept ima svoje prednosti in slabosti. Na projektu smo želeli predstaviti arhitekturo mikrororitev. Ta vrsta arhitekture definira aplikacijo, sestavljeno iz majhnih samostojnih enot. Vsaka enota ima točno določeno funkcijo in lahko deluje neodvisno od ostalih enot.

V aplikaciji smo definirali dve mikrororitvi (Slika 3.2). Ena storitev skrbi

za delo s publikacijami, druga pa za delo z uporabniki. Storitvi med seboj komunicirata s protokolom HTTP.



Slika 3.2: Arhitektura mikrorstitev naše aplikacije.

Koncept kode sledi heksagonalni arhitekturi (ang. hexagonal architecture). Z omenjeno arhitekturo zagotovimo, da je domenska logika neodvisna od trenutne infrastrukture. Uporabljamo vzorec repozitorija s katerim ločimo poslovno kodo od kode, ki skrbi za pridobivanje in shranjevanje podatkov. Za vsako zbirko podatkov smo definirali vmesnik (ang. Interface). Vmesnik definira metode, ki so lahko različno implementirane glede na infrastrukturo (Koda 3.3). Ta način nam omogoča enostavno spreminjanje implementacije v primeru, ko hočemo zamenjati infrastrukturo podatkovne baze.

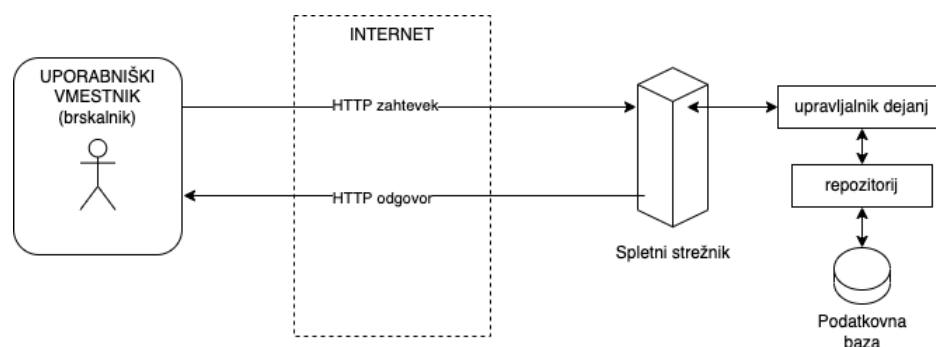
```
1 // ArticleRepository defines the datastore handling
   persisting Article records.
2 type ArticleRepository interface {
3     Create(ctx context.Context, article *Article, callback
       ArticleCallbackFn) (*Article, error)
4     Delete(ctx context.Context, id utils.BINARY16) error
5     Find(ctx context.Context, id utils.BINARY16) (*Article,
       error)
6     Update(ctx context.Context, article *Article, callback
       ArticleCallbackFn) (*Article, error)
7 }
8
```

```
9 // ArticleSearchRepository defines the datastore handling
    searching Article records.
10 type ArticleSearchRepository interface {
11     Search(ctx context.Context, args SearchParams) (
        SearchResults, error)
12 }
```

Koda 3.3: Vmesnik, ki definira repozitorij za publikacije.

3.3 Komunikacija in dokumentacija

Za komunikacijo z zalednim delom aplikacije uporabljamo protokol HTTP (Hypertext Transfer Protocol). HTTP je brezstanjski protokol aplikacijskega sloja, ki skrbi za komunikacijo med odjemalcem in strežnikom. Deluje po protokolu **zahteva-odgovor** (ang. request-response ali request-reply) v modelu odjemalec-strežnik (Slika 3.3). Zahtevki se izvajajo sinhrono, kar pomeni, da je povezava med odjemalcem in strežnikom odprta toliko časa, dokler strežnik ne odgovori s podatki, oziroma do neke časovne omejitve [18].



Slika 3.3: Prikaz komunikacije po protokolu zahteva-odgovor.

Za komunikacijo z zalednim delom aplikacije smo definirali aplikacijski programski vmesnik (API), z uporabo protokola HTTP. Vmesnik je arhitekturno predstavljen kot REST API, kar pomeni, da je vsak vir predstavljen

kot spletna storitev z enoznačnim naslovom URL. Za pridobivanje in urejanje podatkov uporabljamo standardne metode HTTP: GET, POST, PUT, DELETE. Sistemi, ki uporabljajo REST pristop stremijo k hitri, odzivni in stabilni komunikaciji med odjemalcem in strežnikom.

Primer definiranih virov za pridobivanje, urejanje, brisanje in kreiranje publikacij v sistemu:

- **GET** /articles/list - Vrni seznam publikacij
- **POST** /articles/add - Shrani novo publikacijo
- **GET** /articles/:uuid/get - Vrni publikacijo za dodeljen ID
- **DELETE** /articles/:uuid/delete - Izbriši publikacijo za dodeljen ID
- **PUT** /articles/:uuid/update - Osveži obstoječo publikacijo
- **POST** /articles/:uuid/upload-files - Pripni datoteko k publikaciji
- **DELETE** /articles/:uuid/files/:file-uuid - Odstrani datoteko publikacije

Pri definiranju komunikacijskega vmesnika smo sledili znanemu standardu **OpenApi 3.0**. Ustvarili smo specifikacijske datoteke, kjer je definiran vsak klic, ki ga je mogoče izvesti na zaledni del. Te specifikacije nam pomagajo pri grajenju dokumentacije (Slika 3.4), v našem primeru pa na podlagi API specifikacij tudi generiramo kodo, katera skrbi za komunikacijo med odjemalcem in strežnikom. Generirana koda je enostavna za uporabo in je postala dobra praksa saj s tem tudi zagotovimo, da je dokumentacija komunikacijskega vmesnika ves čas pravilna.

Publication Manager API 0.0.1 OAS3

The API to manage publications. It allows you to create, edit, delete and get a list of all of the publications with applying various filters. It provides you also the ability to create dynamic fields, to be able to perform search based on the fields additionally added to the publication.

[Contact Ziga Marolt](#)

Servers

https://localhost:3001/api - dev

Authorize

health

Endpoints for validating service's health status

articles

Endpoints for listing and managing articles

authors

Endpoints for listing and managing authors

methods

Endpoints for listing and managing methods

foodstuffs

Endpoints for listing and managing foodstuffs

sources

Endpoints for listing and managing sources

fields

Dynamic Fields

GET

/fields/types

POST

/fields/add

GET

/fields/list

GET

/fields/{fieldID}

DELETE

/fields/{fieldID}/delete

PUT

/fields/{fieldID}/update

GET

/fields/{fieldID}/values/list

DELETE

/fields/{fieldID}/values/{valueId}/delete

POST

/fields/{fieldID}/values/add

Schemas

Healthz

UploadedFile

Slika 3.4: Dokumentacija z orodjem Swagger.

Da je delo enostavno, smo naredili skripto, ki poskrbi da se generira željena koda odjemalca in strežnika (Koda 3.4).

```
1 #!/bin/bash
2 set -e
3
4 readonly service="$1"
5 readonly docker_image="openapitools/openapi-generator-cli:v5
   .3.0"
6
7 docker run --rm --env "JAVA_OPTS=-Dlog.level=error"
8   -v "${PWD}:/local" \
9   "$docker_image" generate \
10  -i "/local/api/openapi/$service.yml" \
11  -g javascript \
12  -o "/local/web/src/services/clients/$service"
```

Koda 3.4: Skripta, ki poskrbi za generiranje kode za posamezno storitev.

Za zaledni del uporabimo generator **Swagger**, s katerim pridobimo osnovno kodo za strežnik HTTP, ki posreduje podatke odjemalcem. Omogoča nam osnovno validacijo podatkov, ki so poslani v zahtevku in le-te pretvori v primerno strukturo, s katero je enostavno manipulirati dalje v aplikaciji.

Za čelni del generiramo odjemalce JavaScript z **OpenApi** generatorjem, ki poskrbijo za komunikacijo s strežnikom na zalednem delu. S tem nam ni potrebno skrbeti, da bi skonstruirali zahtevek v napačni obliki.

Za izmenjavo podatkov med odjemalcem in strežnikom pa uporabljamo strukturo JSON (ang. JavaScript Object Notation), ki je preprosta oblika za izmenjavo podatkov in je neodvisna od programskega jezika (Koda 3.5). Zaradi besedne zasnove je enostaven za branje in pisanje tako ljudem kot tudi računalnikom [19].

```
1 curl --request POST \  
2 --url https://localhost:3001/api/authors/{authorID}/update \  
3 --header 'Authorization: Bearer ==token==' \  
4 --header 'Content-Type: application/json' \  
5 --data '{  
6 "name": "John Doe"  
7 }'
```

Koda 3.5: Primer izvedbe API klica.

Komunikacija je lahko uspešna ali ne, zato je potrebno odjemalcu odgovoriti z ustrežno kodo, kot je definirano v protokolu HTTP. Statuse lahko razdelimo v različne skupine, ki predstavljajo napake, storjene na strani odjemalca (4xx), napake, ugotovljene na strani strežnika (5xx) in uspešno obdelane zahteve (2xx). V tabeli 3.1 so prikazane vse napake, ki jih aplikacija lahko vrne.

