SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Toni Velkovski

APLIKACIJA ZA PRAĆENJE I POMOĆ U ODRŽAVANJU RASTA BILJKE

PROJEKT

TEORIJA BAZA PODATAKA

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Toni Velkovski

Matični broj: 0016146666

Studij: Informacijsko i programsko inženjerstvo

APLIKACIJA ZA PRAĆENJE I POMOĆ U ODRŽAVANJU RASTA BILJKE

PROJEKT

Mentor:

dr. sc. Bogdan Okreša Đurić

Varaždin, siječanj 2024.

Toni Velkovski

Izjava o izvornosti

Izjavljujem da je ovaj projekt izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI Radovi

Sažetak

Ovaj projektni rad bavi se realizacijom aplikacije za praćenje i pomoć u održavanju rasta biljke. U prvome dijelu rada opisane su korištene tehnologije, uključujući PostgreSQL bazu podataka te zatim kratak uvod vezan uz aktivne i temporalne baze podataka. Prikazane su i zanimljivosti PostgreSQL baze podatak te određeni napredniji koncepti koji bi se mogli primijeniti na većim sustavima te s većim brojem podataka unutar relacija. Drugi dio rada odnosi se na praktičnu primjenu i implementaciju te opis ključnih dijelova implementacije. Na kraju projektnog rada, dan je prikaz i primjeri korištenja implementirane aplikacije kao i kratak osvrt, zaključak vezan uz izradu rada.

Ključne riječi: PostgreSQL; Node.js; JavaScript; Okidač; vremenski tipovi;

Sadržaj

1. Opis aplikacijske domene

Aplikacijska domena koja se želi postići ovim projektom je praćenje i pomoć u održavanju rasta biljke. Sama aplikacija cilja ljude koji se bave uzgojem biljaka u svom vrtu ili ljudi koji jednostavno žele pratiti rast biljaka u svom domu. Specifičnosti ove domene uključuju vrste biljaka, biljke, galeriju slika te popis zadataka.

Vrste biljaka je početna stavka koja služi za grupiranje različitih biljaka unutar jedne vrste. Biljke kao takve su elementi oko kojih se aplikacija bazira, a sadržane su unutar svoje vrste. Galerija slika predstavlja slike tih biljaka koje mogu biti slike dok su biljke kupljene, slike zalijevanja, slike biljaka nakon određenog vremena itd. Popis zadataka predstavlja korisnikovo stvaranje zadataka koje želi, odnosno koje je potrebno napraviti (npr. zalijevanje biljke na određeni datum). Iz navedenog popisa domene, mogu se prepoznati četiri relacije koje će biti potrebne za samu implementaciju: vrsta biljke, biljka, slike biljke te zadaci.

Odabir tehnologija za razvoj aplikacije uključuje korištenje Node.js za pozadinski dio Web aplikacije, PostgreSQL kao sustav za upravljanje bazama podataka, te kombinaciju HTML, CSS i JavaScript za klijentski dio Web aplikacije. Navedene tehnologije odabrane su zbog njihove široke korištenosti i jednostavne kompatibilnosti između sebe. Programski jezik JavaScript najpoznatiji je i najkorišteniji programski jezik za programiranje na strani klijenta, dok je HTML standardni jezik za označavanje i strukturiranje sadržaja na Webu, a CSS standardni jezik za stilizaciju sadržaja Web stranica. Node.js je izvršno okruženje temeljeno na JavaScript-u, a dizajnirano je za izvođenje koda na strani poslužitelja. Odabran je zbog svoje efikasnosti i brzine, praktičnosti i jednostavne korištenosti te zbog svoje velike zajednice i ekosustava paketa.

Prethodno spomenuta zajednica jedan je od razloga velike korištenosti Node.js tehnologije, Node.js ima ogromnu zajednicu razvijatelja koji razvijaju razne pakete te tako pridonose zajednici i ekosustavu paketa. Node Package Manager (npm) je upravitelj paketa za Node.js, a služi za instalaciju, upravljanje i dijeljenje paketa ili JavaScript modula. Jedan od takvih paketa je i *node-postgres* paket, poznatiji kao *pg*.

Node-postgres je Node.js modul koji omogućuje komunikaciju između Node.js aplikacije i PostgreSQL baze podataka. Pomoću ovog modula, razvojnim programerima omogućeno je jednostavno izvršavanje SQL upita, upravljanje transakcijama i komunikacija s PostgreSQL bazom podataka kroz kod. Sam paket moguće je jednostavno preuzeti pomoću naredbe *npm install pg*. [1] Naravno, prije nego se instalira *pg* modul, potrebno je inicijalizirati Node.js projekt pomoću naredbe *npm init*. Ova naredba stvara *package.json* datoteku koja sadrži informacije o osnovnim postavkama projekta, kao što su naziv projekta, opis, verzija, autor, licenca i sl. Datoteka također sadrži i "scripts" te "dependencies" polja. Unutar scripts moguće je postaviti vlastite skripte koje će se pokretati pomoću *npm* naredbe, a unutar dependencies polja sadržan je popis svih ovisnosti koje su preuzete i korištene u projektu. Nakon instaliranja *pg* modula, on je prikazan kao ovisnost projekta.

Sav programski kod moguće je preuzeti preko GitHub-a koji je dostupan na linku. Unutar README.md dokumenta dostupna je dokumentacija za instaliranje potrebnog i pokretanje aplikacije.

2. Teorijski uvod

Baza podataka koja je korištena za izradu ovog projekta bazira se na dvije vrste osnovnih pristupa bazama podataka: aktivne i temporalne baze podataka.

Aktivne baze podataka su baze podataka dizajnirane tako da automatski reagiraju na određene događaje ili promjene unutar sustava. Takve baze podataka podržavaju automatsko izvršavanje određenih akcija kada se dogode određeni uvjeti. Akcije nad bazom podataka definirane su kao akcije ažuriranja, akcije pristupa podacima i akcije transakcija. Akcije također mogu biti i pohranjene procedure pisane u nekom proceduralnom programskom jeziku. [2] Jedan od takvih jezika je Pl/pgSQL proceduralni jezik koji se koristi u PostgreSQL bazi podataka. To je proširenje SQL-a koje omogućuje pisanje pohranjenih procedura, funkcija, okidača (engl. *trigger*) i drugih programa koji se izvršavaju nad bazom podataka.

