Web servis za skladištenje podataka dobivenih pomoću MQTT-a

Brezić, Ivan

VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

WEB SERVIS ZA SKLADIŠTENJE PODATAKA DOBIVENIH POMOĆU MQTT-a

Završni rad br. 06/RAČ/2020

Ivan Brezić

Bjelovar, listopad 2020.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Kandidat: **Brezić Ivan** Datum: 31.08.2020. Matični broj: 001844

JMBAG: 0246065579

Kolegij: **INTERNET STVARI**

Naslov rada (tema): **Web servis za skladištenje podataka dobivenih pomoću MQTT-a**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Programsko inženjerstvo**

Mentor: **Krunoslav Husak, dipl.ing.rač.** zvanje: **predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Tomislav Adamović, mag.ing.el., predsjednik**
2. **Krunoslav Husak, dipl.ing.rač., mentor**
3. **dr.sc. Zoran Vrhovski, član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 06/RAČ/2020

U radu je potrebno izraditi Web servis za unos podataka s MQTT-a u bazu podataka, te dohvaćanje istih kroz REST API i njihov prikaz u Web aplikaciji.

Korištenje Go programskog jezika (Golang) za izradu Web servisa i REST API-a, PostgreSQL-a za izradu baze podataka i izvršavanje operacija nad podacima (CRUD naredbe), te korištenje React JavaScript bibiloteke za izradu Web aplikacije, odnosno prikaz podataka iz baze podataka.

Cilj rada je izraditi sustav koji omogućava dohvaćanje podataka s IoT uređaja preko MQTT protokola, te pohranjivanje podataka u bazu podataka. Web servis mora omogućiti njihovo dohvaćanje kroz REST API, kako bi se podaci mogli prikazati korisniku kroz Web aplikaciju.

Zadatak uručen: 31.08.2020. Mentor: **Krunoslav Husak, dipl.ing.rač.**

Zahvala

Zahvaljujem se kolegama Dominiku Ernjaku i Robertu Topaloviću te mentoru Krunoslavu Husaku, dipl.ing.rač. na iskazanoj pomoći i izdvojenom vremenu za izradu ovog završnog rada.

Sadržaj

1. UVOD 1

2. ARHITEKTURA IOT WEATHER STATION SUSTAVA 2

2.1 Dijelovi sustava 2
2.1.1 Hardver 2
2.1.2 MQTT 3
2.1.3 Baza podataka 3
2.1.4 Aplikacijsko programsko sučelje i web servis 3
2.1.5 Web aplikacija 3

3. MQTT 4

3.1 Općenito 4

3.2 MQTT publish/subscribe arhitektura 4

3.3 Implementacija u IoT Weather Station sustavu 5

4. BAZA PODATAKA 6

4.1 PostgreSQL 6
4.1.1 Općenito 6
4.1.2 Podrška 6

4.2 Implementacija u IoT Weather Station sustavu 7
4.2.1 Aplikacijsko programsko sučelje 7
4.2.2 Web servis 11
4.2.3 Model baze podataka 12

5. PROGRAMSKI JEZIK GO 14

5.1 Općenito 14

5.2 Implementacija u IoT Weather Station sustavu 15

6. IOT WEATHER STATION API 16

6.1 API 16
6.1.1 REST API 16
6.1.2 API dokumentacija 18

6.2 Funkcionalnosti IoT Weather Station API-a 19
6.2.1 HTTP poslužitelj 20
6.2.2 API metode 22
6.2.3 WebSocket 23

7. IOT WEATHER STATION SERVICE 25

7.1 Web servis 25

7.2 Mikroservis 25

7.3 Funkcionalnosti IoT Weather Station Service-a 26
7.3.1 Servis 26
7.3.2 Funkcije 29

8. Weather app 33

8.1 React 33

8.2 Grafičko korisničko sučelje 33

8.3 Učitavanje web aplikacije 33

8.4 Početna stranica *34*
8.5 Stanice *35*
8.6 Senzori *36*
8.7 Mjerne jedinice *36*
8.8 O nama *37*
8.9 Dodatne funkcionalnosti *37*
9. ZAKLJUČAK **38**
10. LITERATURA **39**
11. OZNAKE I KRATICE **41**
12. SAŽETAK **42**
13. ABSTRACT **43**