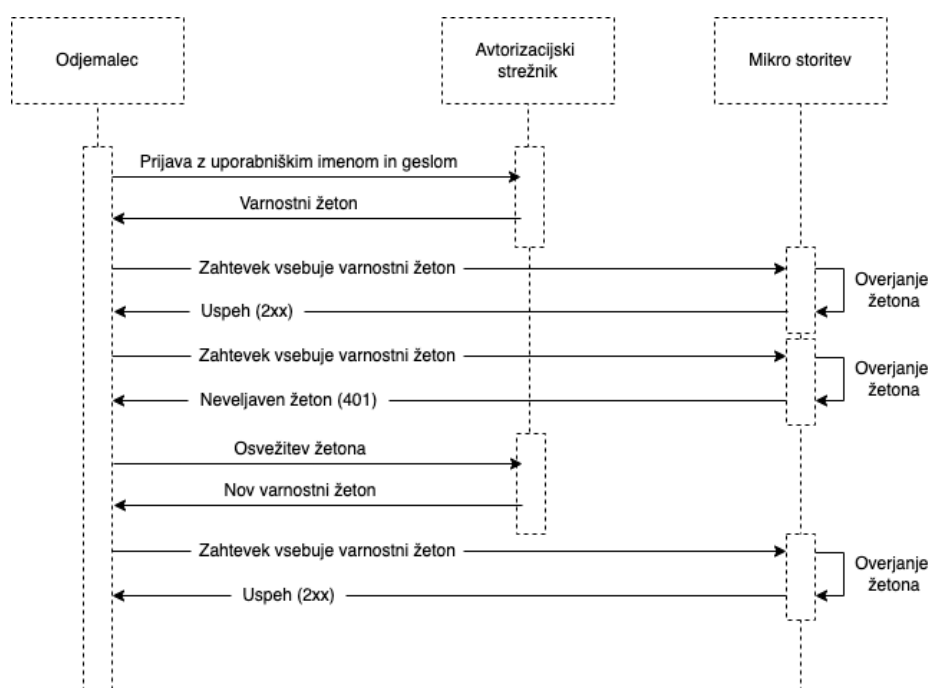
Koda napake	Pomen
200	<i>OK</i> - Zahtevek je uspešno izveden
400	<i>BadRequest</i> - Zahtevek ni veljaven
401	<i>Unauthorized</i> - Žeton ni veljaven
403	<i>Forbidden</i> - Ni zadostnih pravic
404	<i>NotFound</i> - Vir ne obstaja
422	<i>UnprocessableEntity</i> - Zahtevek vsebuje neveljavne podatke
429	<i>TooManyRequests</i> - Narejenih preveč zahtevkov v časovnem obdobju
500, 502, 503, 504	<i>InternalServerError</i> - Napaka na zalednem sistemu

Tabela 3.1: Seznam napak, ki jih vrača strežnik.

3.4 Avtentikacija in avtorizacija

Za preverjanje pristnosti uporabnika med posameznimi zahtevki uporabljamo žeton JWT (JSON Web Token), v katerem je zapisan identifikator uporabnika in njegova vloga. Žeton je brezstanjski, kar pomeni, da ni nikjer shranjen. To nam omogoča izdelavo ločenih sistemov, ki niso vezani na določeno shemo preverjanja pristnosti. Žeton je lahko ustvarjen kjer koli in porabljen v katerem koli sistemu, ki za podpis žetona uporablja isti skrivni ključ (ang. secret key). Tako nam podatkov o uporabniku ni potrebno vedno znova, ob vsakem zahtevku preveriti s podatki v bazi.

Odjemalec v glavi zahtevka pošlje **Bearer** žeton s katerim identificira uporabnika. Ker ima žeton določeno življenjsko dobo, je bilo potrebno poskrbeti tudi za osveževanje tega žetona (Slika 3.5). Ob vsaki spremembi žetona odjemalec nastavi novo vrednost v glavi zahtevka. S tem poskrbimo, da je žeton ves čas delujoč in veljaven (Koda 3.6).



Slika 3.5: Diagram prikazuje komunikacijo z uporabo žetona (JWT).

```
1 export function setApiClientAuth(idToken) {
2     users.authentications.bearerAuth.accessToken = idToken;
3     fields.authentications.bearerAuth.accessToken = idToken;
4     ...
5 }
```

Koda 3.6: Izsek kode za nastavljanje žetona posameznemu odjemalcu.

V zalednem sistemu se ob vsakem zahtevku sproži akcija za preverjanje dostopa uporabnika. Na podlagi skrivnega ključa, strežnik preveri, če je poslan žeton veljaven (Koda 3.7). Uporabljen skrivni ključ mora biti enak kot ključ s katerim je bil žeton ustvarjen. S tem zagotovimo, da se podatki vmes niso spremenili.

```
1 func UserMiddleware(ctx context.Context, authService Service)
   MiddlewareFunc {
2     return func(authToken string) (*User, error) {
3         if !strings.HasPrefix(authToken, "Bearer ") {
4             return nil, apierrors.Forbidden()
5         }
6
7         if authToken == "" {
8             return nil, apierrors.Unauthorized()
9         }
10        token, err := authService.VerifyToken(ctx, authToken)
11        if err != nil {
12            return nil, apierrors.Unauthorized()
13        }
14
15        return &User{
16            UID:          UID(token.UID),
17            Email:         token.Claims["email"].(string),
18            Role:          token.Claims["role"].(string),
19            DisplayName: token.Claims["name"].(string),
20        }, nil
21    }
22 }
```

Koda 3.7: Izsek kode za preverjanje pristnosti uporabnika.

V primeru, da je žeton veljaven, preberemo vrednost žetona, ki vsebuje identifikator uporabnika, uporabniško ime, elektronski naslov in vlogo (ang. role) uporabnika. S prebranimi vrednostmi kreiramo objekt uporabnika in nadaljujemo s procesiranjem zahtevka. V nasprotnem primeru, ko žeton ni veljaven pa odgovorimo z ustreznim odgovorom 401 - Unauthorized.

Preverjanje pravic se preveri v kodi, ki vsebuje logiko, kaj naj se zgodi z zahtevkom (Koda 3.8). Najprej preverimo, če je zahtevke od uporabnika, ki ima zadostne pravice za izvajanje tega zahtevka. V primeru, da uporabnik nima zadostnih pravic, mu odgovorimo z odgovorom 403 - Forbidden. Če uporabnik pravice ima, se procesiranje zahtevka nadaljuje.

```
1 // DeleteUserHandler handles the delete user request
2 func (h *HttpHandler) DeleteUserHandler() users.
   DeleteUserHandlerFunc {
3   return func(params users.DeleteUserParams, user *auth.User)
     middleware.Responder {
4     // check if user has rights to perform this action
5     if user.Role != string(domain.AdminRole) {
6       return users.NewDeleteUserHandlerForbidden()
7     }
8     uuid, err := utils.StringToUUID(string(params.UserID))
9     if err != nil {
10      return h.returnError(err)
11    }
12    u, err := h.app.Commands.DeleteUser.Handle(
13      params.HTTPRequest.Context(),
14      command.DeleteParamsCmd{UserUUID: uuid},
15    )
16
17    if err != nil {
18      return h.returnError(err)
19    }
20    return users.NewUpdateUserNoContent()
21  }
22 }
```

Koda 3.8: Izsek kode za preverjanje pravic uporabnika.

3.5 Podatkovna baza

Upravljanje s podatkovno shemo

Postavitev podatkovne sheme se izvede v zbirki podatkov vsakič, ko je potrebno posodobiti ali povrniti shemo baze podatkov na novo ali na starejšo različico. Ta proces imenujemo tudi migracija podatkovne sheme.

V aplikaciji uporabljamo programsko orodje `go-migrate`. Omogoča izvajanje migracij za različne vrste podatkovnih baz, med njimi tudi PostgreSQL, katero uporabljamo mi. Z orodjem ustvarimo migracijske datoteke, v katere napišemo shemo podatkovne baze. Vsaka migracijska akcija vsebuje dve datoteki. V eni datoteki so definirani ukazi (SQL stavki) za postavitev podatkovne sheme (Koda 3.9), in v drugi ukazi za povrnitev podatkovne sheme v stanje pred migracijo.

```
1  -- Table Definition
2  CREATE TABLE "articles"
3  (
4      "uuid"            uuid    NOT NULL PRIMARY KEY ,
5      "source_uuid"     uuid    NOT NULL REFERENCES sources(uuid)
6                          ON DELETE CASCADE ,
7      "title"           text    NOT NULL ,
8      "url"             text    NOT NULL DEFAULT '' ,
9      "year"            int ,
10     "comment"          text    NOT NULL DEFAULT '' ,
11     "temperature_min" int     NOT NULL DEFAULT 0 ,
12     "temperature_max" int     NOT NULL DEFAULT 0 ,
13     "file_uuid"        uuid ,
14     "approved_by"      uuid ,
15     "released_at"      timestamp          DEFAULT null ,
16     "created_at"       timestamp NOT NULL DEFAULT now() ,
17     "updated_at"       timestamp NOT NULL DEFAULT now()
18 );
```

Koda 3.9: Izsek kode za kreiranje tabele *articles*, katere namenjen je hranjenje podatkov o publikacijah.

Struktura podatkov

Z migracijami smo definirali celotno strukturo podatkovne baze, ki jo potrebujemo. Definirali smo 13 tabel, ki skrbijo, da so podatki pravilno shranjeni. Vizualni prikaz tabel in medsebojnih relacij je viden na sliki 3.6. Opis posameznih tabel:

gomigrate: Tabela vsebuje informacije o stanju migracij. Ob vsakem izvajanju migracij se v tabelo zapiše identifikator migracije, glede na katerega se izvajajo nadaljnje migracije za podatkovno shemo.

users: Tabela vsebuje informacije o uporabnikih, ki so registrirani v sistem. Uporabniki se v aplikaciji razlikujejo glede na enoličen identifikator UUID in unikatni e-poštni naslov ("email").

articles: Tabela vsebuje informacije o publikacijah, vnesenih s strani uporabnika. Vsebuje informacije, kot so naslov, leto izdaje, komentar, najvišja in najnižja temperatura. Poleg informacij, ki jih vnese uporabnik, imamo tudi evidenco o tem, kdo je potrdil publikacijo. Tabela vsebuje tudi relacije z ostalimi entitetami, za povezavo z metodami, avtorji, viri, dinamičnimi podatki in živili. Ena publikacija lahko ima le en vir, zato je ta relacija definirana s tujim ključem v isti tabeli, prav tako velja za pripeto datoteko.

Ostale relacije, kot so živila, pa so definirana kot mnogo-proti-mnogo, kar pomeni, da potrebujemo medsebojne tabele za povezovanje entitet.