Aktivne baze podataka mogu se implementirati pomoću ranije spomenutih okidača ili pomoću aktivnih pravila (engl. *rule*). U novijim verzijama PostgreSQL-a aktivna pravila su sve više zamijenjena upotrebom okidača koji se sve više koriste. Okidači su skupovi pravila ili procedura koje se aktiviraju kada se određeni događaji dogode. Postavljaju se na određene tablice te nakon zadovoljenja uvjeta okidača, izvršavaju se zadane akcije.

Sintaksa okidača u PostgreSQL-u izgleda [3]:

```
1 CREATE [ OR REPLACE ] [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } {
  ↪  event [ OR ... ] }
2     ON table_name
3     [ FROM referenced_table_name ]
4     [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
5     [ REFERENCING { { OLD | NEW } TABLE [ AS ] transition_relation_name } [ ... ] ]
6     [ FOR [ EACH ] { ROW | STATEMENT } ]
7     [ WHEN ( condition ) ]
8     EXECUTE { FUNCTION | PROCEDURE } function_name ( arguments )
```

Isječak koda 1: Sintaksa okidača

Kao i sve, aktivne baze podataka imaju svoje prednosti i nedostatke. Prednosti aktivnih baza podataka:

- automatsko reagiranje na promjenu podataka
- lakša implementacija poslovnih pravila
- poboljšana konzistentnost podataka

Dok bi neki od nedostataka bili:

- kompleksnije održavanje i upravljanje
- veća složenost performansi sustava
- poboljšana konzistentnost podataka

Temporalne baze podataka su baze podataka dizajnirane tako da podržavaju vremenski aspekt podataka, odnosno da prate promjene podataka kroz vrijeme. Temporalne baze podataka imaju ugrađene vremenske aspekte podataka te one omoćavaju rad s takvim podacima. [4]

Unutar PostgreSQL-a temporalne baze podataka podržane su u nekim aspektima, ponajviše s njihovim tipovima podataka. Vremenski tipovi podataka u PostgreSQL-u su: timestamp, date, time, interval, a postoje i mnoge operacije nad tim tipovima podataka poput zbrajanja, oduzimanja datuma i sl., a sam popis moguće je vidjeti u dokumentaciji. [5]

Prednosti temporalnih baza podataka bile bi:

- praćenje povijesti podataka
- korisno u sustavima gdje su bitne vremenske informacije

Dok bi neki od nedostataka bili:

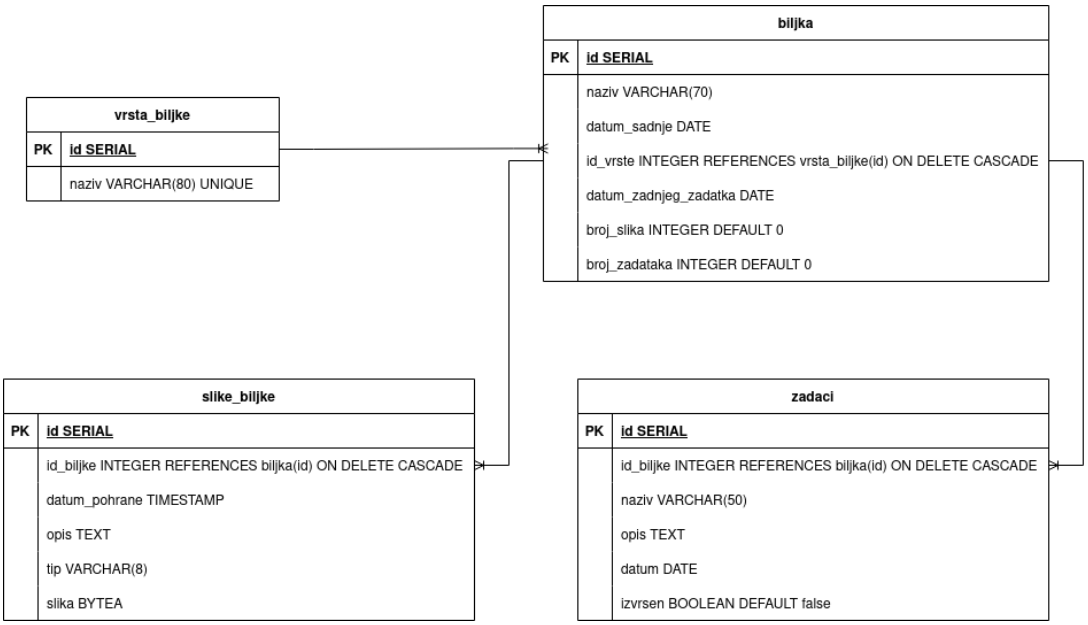
- složenije modeliranje i upravljanje podacima
- povećana potreba za prostorom za pohranu zbog vremenskih informacija

Aktivne baze podataka često se koriste u sustavima gdje je potrebna trenutna i automatska reakcija na određene promjene. To je korisno u poslovnim okruženjima gdje je potrebno odmah reagirati na određene događaje ili gdje je potrebno na neki događaj pokrenuti određenu proceduru i sl. Temporalne baze podataka su korisne u situacijama gdje je važno pratiti promjene kroz vrijeme, kao što je analiza trendova, praćenje povijesti podataka ili upravljanje verzijama podataka.

Za aplikacijsku domenu ovog projekta te dvije vrste pristupa bazama podataka imaju navedene prednosti koje ne bi mogli iskoristiti s npr. polustrukturiranim bazama podataka. Jedna od najvećih prednosti je korištenje okidača za aplikacijsku domenu, budući da je moguće vrlo jednostavno postaviti okidače na određeni događaj (npr. okidač na unos slika biljke u bazu podataka). Također i vremenski tipovi poput datuma koji su od velike važnosti za same zadatke.

3. Model baze podataka

Baza podataka, nazvana *odrzavanje_biljaka* sadrži četiri relacije koje su ranije spomenute, a to su: *vrsta_biljke*, *biljka*, *slike_biljke* i *zadaci*. Navedene relacije i njihove atribute moguće je vidjeti na slici ERA modela (Slika 1) koji opisuje atribute relacija i njihove odnose.