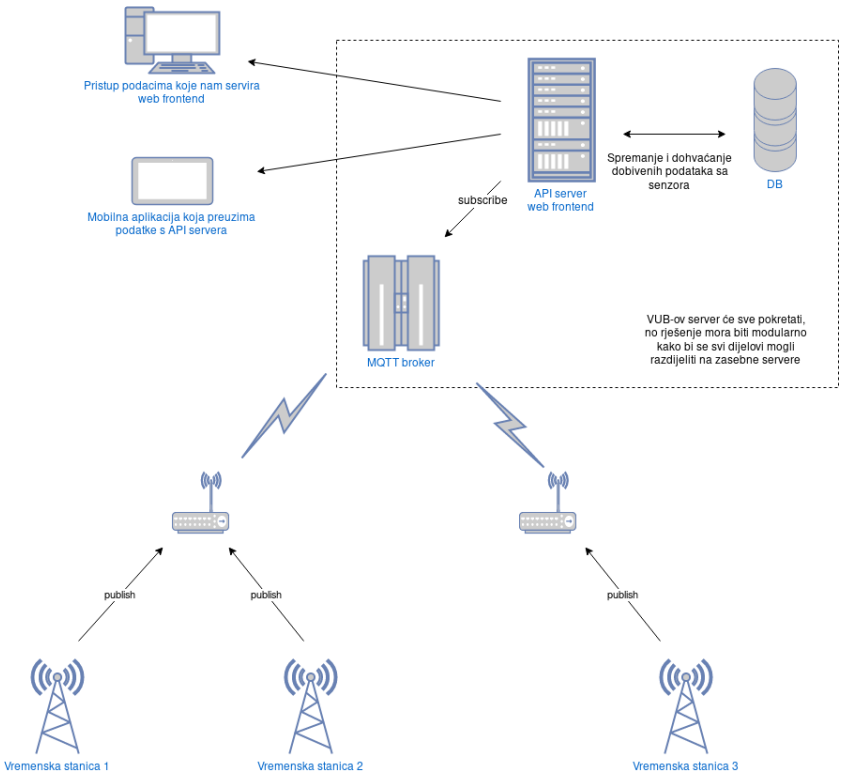
1. UVOD

Tema ovog završnog rada je skladištenje podataka dobivenih s MQTT posrednika u bazu podataka. Rad će pobliže opisati dijelove i programska rješenja *IoT Weather Station* sustava. Funkcija *IoT Weather Station* sustava je prikupljanje podataka sa senzora vremenskih stanica, objavljivanje podataka na MQTT posrednik, prikupljanje podataka s MQTT posrednika, pohranjivanje, odnosno skladištenje podataka u bazu podataka i prikaz podataka kroz grafičko korisničko sučelje u obliku web aplikacije. Glavni dijelovi sustava su hardver, MQTT posrednik, baza podataka, aplikacijsko programsko sučelje, web servis i web aplikacija. Rezultat završnog rada je funkcionalan IoT sustav za praćenje vremenski parametara kao što su temperatura, vlaga, plinovi, tlak zraka itd.

U poglavlju dva detaljnije su opisani dijelovi i funkcije *IoT Weather Station* sustava, dok su MQTT protokol i njegova primjena u sustavu opisani u poglavlju tri. Poglavlje četiri opisuje bazu podataka i njenu implementaciju u sustavu. U poglavlju pet opisan je programski jezik Go pomoću kojega su razvijeni aplikacijsko programsko sučelje za dohvat podataka iz baze podataka i web servis za unos podataka u bazu podataka, što je opisano u poglavlju šest: *IoT Weather Station API* i poglavlju sedam: *IoT Weather Station Service*. Osmo poglavlje opisuje grafičko korisničko sučelje realizirano kroz *Weather App* web aplikaciju, a prijedlozi daljnjeg unaprjeđenja dani su u poglavlju Zaključak.