- `article.methods`: tabela za povezovanje publikacij in metod
- `article.foodstuffs`: tabela za povezovanje živil in publikacij
- `article.authors`: tabela za povezavo publikacij in avtorjev
- `article.fields`: tabela za povezavo dinamičnih podatkov s publikacijami

authors: Tabela vsebuje informacije o avtorjih, katerih publikacije so vnešene v našo aplikacijo. Za povezavo se uporablja medsebojna tabela

article_authors.

methods: Tabela vsebuje informacije o metodah, ki jih posamezne publikacije uporabljajo. Za medsebojno povezavo s publikacijami se uporablja tabela *article_methods*.

foodstuffs: Tabela vsebuje informacije o živilih, ki so bila uporabljena pri posameznih publikacijah. Za medsebojno povezavo s publikacijami se uporablja tabela *article_foodstuffs*.

foodstuffs: Tabela vsebuje informacije o živilih, ki so registrirani v sistemu. Za medsebojno povezavo s publikacijami se uporablja tabela *article_foodstuffs*.

sources: Tabela vsebuje informacije o virih, iz katerih črpamo publikacije. Tip vira je definiran kot *enum*, čigar vrednost je definirana kot knjiga, internetni vir ali revija. Za definicijo tipa smo uporabili *PostgreSQL* ukaz:

```
1 CREATE TYPE source_type AS ENUM ('web', 'book', 'magazine')
```

fields: Tabela vsebuje informacije o vseh dinamično definiranih podatkih v aplikaciji. Podatki se razlikujejo glede na tip. Vsebujejo lahko enega ali več vrednosti. V primeru, da je podatek sestavljen iz večih vrednosti, uporabimo relacijsko tabelo *field_values*, kjer so zapisane vrednosti za posamezen podatek.

article_fields: Tabela vsebuje relacijo med dinamičnimi podatki in njihovimi vrednostmi. Za potrebe optimalnega iskanja hrani različne tipe vrednosti:

- *value_vchar*: hrani tekstovno vrednost
- *value_date*: hrani časovno vrednost
- *value_number*: hrani numerično vrednost

Shranjevanje podatkov

Potrebno je poskrbeti, da so podatki shranjeni v popolni obliki. V primeru, da pri shranjevanju pride do napake, jo je potrebno ustrezno rešiti. Da so podatki shranjeni v popolni obliki, uporabljamo transakcije. Transakcija je atomična enota, s čimer zagotovimo, da so vse poizvedbe izvedene uspešno. V primeru, da ena poizvedba ni bila uspešno izvedena, se vse poizvedbe, storjene v isti transakciji, povrnejo v stanje pred izvedbo.

Posamezna metoda v repozitoriju poleg ostalih parametrov prejme še funkcijo, katera se izvede znotraj transakcije (Koda 3.10). V primeru, da funkcija vrne napako, se transakcija resetira in povrne podatkovno bazo v stanje pred shranjevanjem.

```
1 func (h AddArticleHandler) Handle(ctx context.Context, cmd
    AddArticleParamsCmd) (res *domain.Article, err error) {
2     res, err = h.repo.Create(ctx, article, func(ctx context.
        Context, a *domain.Article) error {
3         return h.repo.AssignAuthors(ctx, a, cmd.Authors, func(ctx
            context.Context, a *domain.Article) error {
4             return h.repo.AssignFoodstuffs(ctx, a, cmd.Foodstuffs,
                func(ctx context.Context, a *domain.Article) error {
5                 return h.repo.AssignMethods(ctx, a, cmd.Methods, func
                    (ctx context.Context, a *domain.Article) error {
6                     return h.repo.AddExtraFields(ctx, a, fields)
7                 })
8             })
9         })
10    })
11
12    if err != nil {
13        return res, errors.WrapErrorf(err, errors.
            ErrorCodeUnknown, "AddArticleHandler.repo")
14    }
15
16    return res, nil
```

```
17 }
```

Koda 3.10: Prikaz shranjevanja publikacije.

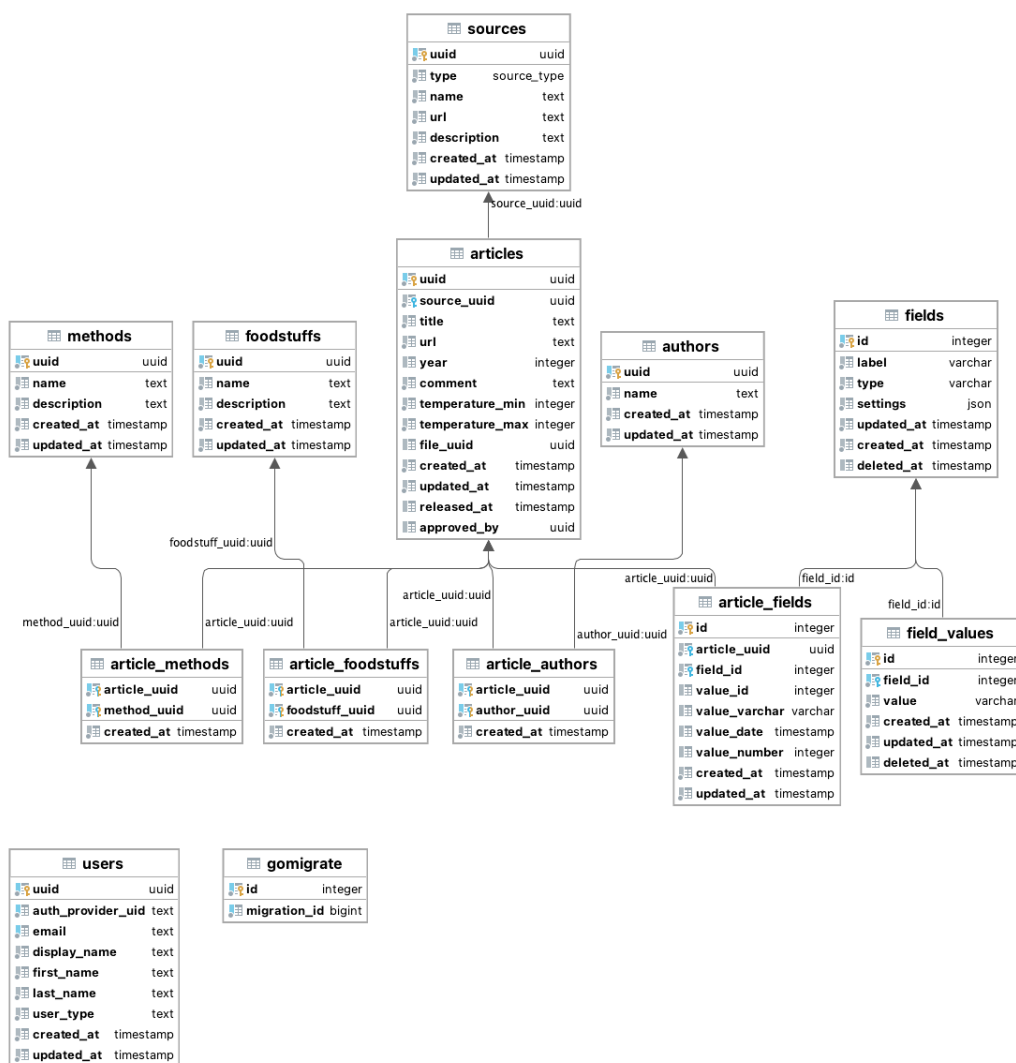
Datotek, pripetih k publikacijam, ne shranjujemo v relacijsko podatkovno bazo. Definirali smo vmesnik, ki nam pove, katere metode moramo implementirati za pravilno delovanje (Koda 3.11). Implementirali smo repozitorij, ki uporablja S3 kompatibilnega odjemalca. Za hrambo teh datotek uporabljamo storitev Minio (2.4.6).

```
1 // FileRepository provides functions to upload and read file
2 FileRepository interface {
3     UploadFile(ctx context.Context, rc io.ReadCloser, size
4         int64) (*utils.BINARY16, error)
5     GetFile(ctx context.Context, uuid utils.BINARY16) (io.
6         ReadCloser, error)
7 }
```

Koda 3.11: Prikaz vmesnika za shranjevanje in branje datotek.

Branje podatkov

V aplikaciji smo poskrbeli, da do težave z izvajanjem poizvedb po nepotrebem ne bo prišlo. Pri poizvedovanju podatkov o publikacijah iz podatkovne baze je potrebno biti pozoren na število izvedenih klicev. Če nismo pazljivi, lahko pridemo do težave s poizvedbo $n+1$. Ta se zgodi, ko del aplikacije izvede n dodatnih poizvedb, da bi pridobil iste podatke, ki bi jih bilo mogoče pridobiti pri izvajanju primarne poizvedbe SQL. Večja kot je vrednost n , več poizvedb bo izvedenih, večji je vpliv na zmogljivost.



Slika 3.6: Predstavitev podatkovne sheme z ER diagramom.

3.6 Čelni del aplikacije

Čelni del aplikacije uporablja ogrodje `Vue.js` (2.4.7). Je zelo enostavno `JavaScript` ogrodje s številnimi orodji in knjižnicami, s katerimi si poenostavimo in pohitrimo razvoj. Knjižnice, ki jih aplikacija potrebuje, so navedene v datoteki `package.js` (Koda 3.12). V omenjeni datoteki so navedeni tudi ukazi, s katerimi prožimo določene akcije, kot so gradnja statičnih paketov, postavitve spletnega strežnika za streženje datotek in gradnja CSS datotek s pomočjo knjižnice `tailwind`. Za nadzor nad uporabljenimi knjižnicami skrbi orodje `NPM` (ang. Node Package Manager).

```
1 {
2   "name": "diploma",
3   "version": "1.0.0",
4   "private": true,
5   "scripts": {
6     "serve": "vue-cli-service serve",
7     "build:tailwind": "npx tailwindcss ...",
8   },
9   "dependencies": {
10    "@fortawesome/fontawesome-free": "^5.15.4",
11    "@tailwindcss/forms": "^0.3.4",
12    "@vueform/multiselect": "^2.2.1",
13    "@vueform/slider": "^2.0.8",
14    "core-js": "^3.19.1",
15    "firebase": "^9.3.0",
16    "jsonwebtoken": "^8.5.1",
17    "litepie-datepicker": "^1.0.14",
18    "lodash": "^4.17.21",
19    "moment-timezone": "^0.5.34",
20    "superagent": "^6.1.0",
21    "v-tooltip": "^4.0.0-beta.2",
22    "vue": "^3.2.21",
23    "vue-router": "^4.0.0-0",
24    "vue-toast-notification": "^2.0.1"
25  },
26  "devDependencies": {
```

```
27     ...
28     "sass": "^1.32.5",
29     "sass-loader": "^10.1.1",
30     "tailwindcss": "^2.2.19"
31   }
32 }
```

Koda 3.12: Izsek naštetih knjižnic in ukazov v datoteki `package.json`.

Ogrodje je odvisno od `node.js` programskih datotek, zato je moramo v sliko vsebnika namestiti potrebne stvari za delovanje tega ogrodja. Uporabimo obstoječo sliko vsebnika `node:17.3.0-alpine3.14`, ki že ima nameščena orodja kot so `node.js` in upravljalac paketkov `npm`. Ker je vsebnik uporabljen v razvojnem okolju, definiramo vrednost `NODE_ENV` na vrednost `development`. Skripta `run.sh` je skopirana v sliko z dodatnimi pravicami za zagon. Na koncu se skripto sproži v izvajanje (Koda 3.13).

```
1 FROM node:17.3.0-alpine3.14
2
3 ENV NODE_ENV development
4
5 ADD start.sh /
6 RUN chmod +x /start.sh
7
8 CMD ["/run.sh"]
```

Koda 3.13: Dockerfile datoteka za razvijanje Vue aplikacije.