Slika 1: ERA model **[autorski rad]**

Sa slike je moguće vidjeti kako su relacije *vrsta_biljke*, *slike_biljke* i *zadaci* povezane sa relacijom *biljka* na iduči način:

- relacija *vrsta_biljke* povezana je s jedan naprama više s relacijom *biljka* gdje je više na strani *biljka*, jer više biljaka može biti sadržano unutar jedne vrste
- relacija *slike_biljke* povezana je s više naprama jedan s relacijom *biljka* gdje je jedan na strani *biljka* jer jedna biljka može imati više slika
- relacija *zadaci* povezana je s više naprama jedan s relacijom *biljka* gdje je jedan na strani *biljka*, jer jedna biljka može imati više zadataka

Ključna riječ *REFERENCES* označava vanjski ključ, pa tako su relacije textitvrsta_biljke, *slike_biljke* i *zadaci* povezane s tablicom *biljka* preko vanjskog ključa.

Također postavljena su i ograničenja *ON DELETE CASCADE* nad vanjskim ključevima tablica, budući da je potrebno osigurati kaskadna brisaja. Na primjeru, ako obrišemo određenu biljku, potrebno je kaskadno obrisati i sve slike koje su vezane uz tu određenu biljku.

4. Implementacija

Kao što je spomenuto u opisu aplikacijske domene, korištene tehnologije za implementaciju aplikacije za praćenje i pomoć u održavanju rasta biljke su: PostgreSQL baza podataka, Node.js za pozadinski dio Web aplikacije te HTML, CSS i JavaScript za klijentski dio Web aplikacije. Tim redom biti će i opisane unutar ovog poglavlja.

4.1. Baza podataka za održavanje biljaka

Unutar *Create.sql* skripte moguće je vidjeti sav kod koji je potreban za kreiranje baze podataka, tablica, okidača i opcionalno indeksa. Skriptu je moguće izvršiti kopiranjem SQL koda ili na način da se unutar terminala pokrene naredba *npm run initDB* koja će se povezati na PostgreSQL bazu podataka. Prilikom ovakvog pokretanja, potrebno je u datoteci *package.json*, unutar polja *scripts* izmijeniti podatke da se poklapaju s PostgreSQL bazom podataka (korisnik, lozinka korisnika i sl.).

Kreiranje tablica izvršava se jednostavnom *CREATE TABLE* sintaksom: Iz isječka koda

```
CREATE TABLE zadaci (
    id SERIAL PRIMARY KEY,
    id_biljke INTEGER REFERENCES biljka(id) ON DELETE CASCADE,
    naziv VARCHAR(50),
    opis TEXT,
    datum DATE,
    izvrsen BOOLEAN DEFAULT false
);
```

Isječak koda 2: Kreiraje tablice zadaci

moguće je vidjeti da je primarni ključ *id* koji je tipa 'SERIAL', što znači da se svakim unosom povećava za jedan, vanjski ključ je *id_biljke*, *datum* je tipa 'DATE', a varijabla *izvrsen* koja je tipa 'BOOLEAN' ima za predefiniranu vrijednost 'false'.

kodom, dok je za kreiranje okidača potrebno kreirati funkciju koju će okidač pozivati. Implementirana su tri okidača:

- okidač za ažuriranje slika
- okidač za ažuriranje broja zadataka
- okidač za ažuriranje datuma nadolazećih događaja

Okidač za ažuriranje broja slika moguće je vidjeti u idućem isječku koda:

Okidač *broj_slika_azuriranje* okida se nakon unosa ili brisanja podataka nad tablicom *slike_biljke* te poziva *azuriraj_broj_slika()* funkciju koja ovisno o događaju (unos ili brisanje) povećava odnosno smanjuje broj slika. Zanimljivo je da bi se ova funkcija, odnosno okidač mogao odvojiti na dva zasebna, no PostgreSQL ima svoju specifičnu varijablu 'TG_OP' koja

```sql
CREATE OR REPLACE FUNCTION azuriraj_broj_slika()
RETURNS TRIGGER AS $$
BEGIN
  IF TG_OP = 'INSERT' THEN
    UPDATE biljka
    SET broj_slika = broj_slika + 1
    WHERE id = NEW.id_biljke;
  ELSIF TG_OP = 'DELETE' THEN
    UPDATE biljka
    SET broj_slika = broj_slika - 1
    WHERE id = OLD.id_biljke;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER broj_slika_azuriranje
    AFTER INSERT OR DELETE ON slike_biljke
    FOR EACH ROW
EXECUTE PROCEDURE azuriraj_broj_slika();
```

Isječak koda 3: Okidač za ažuriranje broja slika

se koristi u okidačima kako bi prikazala tip događaja koji je aktivirao okidač. TG_OP varijabla može imati vrijednosti 'INSERT', 'UPDATE' ili 'DELETE'.

Okidač za ažuriranje broja zadatak i okidač za ažuriranje datuma nadolazećih događaja nešto su jednostavniji, no okidač *datum_nadolazecih_zadataka_azuriranje* poziva funkciju *azuriraj_datum_nadolazecih_zadataka* koja unutar sebe sadrži dio koda koji koristi *MIN()* funkciju za pronalazak najmanjeg datuma te koristi *CURRENT_DATE* funkciju koja u PostgreSQL-u predstavlja funkciju koja dohvaća trenutni datum.

```sql
CREATE OR REPLACE FUNCTION azuriraj_datum_nadolazecih_zadataka()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE biljka
  SET datum_zadnjeg_zadatka = (
    SELECT MIN(datum)
    FROM zadaci
    WHERE id_biljke = NEW.id_biljke AND datum > CURRENT_DATE
  )
  WHERE id = NEW.id_biljke;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER datum_nadolazecih_zadataka_azuriranje
    AFTER INSERT OR UPDATE OF datum ON zadaci
    FOR EACH ROW
EXECUTE PROCEDURE azuriraj_datum_nadolazecih_zadataka();
```

Isječak koda 4: Funkcija za azuriranje datuma nadolazecih dogadaja

Na kraju tu funkciju poziva okidač *datum_nadolazecih_zadataka_azuriranje* koji se okida na događaje unosa ili ažuriranja datuma nad tablicom zadaci.