2. ARHITEKTURA IOT WEATHER STATION SUSTAVA

IoT Weather Station sustav je sustav koji je razvijen u svrhu završnog rada, a sastoji se od sljedećih dijelova: baze podataka, aplikacijskog programskog sučelja, web servisa i web aplikacije. Ostali dijelovi sustava kao što su hardver, MQTT posrednik i Android mobilna aplikacija, razvijeni su i konfigurirani od strane kolega i profesora s Veleučilišta u Bjelovaru te će biti spomenuti kroz rad. Dijelovi *IoT Weather Station* sustava prikazani su na slici 2.1.



Slika 2.1: Shematski prikaz IoT Weather Station sustava

2.1 Dijelovi sustava

U sljedećim odlomcima ukratko su opisani dijelovi *IoT Weather Station* sustava i korišteni programski jezici. Za razvoj programskog dijela *IoT Weather Station* sustava korišten je program za uređivanje koda Visual Studio Code.

2.1.1 **Hardver**

Hardver, razvijen od strane kolege s Veleučilišta, razvijen je kroz ESP32 mikroupravljač (slika 2.2 a)) na koji su implementirana dva senzora: DHT22 (slika 2.2 b)) i

BME680 (slika 2.2 c)). Njegova svrha je prikupljanje parametara poput temperature, vlage, plinova, tlaka zraka, itd. Mikroupravljač je programiran u C++ programskom jeziku.

a) ESP32 mikroupravljač

b) DHT22 senzor

c) BME680 senzor

Slika 2.2: Hardver IoT Weather Station sustava

2.1.2 **MQTT**

MQTT posrednik (engl. *broker*), konfiguriran i objavljen na poslužitelj Veleučilišta od strane mentora, a koristi se za komunikaciju hardvera s web servisom kako bi se podatci sa senzora pohranili u bazu podataka.

2.1.3 **Baza podataka**

Baza podataka implementirana je PostgreSQL bazom podataka (poglavlje 3.1), a njena svrha je pohranjivanje podataka dohvaćenih s MQTT-a putem web servisa te dohvaćanje istih podataka putem aplikacijskog programskog sučelja.

2.1.4 **Aplikacijsko programsko sučelje i web servis**

Aplikacijsko programsko sučelje (*IoT Weather Station API*) i web servis (*IoT Weather Station Service*) razvijeni su u programskom jeziku Go (poglavlje 5). Aplikacijsko programsko sučelje koristi se za dohvaćanje podataka iz baze podataka, a web servis se koristi za dohvaćanje podataka s MQTT-a i njihovo pohranjivanje u bazu podataka.

2.1.5 **Web aplikacija**

Web aplikacija (*Weather App*), razvijena pomoću JavaScript biblioteke React, prikazuje podatke dohvaćane s aplikacijskog programskog sučelja.

3. MQTT

MQTT je OASIS standardni protokol prilagođen za razmjenu poruka IoT klijenata (npr. IoT uređaji, programi, itd.) [1]. Temelji se na *publish*/*subscribe* principu razmjene poruka koji je idealan za povezivanje uređaja s malim otiskom koda (engl. *Code footprint*) i minimalnom mrežnom propusnosti (engl. *Network bandwidth*) [2]. MQTT se danas koristi u raznim industrijama, poput automobilske, prerađivačke telekomunikacijske, naftne, plinske, itd [4].