V datoteki `start.sh` (Koda 3.14) so navedeni ukazi, ki se zgodijo ob vsakem zagonu vsebnika. Ukaz `npm install` namesti vse potrebne knjižnice, ki jih v aplikaciji uporabljamo. Ukaz `npm serve` pa zažene spletni strežnik, ki streže vsebino aplikacije iz trenutnega direktorija.

```
1 set -e
2
3 npm install
4 npm serve
```

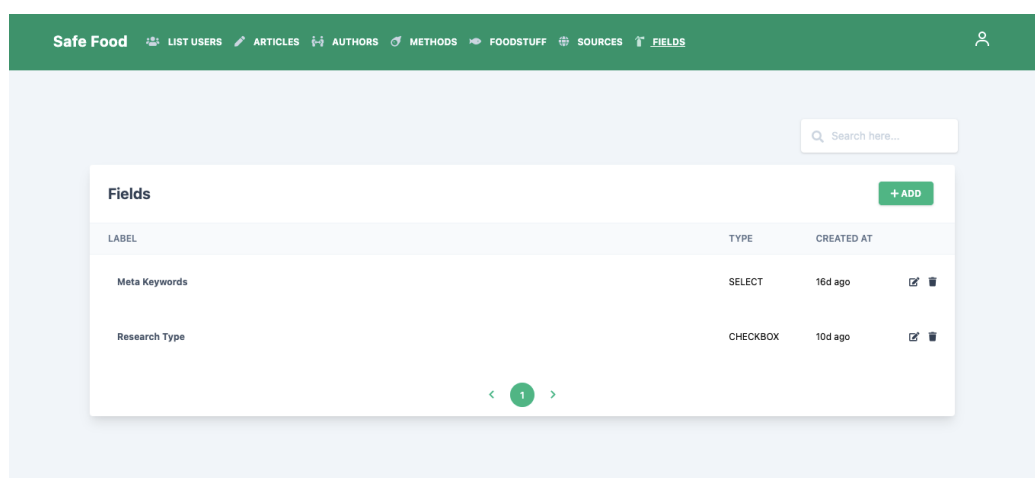
Koda 3.14: Ukazna datoteka, ki naloži potrebne knjižnice in streže aplikacijo.

Pomembno je tudi poudariti, da je aplikacija enostranska (ang. single-page), kar pomeni, da deluje znotraj brskalnika in ne potrebuje ponovnega nalaganja strani med svojim delovanjem. Je le ena sama stran, ki jo obiščemo in na kateri se nato naloži vso ostalo vsebino s pomočjo kode *JavaScript*.

Kot je razvidno iz prikaza zagnanih vsebnikov (Koda: 3.2), ima čelni del odprta vrata 80, kar pomeni, da je stran dostopna na lokalnem spletnem naslovu `http://localhost:80/`.

3.7 Razvoj vnosa dinamičnih podatkov

Za potrebe aplikacije smo razvili popolnoma dinamičen način za dodajanje novih parametrov publikacije (Slika 3.7). Te paremetre lahko dodajamo, urejamo in brišemo tudi med tem, ko urejamo publikacijo, in kasneje izvajamo razne filtre nad vnesenimi podatki.



Slika 3.7: Stran za prikaz in urejanje dinamičnih parametrov.

Zaradi potrebe po razlikovanju po različnih tipih podatkov smo definirali več podatkovnih tipov, ki predstavljajo posamezne podatke. Zaradi raznolikosti med podatki vseh podatkov ne moremo predstaviti enako. Podatek, predstavljen kot številka, se ne odraža enako kot podatek, ki je predstavljen

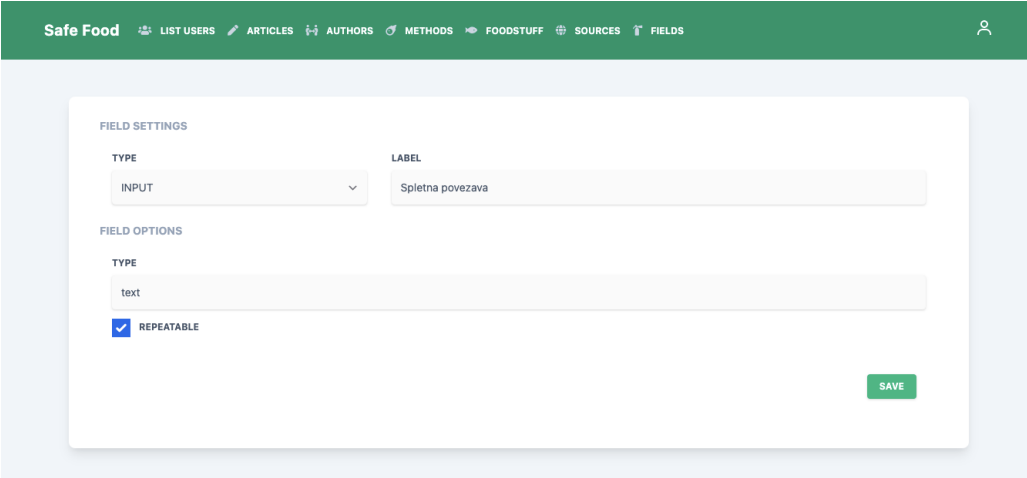
kot tekstovno polje. Katere tipe podatkov aplikacija podpira, je definirano v zalednem sistemu in je dostopno preko aplikacijskega programskega vmesnika (Koda 3.15).

```
1 curl --request GET \  
2   --url https://localhost:3001/api/fields/types \  
3   --header 'Authorization: Bearer ==token==' \  
4   --header 'Content-Type: application/json' \  
5   { \  
6     "types": [ \  
7       { \  
8         "name": "select", \  
9         "has_values": true, \  
10        "settings": [ \  
11          { \  
12            "name": "string", \  
13            "type": "number", \  
14            "default_value": "string" \  
15          } \  
16        ] \  
17      } \  
18    ] \  
19  }
```

Koda 3.15: Izsek aplikacijskega vmesnika za pridobitev vseh podprtih podatkovnih tipov v aplikaciji.

Vnosno polje

Vnosno polje predstavlja podatek, ki ga je potrebno vnesti vsakič znova in nima definirane nobene vnaprej določene vrednosti. Vnosno polje je lahko predstavljeno kot številčna vrednost, ali pa kot tekstovno polje. Za ta podatek imamo možnost vnosa mnogih vrednosti, tako da definiramo podatek kot ponavljajoč podatek - **repeatable**. To je uporabno za primere, ko hočemo dodati zraven publikacije enega ali več spletnih virov.



The screenshot shows a web application interface with a green header bar containing the text "Safe Food" and several navigation icons. Below the header, a white dialog box titled "FIELD SETTINGS" is displayed. Inside the dialog, there are two main sections: "FIELD SETTINGS" and "FIELD OPTIONS". In the "FIELD SETTINGS" section, the "TYPE" dropdown menu is set to "INPUT", and the "LABEL" text field contains the text "Spletna povezava". In the "FIELD OPTIONS" section, the "TYPE" dropdown menu is set to "text", and the "REPEATABLE" checkbox is checked. A green "SAVE" button is located at the bottom right of the dialog box.

Slika 3.8: Definiranje vnosnega polja tipa *input*.

Na sliki 3.8 je razvidno, kako se definira podatek imenovan „Spletna povezava“. Vrednost spletne povezave je tekstovnega tipa, zato smo za vrednost izbrali `text`. Izbrali smo tudi vrednost `repeatable`, ker želimo omogočiti več vnosov spletnih povezav. Dodan parameter se lahko pojavi in je na voljo med urejanjem publikacije.

Na dnu se nam prikaže vnosno polje, za katerega lahko izberemo in definiramo njegovo vrednost (Slika 4.13). Ker je parameter definiran tako, da je lahko predstavljen z več kot eno vrednostjo, se nam ponudi gumb za dodajanje nove vrednosti.

Izbirno polje - Select

Ta podatek predstavlja podobne vrednosti, kot vnosno polje „Input“, z razliko, da so vrednosti lahko že vnaprej definirane. Te vrednosti nam omogočajo hitrejše vnašanje podatkov. Na voljo je tudi iskanje po vnesenih vrednostih in sprotno dodajanje vrednosti v primeru, da vrednost še ne obstaja. Vse to je možno nastaviti med samim definiranjem parametra (Slika 3.9). Na voljo je tudi možnost izbire več kot ene vrednosti, z izbiro opcije `multiple`.

Opis posameznih opcij, med samim definiranjem parametra :

- **taggable** - omogoča dodajanje novih vrednosti med samim urejanjem
- **multiple** - omogoča izbiro več kot ene vrednosti
- **searchable** - omogoča iskanje po vnesenih vrednostih

The screenshot shows the 'Safe Food' application's 'FIELD SETTINGS' dialog. The 'TYPE' is set to 'SELECT'. The 'LABEL' is 'Meta Keywords'. Under 'FIELD OPTIONS', the checkboxes for 'TAGGABLE', 'MULTIPLE', and 'SEARCHABLE' are all unchecked. A 'SAVE' button is at the bottom. The 'FIELD VALUES' section on the right lists predefined values: 'cold chain logistics', 'shelf life', 'prediction', 'Total-Volatile Basic Nitrogen (TVB-N)', 'Psychrotrophic Plate Count (PPC)', and 'sensory score'. Each value has a trash icon to its right for deletion. A 'Type in ...' input field with a checkmark is at the top of this section.

Slika 3.9: Definiranje vnosnega polja tipa *select*.

Izbirno polje - checkbox

Ta predstavlja podatek, katere vrednosti so statične in vnaprej definirane. Vnos podatkov je podoben kot pri „izbirnem polju - Select“ (Slika 3.10), z razliko, da je podatek prikazan na drugačen način. Ta tip podatka nam ne omogoča iskanja po vrednostih in sprotnega dodajanja med samim urejanjem publikacije.

Polje za izbiranje časovnega podatka (datum)

Predstavlja podatek, ki ima časovno vrednost. Uporabniku je omogočeno definiranje podatka z izbiro datuma in časa na koledarju (Slika 3.11).