4.2. Prikaz koda za pozadinski dio Web aplikacije - Node.js kod

Sav Node.js kod sadržan je unutar *app.js* datoteke koja se zapravo i pokreće pomoću naredbe: *node app.js* ili jednostavnije pomoću naredbe *npm start* budući da je postavljena kao startna datoteka. Sama datoteka sadrži određene module koje je potrebno preuzeti, a sadržani su unutar *dependencies* polja unutar *package.json* datoteke. To je moguće napraviti na način da se u terminalu pokrene *npm i* naredba koja će preuzeti sve potrebne module.

Intalirane module potrebno je uključiti u kod, a to je prikazano na idućem isječku:

```
1 const express = require('express');
2 const { Pool } = require('pg');
3 const multer = require('multer');
4 const mime = require('mime-types');
5 const bodyParser = require('body-parser');
6 require('dotenv').config();
```

Isječak koda 5: Korišteni moduli

Korišteni moduli su:

- **express modul** - Node.js modul za izradu Web aplikacija i API-ja
- **Pool modul** iz pg paketa
- **multer modul** za obradu HTTP zahtjeva s višestrukim dijelovima
- **mime modul** koji omogućava mapiranje ekstenzija datoteka na MIME tipove
- **body-parser modul** za parsiranje HTTP zahtjeva
- **dotenv modul** za učitavanje konfiguracijskih varijabli iz .env datoteke

Dio koda vezan za pokretanje Express Web servera i osluškivanje porta:

```
1 app.listen(process.env.APP_PORT, () => {
2     console.log('Server listening on port ' + process.env.APP_PORT);
3 });
```

Isječak koda 6: Pokretanje Web servera

Nakon pokretanja, Web server biti će dostupan na portu koji je zapisan u varijabli 'APP_PORT', koja je sadržana unutar *.env* datoteke te joj je jednostavno promijeniti vrijednost ukoliko je to potrebno.

4.2.1. Definiranje ruta Web aplikacije

Rute u Web aplikaciji definiraju odnose između URL-ova i odgovarajućih resursa ili funkcionalnosti na Web serveru. Kada korisnik pristupi određenom URL-u putem Web preglednika, ruta se koristi za mapiranje tog URL-a na određenu akciju ili prikazivanje određenog sadržaja. Unutar Express modula, rute je moguće definirati pomoću metoda: app.get, app.post, app.put i sl., ovisno o HTTP metodi za koju se ruta odnosi.

Idući isječak koda prikazuje definiranje ruta nad kojima se šalju određene HTML datoteke:

```
app.get('/', (req, res) => res.sendFile(__dirname + "/static/html/index.html"));
app.get('/plant_type', (req, res) => res.sendFile(__dirname +
↪  "/static/html/plant_type.html"));
app.get('/image_gallery', (req, res) => res.sendFile(__dirname +
↪  "/static/html/image_gallery.html"));
app.get('/task_list', (req, res) => res.sendFile(__dirname +
↪  "/static/html/task_list.html"));
app.get('/type/:id', async (req, res) => {
    res.sendFile(__dirname + '/static/html/plant_type.html');
});
```

Isječak koda 7: Rute koje šalju HTML datoteku

Svaki poziv app.get služi za registraciju rute koja će biti izvršena kad primi HTTP GET zahtjev. Na primjer, ruta '/' odnosi se na početnu stranicu Web aplikacije, a kad se taj URL posjeti, Web server će poslati datoteku *index.html* koja se nalazi u direktoriju *static/html*.

4.2.2. Pg modul i izvršavanje upita

Ostatak Node.js programskog koda svodi se na definiranje endpointova i određenih radnji nad njima. Prije toga, potrebno je iskoristiti *pg* modul za stvaranje veze s PostgreSQL bazom podataka. Idući kod prikazuje stvarnje pool objekta:

```
const pool = new Pool({
    user: process.env.PG_USER,
    host: process.env.PG_HOST,
    database: process.env.PG_DATABASE,
    password: process.env.PG_PASSWORD,
    port: process.env.PG_PORT
});
```

Isječak koda 8: Pool objekt

Stvoreni pool objekt predstavlja vezu prema PostgreSQL bazi podataka, a sadrži iduće parametre [1]:

- **user** - korisničko ime koje se koristi prilikom autentifikacije na PostgreSQL bazu podataka
- **host** - adresa poslužitelja na kojem se nalazi PostgreSQL baza podataka

- **database** - ime baze podataka s kojom se želi uspostaviti veza
- **password** - lozinka koja se koristi za autentifikaciju prilikom spajanja s bazom podataka
- **port** - port na kojem PostgreSQL osluškuje dolazne veze

Navedeni parametri popunjeni su podacima iz *.env* datoteke pod odgovarajućim varijablama te ukoliko postoji razlika, potrebno je izmijeniti unutar *.env* datoteke.

Ranije spomenuti endpoint u kontekstu Web razvoja predstavlja krajnju točku (URL) koja se koristi za pristup određenim uslugama ili resursima na Web serveru. Svaki endpoint može biti povezan s određenom HTTP metodom (GET, POST, PUT itd.) koje određuju vrstu operacije koja će se izvršiti na tom resursu. Jedan takav endpoint prikazan je idućim kodom:

```
app.post('/addPlantType', async (req, res) => {
    const { typeName } = req.body;

    try {
        const result = await pool.query(
            'INSERT INTO vrsta_biljke(naziv) VALUES ($1) RETURNING *',
            [typeName]
        );
        res.status(201).json({ success: true, novaVrsta: result.rows[0] });
    } catch (error) {
        console.error(error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});
```

Isječak koda 9: Dodavanje vrste biljke u bazu podataka

Iz prikazanog koda može se vidjeti da se koristi HTTP POST metoda (app.**post**) koja označava unos. Unos je prikazan klasičnim INSERT upitom koji je unutar *pool.query*, odnosno *pool* veze i *query* koji označava izvršavanje upita. Zanimljivo je napomenuti 'RETURNING *' na kraju upita, što je PostgreSQL sintaksa koja se u ovome slučaju koristi kod 'INSERT' upita kako bi se dobio sadržaj novih redova nakon izvršavanja tog upita.

Idući isječak koda prikazuje prikazivanje svih podataka o biljci te podatak o nazivu vrste biljke, pa je potrebno koristiti JOIN operaciju. Filtriraju se samo oni rezultati gdje je id vrste biljke jednak 'typeId' varijabli, koji dobivamo preko same Web aplikacije.