3.1 Općenito

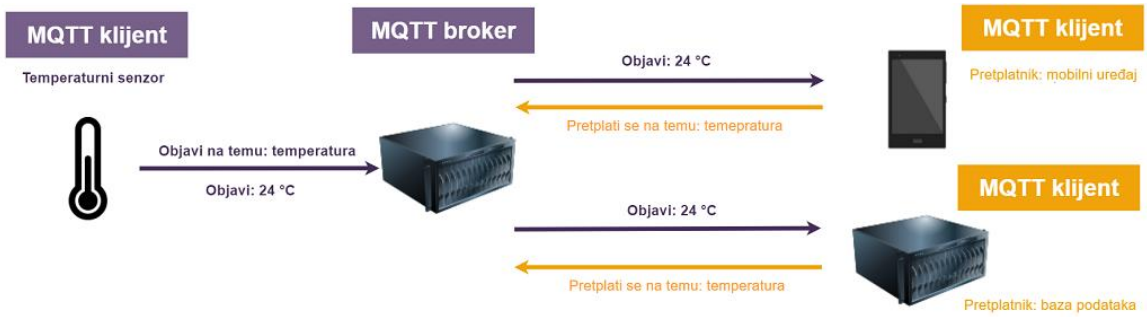
Prva verzija MQTT protokola korištena je za nadzor naftovoda u pustinji, a cilj je bio kreiranje protokola koji je lagan (engl. *lightweight*), troši malo energije i postiže minimalnu mrežnu propusnost (engl. *network bandwidth*). Njegove prednosti navedene su u odlomku ispod [3].

MQTT klijenti zahtijevaju malo resursa te se stoga mogu koristiti na mikroupravljačima poput ESP32 mikroupravljača korištenog u *IoT Weather Station* sustavu. Zaglavlja MQTT poruka su mala kako bi se optimizirala mrežna propusnost. MQTT podržava dvosmjernu komunikaciju (klijent-poslužitelj i poslužitelj-klijent), što omogućava emitiranje (engl. *Broadcasting*) poruka grupama klijenata. MQTT je skalabilan, odnosno može se prilagoditi povezivanju s milijunima IoT uređaja. MQTT ima 3 definirane razine kvalitete isporuke poruka: 0 - najviše jednom, 1 - barem jednom i 2 - točno jednom. Ako se koristi nepouzdana mreža, MQTT smanjuje vrijeme za ponovno povezivanje klijenta s posrednikom (engl. *broker*). MQTT olakšava šifriranje poruka pomoću TLS-a i omogućava provjeru autentičnosti klijenata pomoću modernih protokola za provjeru autentičnosti, kao što je OAuth [4].

3.2 MQTT publish/subscribe arhitektura

MQTT protokol temelji se na *publish*/*subscribe* arhitekturi razmjene poruka, a to je arhitektura u kojoj pošiljatelj (engl. *publisher*) ne šalje poruke direktno prema primateljima (engl. s*ubscriber*), nego ih objavljuje u kategorizirane teme (engl. *topic*). Primatelj, odnosno pretplatnik, pretplaćuje se na temu koja je u njegovom interesu kako bi primio

poruke [5]. Primjer MQTT *publish*/*subscribe* arhitekture sa senzorom temperature prikazan je na slici 3.1.



Slika 3.1: MQTT publish/subscribe arhitektura

3.3 Implementacija u IoT Weather Station sustavu

IoT Weather Station sustav implementira MQTT protokol kroz Eclipse Mosquitto MQTT posrednik, a njegova svrha je kategoriziranje objavljenih poruka u teme i slanje poruka pretplaćenim korisnicima [6].

MQTT posrednik koriste dva dijela *IoT Weather Station* sustava:

- hardver
    - ESP32 mikroupravljač objavljuje konfiguraciju vremenske stanice i vrijednosti učitane sa senzora u dvije teme: *ws/{stationId}/config* i *ws/{stationId}/data*
- *IoT Weather Station Service*
    - servis je pretplaćen na obje teme i prilikom objave nove poruke na MQTT posrednik poziva funkcije za provjeru i unos konfiguracije i vrijednosti vremenske stanice u bazu podataka

4. BAZA PODATAKA

Baza podataka je temeljni dio svake web aplikacije i web sustava te je tako i jedan od glavnih dijelova *IoT Weather Station* sustava. U nastavku će biti opisana baza podataka koju koristimo i način na koji je baza podataka implementirana u sustav, a koristi se za pohranjivanje, odnosno skladištenje podataka prikupljenih s MQTT posrednika. Baza podataka *IoT Weather Station* sustava implementirana je pomoću PostgreSQL baze podataka.