The screenshot shows the 'Safe Food' application interface. At the top is a green navigation bar with the title 'Safe Food' and several menu items: LIST USERS, ARTICLES, AUTHORS, METHODS, FOODSTUFF, SOURCES, and FIELDS. A user profile icon is on the right. Below the navigation bar is a light blue background with a white card containing two sections: 'FIELD SETTINGS' and 'FIELD VALUES'. In the 'FIELD SETTINGS' section, the 'TYPE' is set to 'CHECKBOX' and the 'LABEL' is 'Select Continent'. A green 'SAVE' button is at the bottom right of this section. The 'FIELD VALUES' section on the right lists five values: 'Type in ...' (with a checkmark), 'Australia', 'Europe', 'America', and 'Asia'. Each value has a trash icon to its right.

Slika 3.10: Definiranje vnosnega polja tipa *checkbox*.

Uporabili smo minimalistično knjižnico `Day.js`, ki z enostavnim aplikacijskim vmesnikom razčlenjuje, preverja, manipulira in prikazuje datume in ure za sodobne brskalnike. Ponuja tudi podporo z ostalimi knjižnicami, kot je na primer bolj priznana knjižnica `Moment.js`.

The screenshot shows a 'RELEASE DATE' input field. The text inside the field is '2022-01-13 10:02:32'. To the right of the field is a green 'SAVE' button. Below the input field, a date picker calendar is open, showing the month of January for the year 2022. The calendar has a header with navigation arrows, the month 'JAN', and the year '2022'. The days of the week are listed as Sun, Mon, Tue, Wed, Thu, Fri, Sat. The dates are arranged in a grid, with the 13th of January highlighted in a grey circle.

Slika 3.11: Prikaz grafičnega vmesnika za izbiranje datuma.

Nalaganje datoteke

Ta komponenta nam omogoča nalaganje datoteke za posamezno publikacijo. Na voljo je dodajanje slik in datotek tipa `.pdf`. Ostalih vrst datotek aplikacija ne sprejme.

Poglavje 4

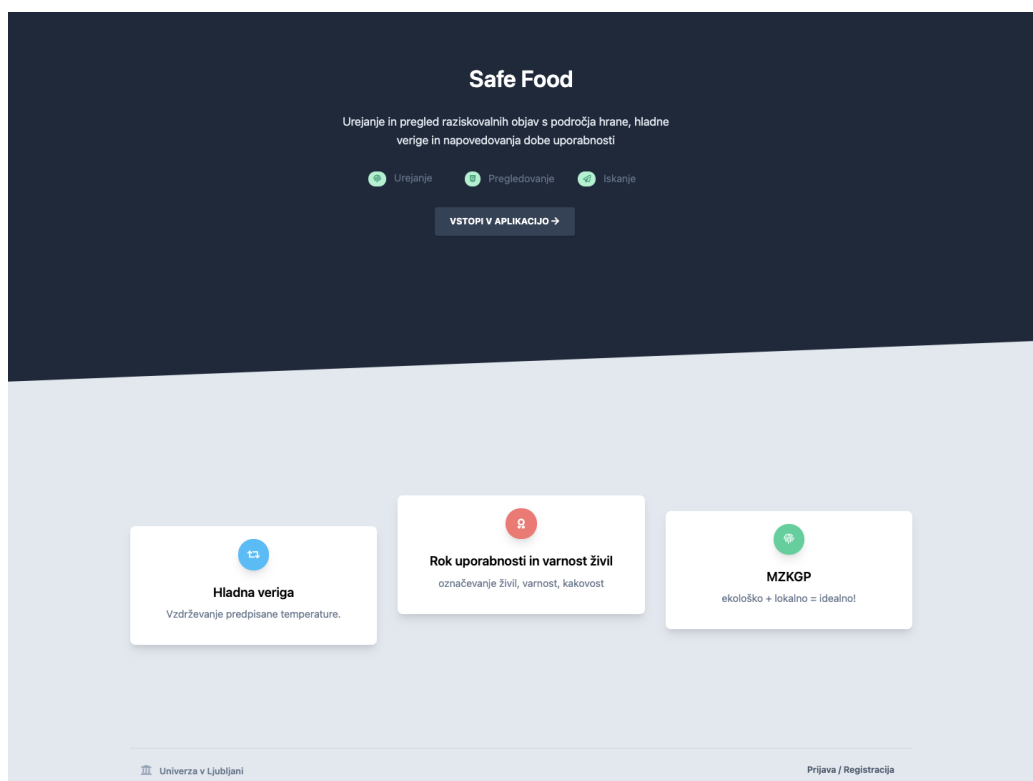
Predstavitev aplikacije

Aplikacija, ki smo jo poimenovali „Safe Food“ je sestavljena iz zalednega in čelnega dela. Zaledni del vključuje več komponent, ki sledijo arhitekturni postavitvi mikrorstitev. Čelni del pa je enostavna aplikacija, napisana v programskem ogrodju `Vue.js`. Namenjena je uporabniku, zato je pomembno, da je na videz enostavna in pregledna, hkrati pa tudi hitra in funkcionalna.

4.1 Uvodna stran

Ob obisku spletnega mesta `http://localhost` se odpre „Uvodna stran“ (Slika 4.1). Stran omogoča lokalno uporabo aplikacije in povezave na zunanje vire, kjer lahko pridobimo koristne informacije o živilih. Spodaj je povezava do strani Univerze v Ljubljani in pa do prijave oziroma registracije v aplikacijo.

S klikom na gumb „Vstopi v aplikacijo“, nadaljujemo v aplikacijo. V primeru, da smo v aplikaciji že prijavljeni, se nam prikaže stran z urejenimi publikacijami, nad katerimi lahko uporabljamo številne filtre (Poglavje 4.6). V nasprotnem primeru, ko še nimamo tekoče seje, pa se nam prikaže stran za prijavo.



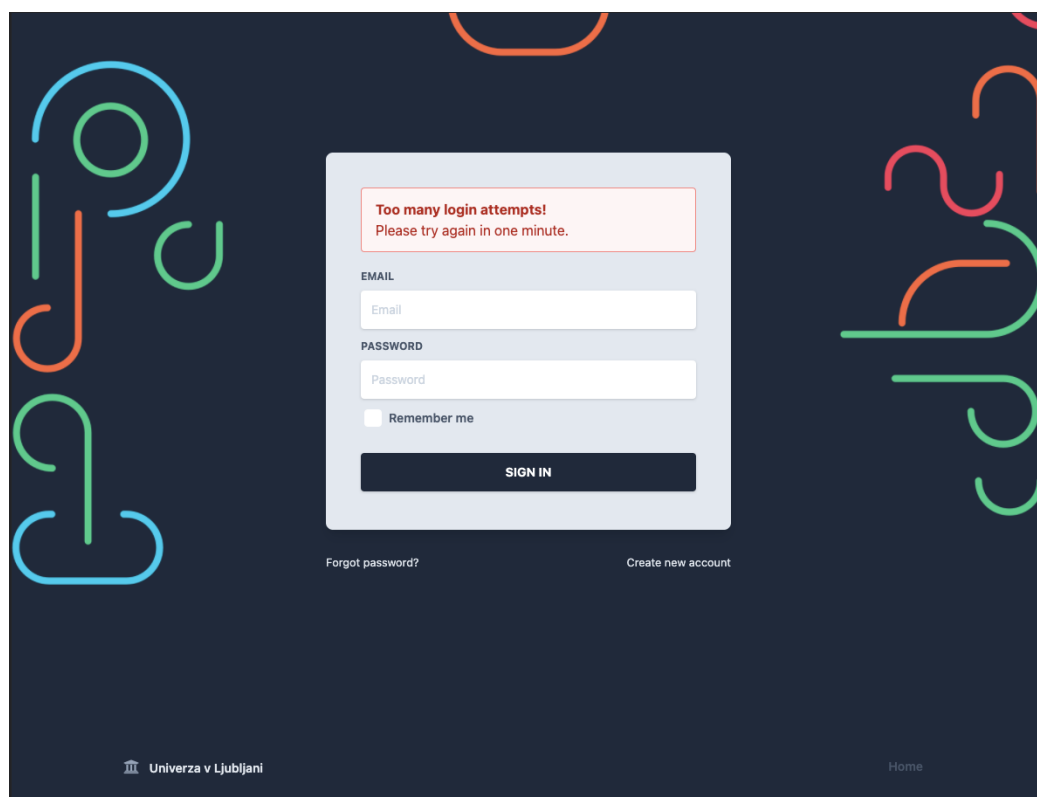
Slika 4.1: Uvodna stran aplikacije s povezavami na zunanje vire.

4.2 Prijava uporabnika

Obstoječi uporabnik se lahko prijavi z uporabniškim imenom in geslom (Slika 4.2). Če je uporabnik presegel določeno število neuspešnih poskusov prijave, mu je dostop do strani začasno omejen in prijava onemogočena. Na vrhu obrazca se izpiše ustrezna napaka, ki podaja razlog napake in navodila za njeno odpravljanje. Poleg tega lahko uporabnik spodaj klikne „Forgot password“, če je pozabil geslo, ali „Create new account“, če še ni prijavljen.

4.2.1 Pozabljeno geslo

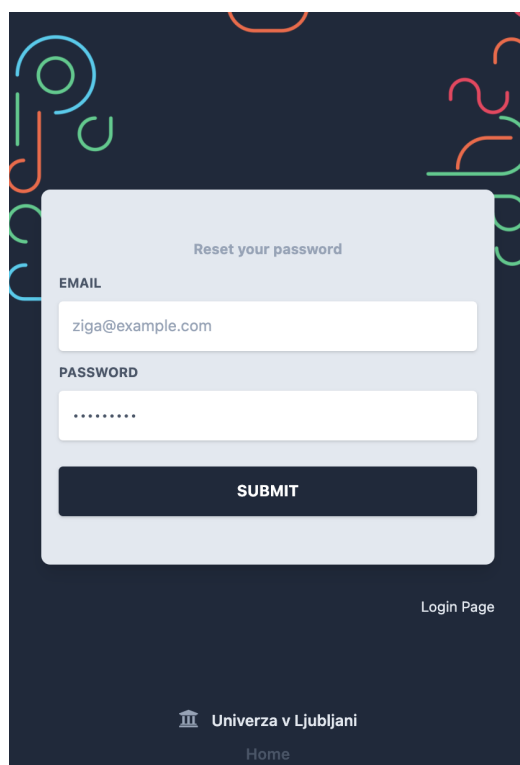
V primeru, da izgubimo ali pozabimo geslo za prijavo v aplikacijo, na prijavni strani izberemo „Forgot password“. S klikom na gumb se odpre spletni



Slika 4.2: Prijava uporabnika in izpis napake v primeru blokiranja uporabnika.

obrazec (Slika 4.3), kjer vnesemo e-poštni naslov, s katerim smo identificirani v aplikaciji. Na ta naslov prejmemo sporočilo, ki zajema unikatno ustvarjeno URL povezavo do aplikacije. S klikom na povezavo se odpre stran za vnos novega gesla. Ob uspešni nastavitvi gesla je uporabnik preusmerjen na prijavno stran, kjer se mora prijaviti z novim geslom.