Na kraju upita postavljeno je 'ORDER BY biljka.id' kako bi sortirao biljke po njihovom id-u. To možda u ovom slučaju inače ne bi bilo potrebno, no postavljeno je radi jedne zanimljive specifičnosti koju PostgreSQL ima, a vezana je uz ažuriranje, odnosno 'UPDATE' upit.

```
app.get('/plantType/:id', async (req, res) => {
    const typeId = req.params.id;

    try {
        const result = await pool.query(
            `SELECT biljka.id, biljka.naziv, biljka.datum_sadnje, biljka.id_vrste,
            ↪  biljka.datum_zadnjeg_zadatka, biljka.broj_slika,
            biljka.broj_zadataka, vrsta_biljke.naziv AS naziv_vrste
            FROM vrsta_biljke
            LEFT JOIN biljka ON vrsta_biljke.id = biljka.id_vrste
            WHERE vrsta_biljke.id = $1
            ORDER BY biljka.id;`,
            [typeId]
        );

        const plants = result.rows;

        res.json({ plants });
    } catch (error) {
        console.error('Error fetching data:', error);
        res.status(500).send('Internal Server Error');
    }
});
```

Isječak koda 10: Dohvat podataka o biljci određene vrste

Naime nakon što se izvrši 'UPDATE' naredba, ažurirani redak je unutar tablice na "dnu". To je vjerojatno zato što PostgreSQL radi ažuriranje na principu brisanja i upisivanja - obriše prethodni element a unese novi, ažurirani element. Slika (Slika 2) prikazuje kako izgleda redak nakon ažuriranja u tablici.

```
odrzavanje_biljaka=#
odrzavanje_biljaka=# SELECT * FROM vrsta_biljke;
 id |   naziv
----+-----------
  1 | Ruža
  2 | Tulipan
  3 | Suncokret
(3 rows)

odrzavanje_biljaka=# UPDATE vrsta_biljke SET naziv = 'Tratinčica' WHERE naziv = 'Ruža';
UPDATE 1
odrzavanje_biljaka=# SELECT * FROM vrsta_biljke;
 id |   naziv
----+------------
  2 | Tulipan
  3 | Suncokret
  1 | Tratinčica
(3 rows)

odrzavanje_biljaka=#
```

Slika 2: PotsgreSQL UPDATE upit **[autorski rad]**

Još jedan od zanimljivih pojmova je indeks, odnosno pojam indeksiranja. Indeks (engl.*index*) se odnosi na proces stvaranja posebne strukture podataka koja omogućuje brže pretraživanje podataka, sortiranje podataka te dohvaćanje podataka iz tablice. Indeksi povećavaju performanse upita tako da smanjuju potrebno vrijeme za traženje i dohvaćanje podataka. Glavna svrha indeksiranja je ubrzati upite, posebno onih koji se često koriste i koji rade nad velikim brojem podataka. Jedna od napomena je da su indeksi korisni nad poljima koja su 'UNIQUE' jer u suprotnom neće biti velike razlike u brzini upita. [6]

U PostgreSQL bazi podataka, indeks je posebna tablica koja sadrži redove s informacijama o vrijednostima određenih stupaca u glavnoj tablici. Postoje različite vrste indeksa, uključujući B-stablo, GIN (engl. *Generalized Inverted Index*), GIST (engl. *Generalized Search Tree*) i sl.

Moguće je stvoriti indeks nad jednim ili više atributa, a sintaksa stvaranja indeksa izgleda kao u primjeru stvarnja indeksa nad tablicom *zadaci*:

```
1 --Jednostavan indeks
2 CREATE INDEX idx_zadaci_id_biljke ON zadaci (id_biljke);

3 --Kompozitni indeks
4 CREATE INDEX idx_zadaci_kompozitni ON zadaci (id_biljke, EXTRACT(month FROM datum),
  ↪  EXTRACT(year FROM datum));
```

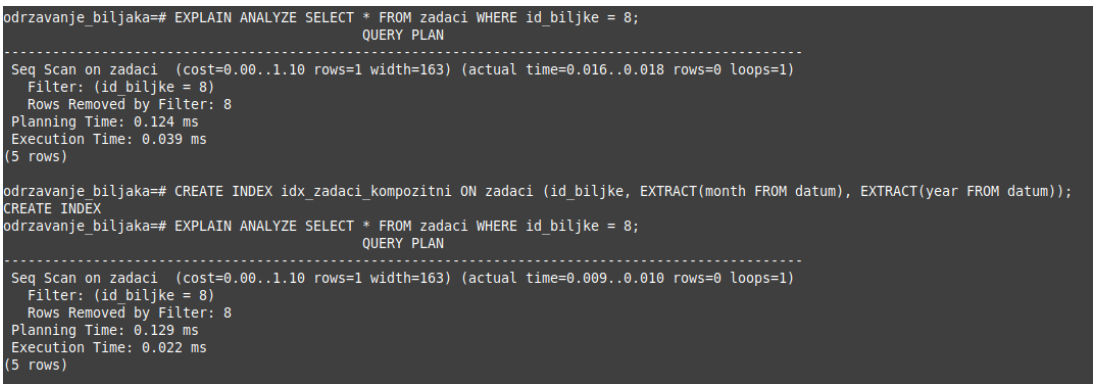
Isječak koda 11: Stvaranje indeksa

Brisanje indeksa jednostavno je učiniti naredbama:

```
1 --Brisanje jednostavnog i kompozitnog indeksa
2 DROP INDEX IF EXISTS idx_zadaci_id_biljke;
3 DROP INDEX IF EXISTS idx_zadaci_kompozitni;
```

Isječak koda 12: Brisanje indeksa

Brzina dohvata podataka bez i sa indeksima moze biti razlicita i trebalo bi je testirati nad tablicom sa više podataka te ih koristiti nad upitima koji se često pozivaju, no radi primjera na slici (Slika 3) prikazana su vremena izvršavanja istog upisa sa i bez indeksa. U ovom slučaju, sa indeksom vrijeme izvršavanja upita (*execution time*) se smanjio sa 0.039ms na 0.022ms.