4.1 PostgreSQL

PostgreSQL je samoprozvana najnaprednija *open source* baza podataka na svijetu [7], a njene mogućnosti mogu se usporediti s plaćenim verzijama poput Oracle-a [8]. Koriste ju tvrtke poput Uber-a, Netflix-a, Instagram-a, Reddit-a, Twitch-a, itd. [9] U usporedbi s ostalim *open source* bazama podataka (npr. MySQL, SQLite, itd.), PostgreSQL pruža podršku za objektno-orijentirano programiranje baza podataka, dok ostale baze podataka podržavaju samo relacijski model [10].

4.1.1 **Općenito**

PostgreSQL temeljni se na objektno-relacijskom modelu baza podataka, temeljni model je relacijski koji organizira podatke u tablice koje sadrže retke i stupce. Relacijski model je temeljni model današnjih baza podataka, no većina modernih baza pruža podršku za objektno-relacijski model, odnosno objektno-orijentirane modele baze podataka koji sadrže objekte, klase i nasljeđivanje, a dostupni su u samim shemama baze podataka i kroz proširenje SQL programskog jezika [11].

4.1.2 **Podrška**

PostgreSQL je podržan na svim većim operacijskim sustavima (npr. Windows, Linux, Mac OS X, itd.), a neke od njegovih značajki su: razni tipovi podataka, očuvanje integriteta podataka, konkurentnost, performanse i sigurnost. A najveća značajka mu je proširivost koja omogućuje korištenje velikog broja proceduralnih jezika poput: Perl-a, Python-a, C-a, PL/pgSQL-a (slično PL/SQL-u), Java-e, JavaScript-a, R-a, Ruby-a, itd [12].

4.2 Implementacija u IoT Weather Station sustavu

PostgreSQL naredbe nalaze se unutar Go programskog koda, odnosno unutar aplikacijskog programskog sučelja i web servisa, a izvršavaju se pomoću *database/sql* i *github.com/lib/pq* Go paketa.

U nastavku je opisana implementacija na dijelovima sustava koji koriste PostgreSQL naredbe, a koriste se sljedeći tipovi SQL naredbi: DDL (*create*, *drop*, *alter*), DML (*insert*, *update*, *delete*), TCL (*commit*, *rollback*) i DQL (*select*).

4.2.1 **Aplikacijsko programsko sučelje**

Aplikacijsko programsko sučelje naziva *IoT Weather Station API* (poglavlje 6) koristi PostgreSQL naredbe za manipulaciju tablicama (npr. *create*, *drop*, *alter*) i kreiranje potrebnih atributa tablica (npr. *index*, *constraint*, *primary key*, *foreign key*) potrebnih za rad sustava. Svrha baze podataka u aplikacijskom programskom sučelju je dohvaćanje podataka pomoću PostgreSQL upita, a za manipulaciju podataka podržana je *get* API metoda, odnosno *select* PostgreSQL naredba jer je uloga aplikacijskog programskog sučelja dohvaćanje podataka pohranjenih u bazu podataka.

Za potrebe aplikacijskog programskog sučelja kreirani su Go paketi koji odrađuju povezivanje s bazom podataka te izvršavaju PostgreSQL upite:

- *config*
    - dohvaća konfiguraciju API-a zapisanu u *configuration.toml* konfiguracijskoj datoteci (programski kod 4.1 a)) te kreira *Database* objekt koji sadrži parametre potrebne za povezivanje na bazu podataka (programski kod 4.1 b))

Programski kod 4.1: Konfiguracijski parametri IoT Weather Station API-a

a) configuration.toml konfiguracijska datoteka

configuration.toml

```
[database]
host = "127.0.0.1"
port = "5432"
user = "iot"
password = "IoTWeatherStation4"
name = "iot_weather_station"

[server]
address = "127.0.0.1:8001"

[log]
filename = "iot-weather-station-output-api.log"

[router]
release = false
```

b) Database objekt

```
// Database defines DB configuration
type Database struct {
    Host     string
    Port     string
    User     string
    Password string
    Name     string
}
```