V primeru, da aplikacijo uporabljamo na različnih napravah, nam bo odvzet dostop iz vseh ostalih naprav, kar pomeni, da bo na vseh potrebna ponovna prijava.



Reset your password

EMAIL

ziga@example.com

PASSWORD

.....

SUBMIT

Login Page

Univerza v Ljubljani

Home

Slika 4.3: Vnos novega gesla.

4.2.2 Stran za registracijo

Za uporabo aplikacije je potrebna registracija. Uporabniki, ki še niso registrirani v aplikacijo, imajo možnost, da si ustvarijo račun (Slika 4.4). Na strani se nahaja obrazec za vnos potrebnih informacij za ustvarjanje računa.

Ob kliku na gumb „Create account“ se sproži validacija, ki preveri, ali vneseni podatki ustrezajo zahtevam aplikacije. V primeru vnosa napačnega uporabniškega imena, elektronskega naslova ali gesla, se pod posameznim vnosnim poljem izpiše ustrezno opozorilo o napaki. Validacija se izvaja tako na čelnem kot tudi na zalednem delu aplikacije (Koda 4.1).

Ob registraciji prejme uporabnik novo sporočilo za potrditev uporabniškega računa. Potrditveno sporočilo je namenjeno zagotavljanju verodostoj-

Sign up with credentials

NAME

Ziga-Narobe

Invalid username. Please use only letters (a-z), numbers, dots or underscore.

EMAIL

ziga@example.com

PASSWORD

REPEAT PASSWORD

CREATE ACCOUNT

Already have an account?

Univerza v Ljubljani

Home

Slika 4.4: Registracija uporabnika.

nosti uporabnika. V primeru, da uporabnik ne potrdi svojega e-poštnega naslova v roku enega dneva, mu je dostop do aplikacije onemogočen, vse dokler ga uporabnik ne potrdi.

Nepotrjen uporabnik nima dostopa do dela aplikacije, kjer so prikazane publikacije. Pokaže se mu le stran z navodili, kako potrditi e-poštni naslov. V primeru, da uporabnik ni prejel potrditvenega sporočila, lahko klikne na gumb in sproži ponovno pošiljanje potrditvenega sporočila. Sporočilo bo ponovno ustvarjeno le v primeru, da je minila vsaj ena ura od zadnjega poizkusa pošiljanja tega sporočila.

```
1 function validateUsername(uname) {
2     /*
3         Uporabnisko ime, ne sme biti krajše od treh znakov
4     */
5     if(uname.length < 3) {
6         return false;
7     }
8
9     /*
10        Vsebuje lahko le:
11        - male crke (a-z)
12        - številke (0-9)
13        - pike (.)
14        - podcrtaje (_)
15    */
16    const match = /^[a-z0-9_\.]+$/.exec(uname);
17
18    // veljavno ali ne
19    return Boolean(rezultat);
20 }
```

Koda 4.1: Primer validacijske funkcije za preverjanje uporabniškega imena z uporabo regularnega izraza.

Vsak uporabnik je ob registraciji le navaden uporabnik. Vlogo uporabnika lahko spremeni le administrator, v administracijskem delu aplikacije. O tem bomo govorili v poglavju „Administracija uporabnikov“ (Poglavje 4.3).

V elektronskem sporočilu, ki ga uporabnik prejme na svoj naslov, je navedena povezava do aplikacije. Generirana povezava vsebuje parameter "method", ki pove, katera akcija se mora izvesti, ko uporabnik odpre stran. Komponenta prebere parameter *url* in glede na prebran parameter sproži ustrezno akcijo na storitev za njegovo preverjanje.

V primeru, da je uporabniški račun uspešno potrjen, je preusmerjen na začetno stran aplikacije, kjer lahko pregleduje objavljene publikacije s pripadajočimi podatki. Zgodi pa se lahko tudi, da elektronski naslov ni uspešno potrjen (Koda 4.2). To se lahko zgodi v naslednjih primerih:

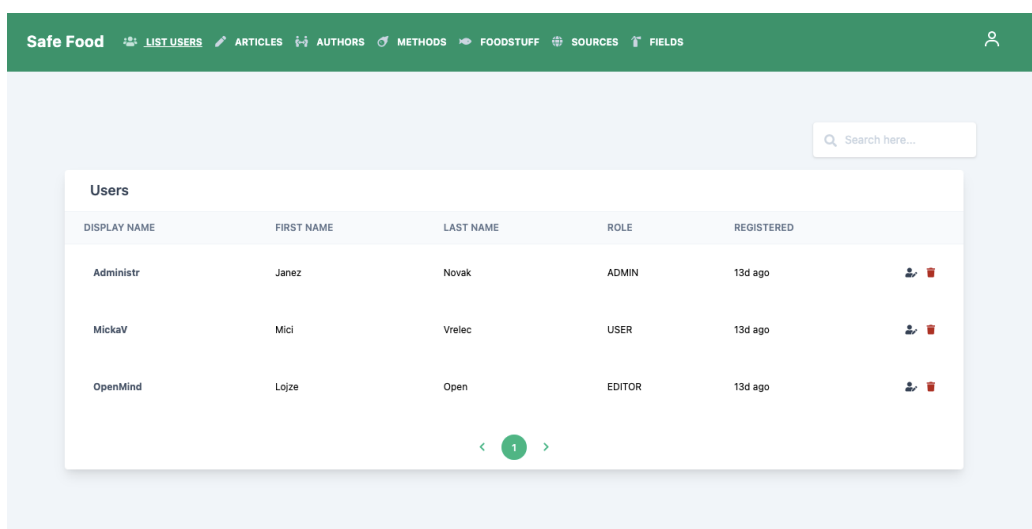
- potrditvena koda je potekla
- potrditvena koda ni veljavna
- uporabniški račun je izbrisan
- uporabniški račun je onemogočen.

```
1 mounted() {
2   this.mode = queryParams.mode;
3   this.oobCode = queryParams.oobCode;
4
5   if (this.mode === "verifyEmail") {
6     verifyAccount(this.oobCode).then(() => {
7       this.$toast.success("Email verified");
8       this.$router.push("home");
9     }, (error) => {
10      switch (error.code) {
11        case 'auth/expired-action-code':
12          this.$toast.error('Code expired!')
13          break;
14        case 'auth/invalid-action-code':
15          this.$toast.error('The code is not valid.')
16          break;
17        case 'auth/user-disabled':
18          this.$toast.error('Account is disabled.')
19          break;
20        case 'auth/user-not-found':
21          this.$toast.error('Your account has been removed.')
22          break;
23        default:
24          this.$toast.error('Email verification failed!')
25      }
26    });
27  }
28 }
```

Koda 4.2: Primer kode, za preverjanje uporabnika.

4.3 Urejanje uporabnikov

Administrator ima možnost pregledovanja in urejanja uporabnikov v aplikaciji. Na strani „Users List“ (Slika 4.5) je seznam registriranih uporabnikov. S klikom na ikono za smeti lahko administrator uporabniku onemogoči račun. Ker je v aplikaciji lahko registriranih veliko uporabnikov, je seznam uporabnikov prikazan na več straneh. Na dnu seznama se nahaja paginacija, s katero lahko prehajamo med stranmi.

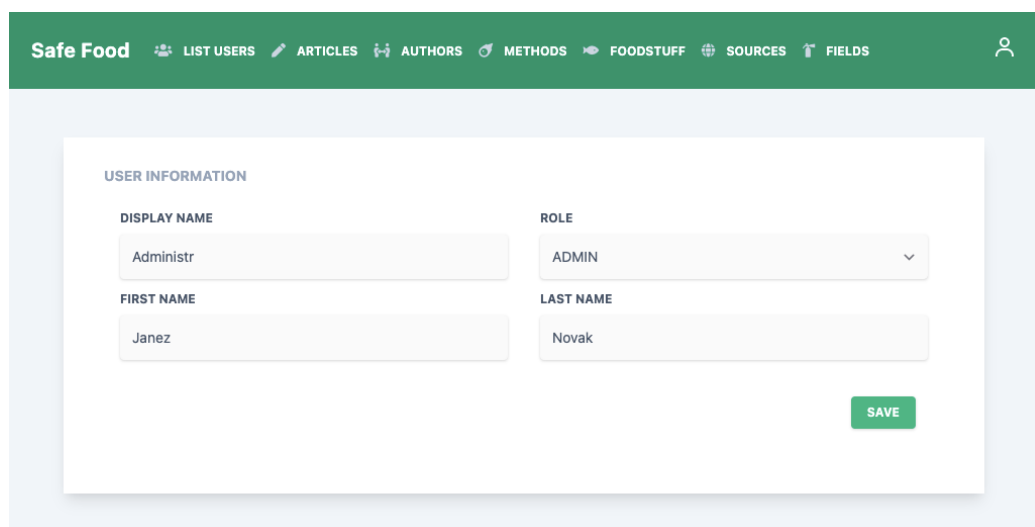


Users				
DISPLAY NAME	FIRST NAME	LAST NAME	ROLE	REGISTERED
Administr	Janez	Novak	ADMIN	13d ago
MickaV	Mici	Vrelec	USER	13d ago
OpenMind	Lojze	Open	EDITOR	13d ago

Slika 4.5: Seznam vseh registriranih uporabnikov.

S klikom na ikono za urejanje se odpre stran za urejanje izbranega uporabnika. Na voljo je spreminjanje uporabniškega imena, e-poštnega naslova in ostalih informacij uporabnika, kot sta ime in priimek (Slika 4.6).

Administratorju je tudi na voljo spreminjanje vrste uporabnika, torej lahko nekemu uporabniku doda pravice za urednika, ga dodeli kot administratorja, ali pa kot navadnega uporabnika, ki lahko pregleduje vnesene publikacije.



Slika 4.6: Stran za urejanje uporabnikov.