Slika 3: Prikaz sa i bez korištenja indeksa **[autorski rad]**

Rad s temporalnim bazama podataka obuhvaća rad s vremenskim tipovima podataka te operacijama nad njima. Jedan od načina rada s vremenskim tipovima je pomoću određenih funkcija za rad s temporalnim podacima, a primjer jedne takve funkcije može se vidjeti u isječku koda 13.

```
app.get('/tasks', async (req, res) => {
    const month = parseInt(req.query.month) || new Date().getMonth();
    const year = parseInt(req.query.year) || new Date().getFullYear();
    const plantId = req.query.plantId;

    try {
        const tasksQuery = `
            SELECT id, naziv, opis, datum, izvrsen
            FROM zadaci
            WHERE EXTRACT(MONTH FROM datum) = $1
            AND EXTRACT(YEAR FROM datum) = $2
            AND id_biljke = $3
            ORDER BY id;
        `;

        const tasks = await pool.query(tasksQuery, [month, year, plantId]);
        const data = { allTasks: tasks.rows };

        res.json(data);
    } catch (error) {
        console.error('Greška prilikom dohvaćanja podataka iz baze:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});
```

Isječak koda 13: Popis svih zadataka

Na endpointu '/tasks' vrši se dohvaćanje svih zadataka iz baze podataka. Parametri zahtjeva koji se koriste su: mjesec - predstavlja mjesec za koji se traže zadaci, godina - predstavlja godinu za koju se traže zadaci i id biljke - predstavlja biljku za koju se traže zadaci.

Unutar prikazanog SQL upita dohvaćaju se podaci o zadatku koristeći *extract()* funkciju gdje se izdvaja mjesec i godina iz polja datum. Funkcija *extract()* upravo služi za dohvaćanje potpolja kao što su godina, mjesec, sat i sl. iz vrijednosti datuma ili vremena. [5] Vrijednost koju vraća *extract()* funkcija je tipa numeric, odnosno brojčana vrijednost. U primjeru, ukoliko je odabran datum: 10.01.2024., upit bi dohvatio podatke o zadacima gdje je vrijednost godine polja datum '2024' i vrijednost mjeseca polja datum '01', za traženu biljku.

4.3. Prikaz koda za klijentski dio Web aplikacije - JavaScript kod

Nakon što se struktura stranice definira pomoću HTML-a, oblikovanje se postiže korištenjem CSS-a, dok JavaScript omogućuje kreiranje interaktivnih elemenata u okviru Web aplikacije.

Funkcija koja je korištena u svakoj napisanoj JavaScript skripti je postavljanje slušača događaja, a jedan primjer je u slijedećem isječku koda:

```
document.addEventListener('DOMContentLoaded', function () {
    fetchPlantTypes();

    const typeForm = document.getElementById('typeForm');
    typeForm.addEventListener('submit', handleTypeFormSubmit);
    updateDropdownOptions();
});
```

Isječak koda 14: Postavljanje slušača događaja na učitavanje

Ovaj blok JavaScript koda koristi addEventListener funkciju kako bi čekao događaj DOMContentLoaded, što znači da će se izvršiti tek kada se cijeli HTML dokument učita. Ovaj pristup osigurava da se JavaScript kod izvrši tek nakon što su svi HTML elementi spremni za manipulaciju. U primjeru koda, nakon što se dokument učita pokreću se *fetchPlantTypes()*, *updateDropdownOptions()* funkcije te se postavlja slušać događaja na 'submit' događaj od forme s id-om 'typeForm'.

Još jedna od važnijih funkcija koja je korištena prilikom implementacije je *fetch* funkcija, a primjer jedne funkcije unutar koje je sadržana *fetch* funkcija je u idućem isječku koda:

```
const fetchTasks = async (month, year) => {
    try {
        const url = `http://localhost:3000/tasks?plantId=${plantId}&month=${month +
        ↪  1}&year=${year}`;
        const response = await fetch(url);
        const data = await response.json();

        return data;
    } catch (error) {
        console.error('Greška prilikom dohvaćanja podataka iz backend-a:', error);
        throw error;
    }
};
```

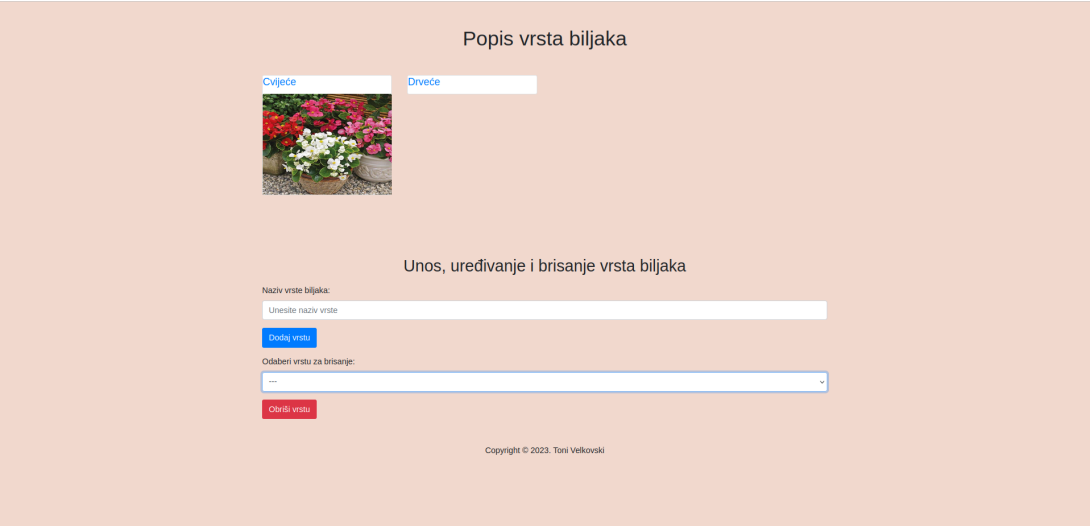
Isječak koda 15: Prikaz funkcije koja koristi *fetch* funkciju

Funkcija *fetchTasks()*, asinkrono dohvaća zadatke s poslužitelja koristeći fetch API, tj. *fetch* funkciju. Ona dohvaća podatke sa URL-a zadanog u varijabli 'url', koja u svojim parametrima ima mjesec i godinu po kojima se na poslužiteljskom dijelu Web aplikacije dohvaćaju zadaci.

5. Primjeri korištenja

Aplikacija je intuitivna i jednostavna za korištenje. Sastoji se od četiri stranice: početne stranice, stranice popisa biljaka odabrane vrste, stranice galerije slika odabrane biljke te popisa zadataka za odabranu biljku.