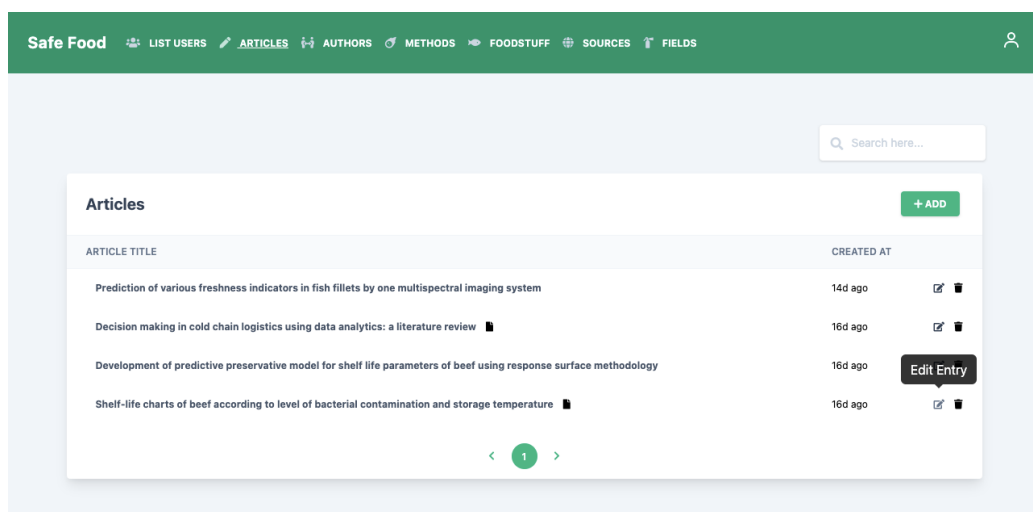
4.4 Vnos publikacije

Uredniki s pravicami urejanja ali dodajanja publikacij imajo dostop do strani „Articles“. Na njej je prikazan seznam vseh vnesenih publikacij. Ker je vsako publikacijo mogoče urediti ali izbrisati, se na desni strani nahaja polje z ikonama za urejanje in brisanje posamezne publikacije.

V primeru, da je vnesenih publikacij veliko, je na dnu seznama na voljo paginacija, s katero se pomikamo med stranmi. Na vrhu seznama imamo tudi možnost iskanja po naslovu publikacije. Seznam publikacij je prikazan z novjšimi publikacijami na vrhu (Slika 4.7). Na strani se nahaja tudi gumb za dodajanje nove publikacije (+Add).

S klikom na ikono za urejanje se odpre stran z izpolnjenim spletnim obrazcem s podatki o vneseni publikaciji (Slika 4.8). V ozadju se uporablja ista `vue.js` komponenta tudi za kreiranje nove publikacije. Glede na akcijo je potrebno poslati ustrezen zahtevek na zaledni sistem.

V primeru urejanja, se v naslovu URL nahaja identifikator publikacije, ki jo urejamo. Z njegovo uporabo pošljemo zahtevek na zaledni sistem, kjer dobimo vse potrebne informacije, ki so bile vnesene za dano publikacijo.



Slika 4.7: Seznam publikacij v administraciji.

Ob nalaganju spletnega obrazca moramo zagotoviti še ostale podatke kot so avtorji, metode, živila, viri, in dinamični podatki, katere je uporabnik definiral naknadno. Za vse te podatke imamo na voljo posamezne vire API, na katere pošljemo zahtevek za pridobitev teh podatkov.

Safe Food [LIST USERS](#) [ARTICLES](#) [AUTHORS](#) [METHODS](#) [FOODSTUFF](#) [SOURCES](#) [FIELDS](#)

ARTICLE

RELEASE DATE

2022-01-27

×

SOURCE

To je nova knjiga

×

+

TITLE

Prediction of various freshness indicators in fish fillets by one multispectral imaging system

YEAR

2019

URL

www.nature.com/scientificreports

AUTHORS

Sara Khoshnoudi-Nia × Marzleh Moosavi-Nasab ×

×

▼

METHODS

BP-ANN × LS-SVM × MLR × nonlinear regression × linear regression ×

×

▼

FOODSTUFF

fish ×

×

▼

TEMPERATURE RANGE

MIN

2

2°C: 10°C:

MAX

10

COMMENT

a simple multispectral imaging (430–1010 nm) system

ATTACHMENT

Attach a file

Screenshot 2022-01-15 at 21.05.22.png

EXTRA FIELDS

META KEYWORDS

sensory score × Psychrotrophic Plate Count (PPC) × Total-Volatile Basic Nitrogen (TVB-N) ×

×

▼

LABEL	TYPE	CREATED AT
Meta Keywords	SELECT	17d ago
Research Type	CHECKBOX	11d ago
Spletna povezava	INPUT	1d ago

SAVE

Slika 4.8: Spletni obrazec za urejanje ali vnos nove publikacije.

Na spletnem obrazcu je mogoče dodajati in urejati naslednje podatke:

Datum objave - polje za vpis datuma, ki določa kdaj naj bo publikacija na voljo ostalim uporabnikom. Ta parameter je na voljo le administratorjem.

Vir - izbirno polje za izbiro vira, na katerega se nanaša publikacija. S klikom na polje se odpre spustno okno z vnaprej vnesenimi viri. Imamo tudi možnost iskanja vira po njegovem naslovu. V primeru, da vir še ne obstaja, imamo na voljo gumb za dodajanje (+) novega vira. S klikom na gumb se odpre okno za kreiranje novega vira (Slika 4.9). Ob shranjevanju vira se samodejno izpolni polje z virom, ki je bil na novo vnesen.

SOURCE

TYPE

☐ BOOK ☐ MAGAZINE ☐ WEB

NAME

URL

DESCRIPTION

SAVE CANCEL

Slika 4.9: Obrazec za dodajanje novega vira.

Naslov - tekstovno polje, ki vsebuje naslov publikacije.

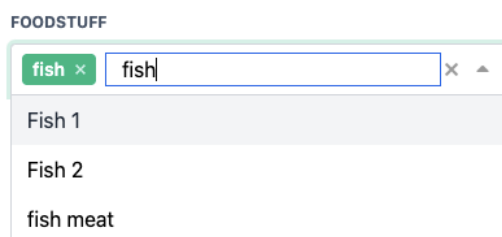
Leto - numerično polje, ki vsebuje leto izdaje publikacije.

Spletni vir - povezava na vir, kjer lahko pridobimo več informacij o publikaciji.

Avtorji - izbirno polje za izbiro enega ali več avtorjev. Ob kliku na polje se odpre spustni meni s seznamom avtorjev po katerih lahko izvajamo iskanje. V primeru, da avtor ne obstaja, imamo možnost dodajanja novega avtorja z vpisom njegovega imena. Ob potrditvi bo avtor avtomatsko dodan v zalednem sistemu in izbran v izbirnem polju.

Metode - uporablja se enaka komponenta kot pri polju za izbiro avtorjev, le da se v tem primeru izbirajo metode.

Živila - uporablja se enaka komponenta kot pri polju za izbiro avtorjev, le da se v tem primeru izbirajo živila (Slika 4.10).



Slika 4.10: Spustni meni s predlaganimi vrednostmi.

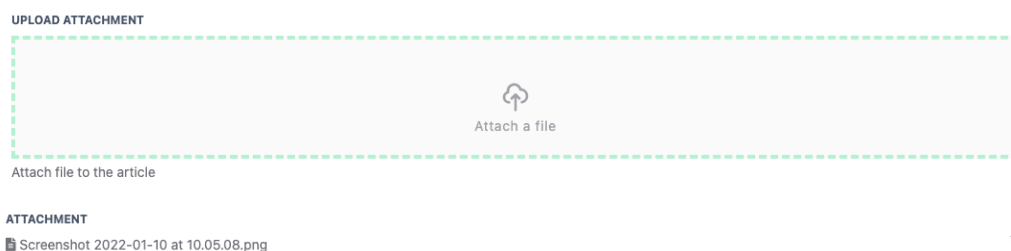
Temperaturno območje - predstavlja območje dveh števil, katerega je mogoče izbrati s horizontalnim drsnikom, ali pa z vpisom vrednosti v posamezna polja (minimalno in maksimalno vrednost v °C) (Slika 4.11).



Slika 4.11: Komponenta za določanje vrednosti v območju.

Komentar - tekstovno polje, kjer je mogoče vnesti daljše besedilo, do 255 znakov.

Priponka - polje za odlaganje datoteke. Na voljo je izbira datotek, ki se pripne k publikaciji. Prikaže se znotraj komponente in jo je mogoče odstraniti (Slika 4.12).



Slika 4.12: Komponenta za nalaganje datotek.

Dodatna polja - seznam dodatnih polj, ki jih želimo dodati k publikaciji. Ob kliku nanje se pojavi možnost za definiranje vrednosti tega podatka. Vnosno polje se prikaže različno glede na tip podatka, ki ga izberemo (Slika 4.13).

EXTRA FIELDS

META KEYWORDS

×
 ×
 ×
 ×

SPLETNA POVEZAVA

+

LABEL	TYPE	CREATED AT
Meta Keywords	SELECT	16d ago
Spletna povezava	INPUT	9s ago
Research Type	CHECKBOX	10d ago

ADD +

Slika 4.13: Prikaz uporabe dodanega parametra, med urejanjem publikacije.

S klikom na gumb „Save“, ki se nahaja spodaj desno, sprožimo zahtevek za shranjevanje vnesenih podatkov. Osnovna validacija se izvaja že med

samim vnosom podatkov na čelnem delu. V primeru napačnega vnosa podatkov se na obrazcu poleg posameznih vnosnih polj prikaže sporočilo z opisom napake. Ob uspešnem shranjevanju se pokaže sporočilo, da je bila publikacija uspešno shranjena.

4.5 Prikaz publikacije

Publikacija je na strani vizualno predstavljena z vso vneseno vsebino. Na vrhu je napisan naslov publikacije, ki je podan kot odebeljeno besedilo, saj je najbolj pomemben podatek za uporabnika. Pod naslovom so izpisani vsi avtorji, ki so sodelovali pri publikaciji. Vsak podatek je predstavljen z imenom in nato s svojo vrednostjo (Slika 4.14).

Sledijo podatki o sami publikaciji. Vir je predstavljen s tipom, imenom in povezavo, če jo ima. V primeru, da spletna povezava ni bila vnesena leta ni prikazana. Ker je lahko vnesenih metod in živil za posamezno publikacijo mnogo, so prikazane kot vrednosti, ločene z vejico. Na skrajno desni strani je predstavljena vrednost temperature, v primeru, da je ta definirana za posamezno publikacijo.

Publikacija lahko vsebuje tudi komentar, ki je prikazan kot daljše tekstovno polje. Poleg komentarja se nahajajo še ostale povezave do zunanjih virov, če so bili vneseni med samim vnosom publikacije. V primeru, da ima publikacija pripete datoteke, se prikažejo kot ikona datoteke. S klikom na ikono se nam odpre novo okno s povezavo do datoteke. Datoteka je posredovana preko zalednega sistema in jo je mogoče prenesti v samem brskalniku.