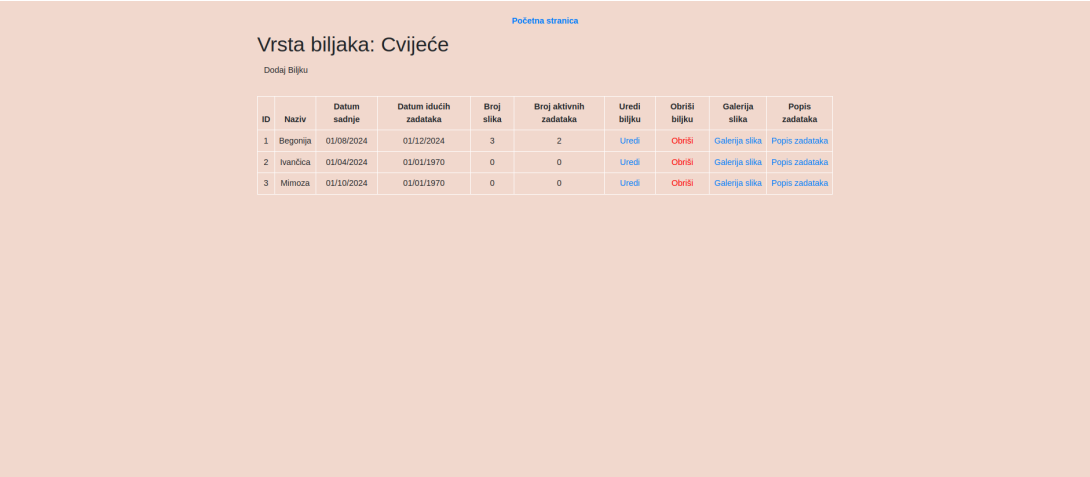
Početnu stranicu moguće je vidjeti na slici (Slika 4). Sastoji se od unosa i brisanja vrsti biljaka, a same vrste prikazane su kao kartice. Također, prilikom kreiranja vrste prikazan je samo njen naziv, dok nakon što se unose slike biljaka unutar vrste, tada se prikazuju i slike u kartici. Slike su prikazane na način da zadnje unesena slika za određenu biljku unutar te vrste je prikazana na početnoj stranici.



Slika 4: Početna stranica **[autorski rad]**

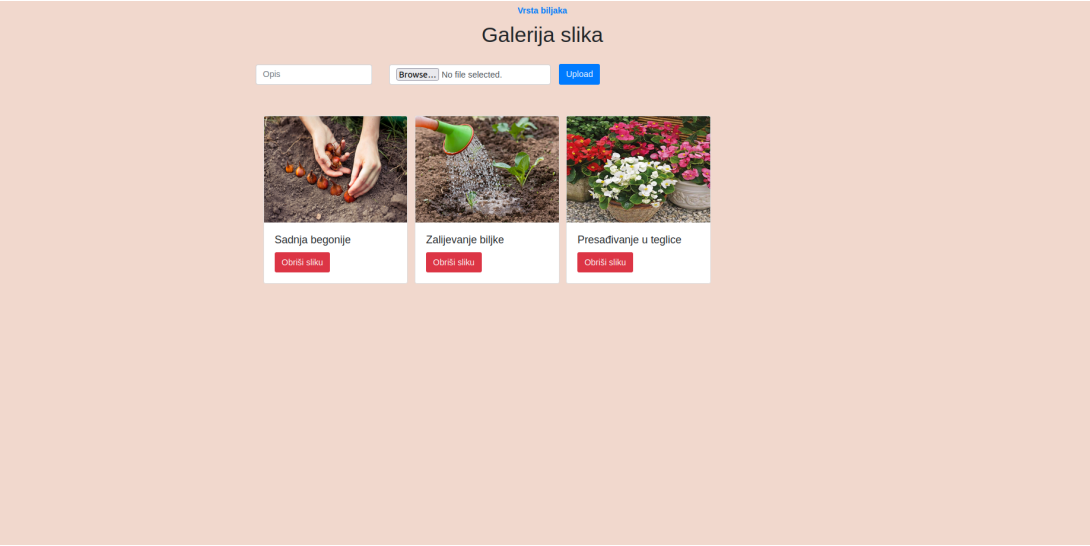
Stranica vrsta biljaka (Slika 5) sadrži popis svih biljaka odabrane vrste. Na stranici je moguće dodavati biljku, urediti određene atribute biljke i brisati biljku. Na klik za dodavanje ili uređivanje biljke otvara se forma unutar koje je moguće dodati naziv biljke i datum njene sadnje.

Također iz te stranice moguće je doći do stranica galerija slika i popis zadataka, koje se odnose na slike i zadatke te odabrane biljke. Također prikazan je i broj slika koje biljka ima, broj aktivnih zadataka (koji nisu izvršeni) te datum kada su idući zadaci na redu. To su oni okidači koji su spominjani u prethodnim poglavljima.



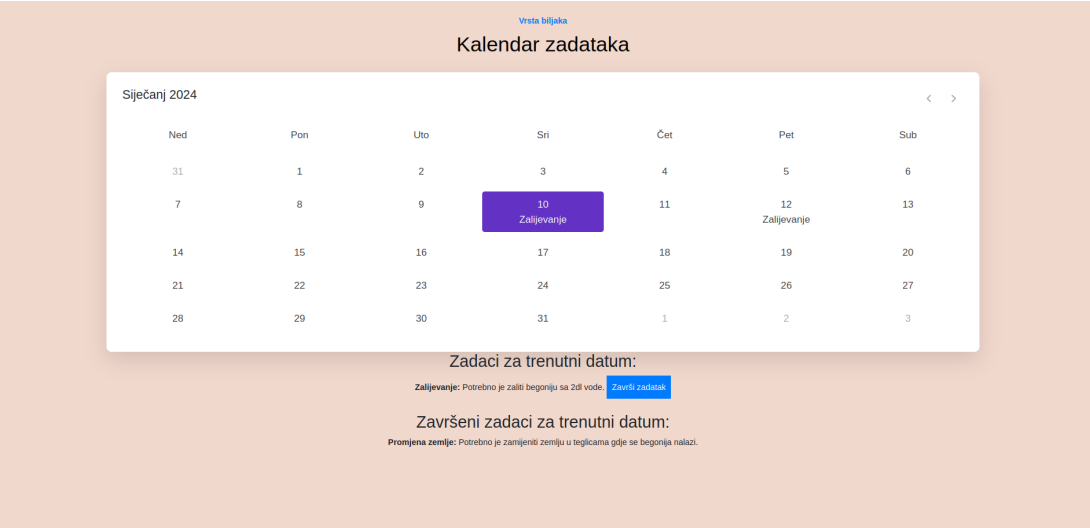
Slika 5: Popis svih biljaka odabrane vrste **[autorski rad]**

Stranica galerija slika (Slika 6) sastoji se od jednostavnog unosa, odnosno uploadanja slike biljke. Prilikom unosa slike, potrebno je staviti njen opis te je odabrati s računala i klikom na 'Upload' gumb slika i njen opis se prikazuju na stranici u obliku kartice. Sliku i opis je moguće jednostavno obrisati klikom na gumb 'Obriši sliku'.