Authors: Atanu Chaudhuri et. al

Type: WEB

Name: The International Journal of Logistics Management 29(1)

<https://www.researchgate.net/publication/322530435> Decision making in cold chain logistics using data analytics a literature review

review paper

Min: 0°C

Max: 0°C

fish meat, dairy products

Review of 38 selected research articles, published between 2000 and 2016

<https://www.researchgate.net/publication/322530435> Decision making in cold chain logistics using data analytics: a literature review

Attachment URL:

cold chain logistics, shelf life

Slika 4.14: Prikaz publikacije z vnesenimi podatki o avtorjih, metodah in ostalih podatkov.

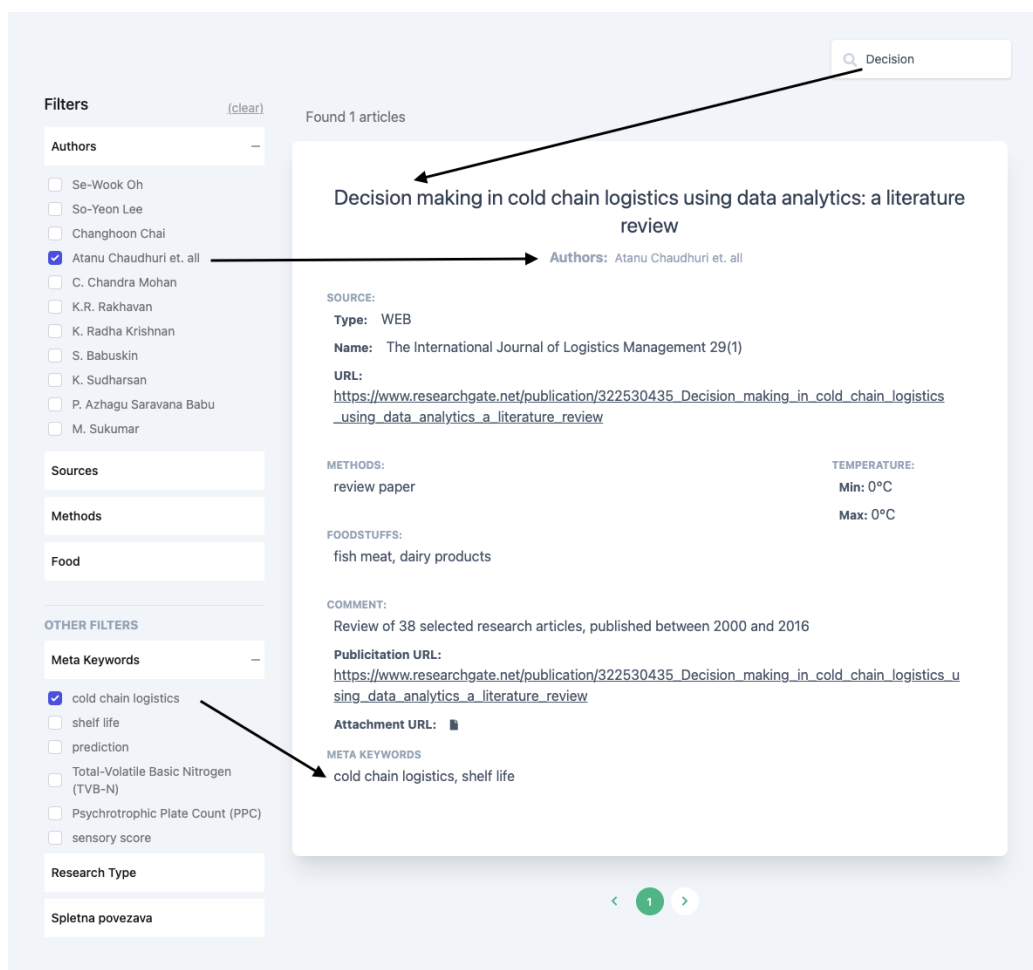
4.6 Iskanje publikacij

Vse vnesene publikacije so uporabniku prijazno prikazane na eni strani. Za pregledovanje ima na voljo številne filtre, kar mu omogoča, da s tem hitreje najde publikacijo, ki jo išče. Iskanje je omogočeno po vseh možnih podatkih, vnesenih za posamezno publikacijo.

Stran je razdeljena na dva dela. Na levem delu so uporabniku na voljo filtri, katere lahko uporabi za iskanje po publikacijah, na desnem delu pa se uporabniku te publikacije prikažejo. V primeru, da je najdenih publikacij za izbrane filtre veliko, imamo na dnu možnost sprehajanja med posameznimi stranmi, na vrhu pa se izpiše število vseh publikacij, ki ustrezajo izbranim filtrom.

Uporaba filtrov je enostavna. Na levem delu so izpisani vsi podatki, po katerih je mogoče iskati. Ob kliku na filter se prikaže nabor vrednosti, katere lahko uporabnik izbira, in s tem vklopi ali izklopi filter. V zgornjem delu strani je na voljo tudi iskanje po tekstovnem nizu besed, kjer uporabnik vnese niz za izvedbo filtriranja. Vsebina na strani se samodejno osveži in prikaže publikacije, ki ustrezajo izbranim filtrom.

Na sliki 4.15 je prikazana praktična uporaba filtrov. V njihovi kombinaciji smo dobili rezultat, ki upošteva vse našete filtre.



Slika 4.15: Praktični prikaz uporabe filtrov na strani za iskanje publikacij.

Poglavje 5

Sklepne ugotovitve

Med razvojem aplikacije, ki smo jo opisali skozi celotno diplomsko delo, sem se hotel naučiti čim več. Spoznal sem nov programski jezik - Go in nov arhitekturni koncept mikrostoritev (ang. microservice). Omenjeni dve stvari sta me dodatno spodbudili k delu in spoznavanju novih orodij in tehnologij na področju razvoja programske opreme in pa tudi same organizacije dela.

Z vpeljavo arhitekturnega koncepta mikrostoritev je bila izdelana spletna aplikacija. Pri izdelavi diplomske naloge je bil upoštevan celoten postopek, od ideje in načrtovanja, do same izvedbe in testiranja aplikacije. Pri njenem razvoju in izvedbi smo se soočili z različnimi omejitvami in težavami, katere smo uspešno odpravili in zagotovili, da aplikacija deluje hitro in brez napak.

Nadaljnji razvoj aplikacije bi lahko potekal v smeri brezglavega urejevalnika vsebin, kar pomeni, da bi zasnovali aplikacijski vmesnik za dinamično urejanje podatkov. Zaradi arhitekturne zasnove je enostavno izluščiti del dinamičnega dodajanja podatkov. Z novo storitvijo, ki bi skrbela za definiranje, shranjevanje, urejanje in brisanje dinamičnih podatkov bi prišli do rešitve, ki bi podpirala uporabo za različne vrste podatkov, ne samo za naš trenutni primer, to je za vnašanje publikacij.

Razvili bi lahko tudi novo storitev s katero bi avtomatizirali vnašanje podatkov v aplikacijo. Ta bi skrbela za pridobivanje publikacij iz različnih virov (API) in jih preko aplikacijskega vmesnika vnašala v našo aplikacijo.

Dodali bi lahko še izvoz in generiranje ustreznega dokumenta za publikacijo.

Literatura

- [1] “Food safety centre.” Dosegljivo: <http://www.foodsafetycentre.com.au/tools.php>. [Dostopano: 09. 01. 2022].
- [2] “Mendeley reference manager.” Dosegljivo: <https://www.mendeley.com/reference-management/mendeley-desktop>. [Dostopano: 20. 01. 2022].
- [3] “Zotero - your personal research assistant.” Dosegljivo: <https://www.zotero.org/>. [Dostopano: 20. 01. 2022].
- [4] “Waterfall model.” Dosegljivo: http://myprojects.kostigoff.net/methodology/development_models/pages/waterfall.htm. [Dostopano: 09. 01. 2022].
- [5] “A short history of Git.” Dosegljivo: <http://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git>. [Dostopano: 27. 01. 2022].
- [6] J. Nickoloff, “Docker in action,” in *Docker in Action*, Manning Publications Co., 2016.
- [7] “Linux containers.” Dosegljivo: <https://linuxcontainers.org/lxd/docs/master/>, note = [Dostopano: 10. 01. 2022],.
- [8] “Microservices guide.” Dosegljivo: <https://www.martinfowler.com/microservices/>. [Dostopano: 11. 01. 2022].

-
- [9] O. Corporation, “A relational database overview.” Dosegljivo: <https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html>. [Dostopano: 15. 01. 2022].
- [10] “Dodatek za geolokacijske funkcionalnosti za postgresql - postgis.” Dosegljivo: <https://postgis.net/>. [Dostopano: 09. 01. 2022].
- [11] D. T. I. Microsoft, Refactored Networks LLC, “A universally unique identifier (uuid) urn namespace.” Dosegljivo: <https://datatracker.ietf.org/doc/html/rfc4122>. [Dostopano: 09. 01. 2022].
- [12] “Rojstnodnevni paradoks.” Dosegljivo: https://en.wikipedia.org/wiki/Birthday_problem. [Dostopano: 09. 01. 2022].
- [13] A. DasGupta, “The matching birthday problem: a contemporary review,” in *The matching, birthday and the strong birthday problem: a contemporary review*, elsevier, 2003.
- [14] “Uuid collisions.” Dosegljivo: https://en.wikipedia.org/wiki/Universally_unique_identifier#Collisions. [Dostopano: 09. 01. 2022].
- [15] “Firabese tools.” Dosegljivo: <https://firebase.google.com/docs/build>. [Dostopano: 15. 01. 2022].
- [16] “Firebase limits.” Dosegljivo: <https://firebase.google.com/docs/auth/limits>. [Dostopano: 15. 01. 2022].
- [17] “What is vue.js?.” Dosegljivo: <https://vuejs.org/v2/guide/index.html#What-is-Vue-js>. [Dostopano: 09. 01. 2022].
- [18] J. R. R. Fielding, M. Nottingham, “Hypertext transfer protocol – http/1.1.” Dosegljivo: <https://datatracker.ietf.org/doc/html/rfc2616#section-1.4>, June 1999. [Dostopano: 15. 01. 2022].

-
- [19] D. Crockford, “The application/json media type for javascript object notation (json).” Dosegljivo: <https://www.ietf.org/rfc/rfc4627.txt>. [Dostopano: 12. 01. 2022].