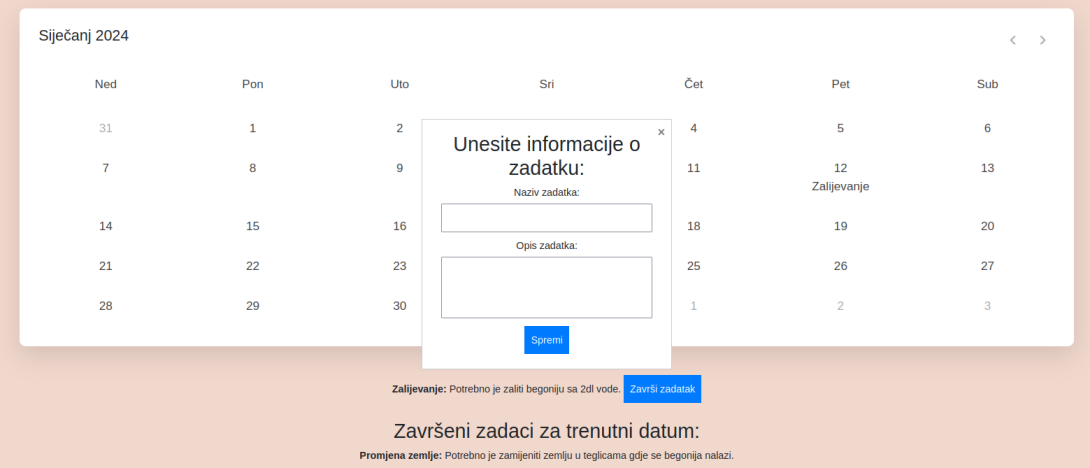
Slika 6: ERA model **[autorski rad]**

Stranica popis zadataka (Slika 7) sastoji se od jednostavnog kalendara unutar kojeg je moguće dodavati zadatke. Odabirom na određeni datum otvara se forma u kojoj je moguće unijeti podatke o zadatku. Na samom kalendaru prikazan je samo naziv zadatka, dok se ispod kalendara prikazuju zadaci za trenutni datum koji su neizvršeni ili izvršeni, a sam zadatak je moguće izvršiti klikom na gumb 'Završi zadatak'.



Slika 7: Popis zadataka odabrane biljke **[autorski rad]**

Na slici (Slika 8) moguće je vidjeti primjer forme kod unosa informacija o zadatku.



Slika 8: Unos informacija o zadatku **[autorski rad]**

6. Zaključak

U ovom projektu korištene su tehnologije poput Node.js za implementaciju poslužiteljskog dijela Web aplikacije, HTML i CSS za klijentski dio Web aplikacije te PostgreSQL kao baza podataka. Odabir tehnologija bazirao se na tome da je potrebno odabrati tehnologije koje će prikladno raditi sa PostgreSQL bazom podataka, a same odabrane tehnologije su jednostavne i dobro podržane za rad s PostgreSQL bazom podataka.

Node.js kao tehnologija za implementaciju poslužiteljskog dijela Web aplikacije je jednostavna za korištenje, brza i dobro povezana s PostgreSQL bazom podataka. Sama komunikacija temelji se na instalaciji *node-postgres*, odnosno *pg* modula te radu s istim. Ostatak implementacije temelji se na izgradnji jednostavnog API-ja koji komunicira sa kreiranom PostgreSQL bazom podataka.

Prilikom same implenentacije potrebno je paziti da se koriste koncepti aktivnih i temporalnih baza podataka. Koncepti aktivnih baza podataka realizirani su korištenjem raznih okidača, dok su temporalne baze podataka korištene primjerom vremenskih tipova i ugrađenih PostgreSQL funkcija koje rade s istim. Također, jedno od ograničenja je formatiranje datuma, gdje je potrebno prevesti datum koji se pohranjuje iz aplikacije u bazu podataka, budući da je moguće da su to datumi različitih tipova.

Pohranjivanje slika u bazu podataka te dohvat slika iz baze su također jedne od mogućih prepreka. Potrebno je uskladiti bazu podataka sa samim API-jem, da se slika ispravno pohranjuje, u ispravnom formatu.

Popis literature

[1] node-postgres. (), adresa: https://node-postgres.com/ (pogledano 9. 1. 2024.).

[2] M. Schatten, „Aktivne baze podataka - prezentacija s predavanja kolegija Teorija baza podataka," Markus Schatten 2023/2024.

[3] PostgreSQL. (), adresa: https://www.postgresql.org/docs/current/sql-createtrigger.html (pogledano 10. 1. 2024.).

[4] M. Schatten, „Temporalne baze podataka - prezentacija s predavanja kolegija Teorija baza podataka," Markus Schatten 2023/2024.

[5] PostgreSQL. (), adresa: https://www.postgresql.org/docs/current/datatype-datetime.html (pogledano 10. 1. 2024.).

[6] PostgreSQL. (), adresa: https://www.postgresql.org/docs/current/indexes.html (pogledano 10. 1. 2024.).

Popis slika

Popis isječaka koda

Prilozi

1. Prilog 1 - poveznice

1. GitHub projekt:

`https://github.com/tonivelkovski/application_for_plant_monitoring`

2. Overleaf projekt:

`https://www.overleaf.com/read/pxrykcktfdpz#f0ee42`

3. Upute za pokretanje:

`https://github.com/tonivelkovski/application_for_plant_monitoring/blob/main/README.md`