- *db*
  - sadrži funkciju za povezivanje na bazu podataka (programski kod 4.4), funkciju za kreiranje tablica s potrebnim atributima (programski kod 4.5) i konstante s PostgreSQL naredbama za kreiranje tablica i atributa (programski kod 4.2)

Programski kod 4.2: Konstante db Go paketa

```
// stations SQL
const sqlCreateStations = `
    CREATE TABLE IF NOT EXISTS stations (
        id int PRIMARY KEY,
        name text NOT NULL,
        sensors text[] NOT NULL,
        latitude float NOT NULL,
        longitude float NOT NULL,
        created_at timestamp DEFAULT now() NOT NULL,
        updated_at timestamp DEFAULT now() NOT NULL,
        UNIQUE(name)
    )`

const sqlCreateStationsIDIndex = `
CREATE INDEX IF NOT EXISTS stations_id_idx ON stations(id)
`

const sqlCreateStationsNameIndex = `
CREATE INDEX IF NOT EXISTS stations_name_idx ON stations(name)
`

const sqlCreateStationsSensorsIndex = `
CREATE INDEX IF NOT EXISTS stations_sensors_idx ON stations(sensors)
`
```

- *consts*
    - sadrži konstante s PostgreSQL naredbama za unos i dohvaćanje podataka (programski kod 4.3)

Programski kod 4.3: Konstante consts paketa

```
// SQLInsertStation insert station
const SQLInsertStation = `
INSERT INTO stations (
    id, name, sensors, latitude, longitude
) VALUES (
    $1, $2, $3, $4, $5
)
`

// SQLSelectAllStations select all stations
const SQLSelectAllStations = `
SELECT
    *
FROM
    stations
ORDER BY
    updated_at desc
LIMIT
    1000
`
```

- *models*
    - sadrži funkcije koje izvršavaju PostgreSQL naredbe (programski kod 4.6) zapisane u *consts* paketu

Aplikacijsko programsko sučelje pri svakome pokretanju ostvaruje novu konekciju sa bazom podataka (programski kod 4.4) i provjerava postojanje tablica i atributa u bazi podataka neophodnih za rad sustava (programski kod 4.5).

Programski kod 4.4: Funkcija za povezivanje na bazu podataka

```go
// Connect creates new Database struct and connects to it
func Connect(db config.Database) (*Database, error) {
    connStr := fmt.Sprintf("host=%s port=%s user=%s password=%s dbname=%s sslmode=disable", db.Host, db.Port, db.User, db.Password, db.Name)
    conn, err := sql.Open("postgres", connStr);
    if err != nil {
        log.Fatal(err)
    }

    err = conn.Ping()
    if err != nil {
        log.Fatal(err)
    }

    return &Database{Conn: conn}, nil
}
```

Programski kod 4.5: Funkcija za kreiranje tablica i atributa

```go
// Migrate migrates database to valid state
func (db *Database) Migrate() (err error) {
    tx, err := db.Conn.Begin()
    if err != nil {
        return errors.New("Unable to start transaction: " + err.Error())
    }

    defer func() {
        if err != nil {
            tx.Rollback()
            err = errors.New("There was an error in migrate transaction: " + err.Error())
        }
        err = tx.Commit()
        if err != nil {
            err = errors.New("There was an error in commiting migrate transaction: " + err.Error())
        }
    }()

    // ########
    // Stations
    _, err = tx.Exec(sqlCreateStations)
    if err != nil {
        return
    }

    _, err = tx.Exec(sqlCreateStationsIDIndex)
    if err != nil {
        return
    }

    _, err = tx.Exec(sqlCreateStationsNameIndex)
    if err != nil {
        return
    }

    _, err = tx.Exec(sqlCreateStationsSensorsIndex)
    if err != nil {
        return
    }
```

Nakon ostvarene konekcije na bazu podataka i provjere postojanja tablica pokreće se HTTP poslužitelj preko kojega je dostupno aplikacijsko programsko sučelje, odnosno API metode (engl. e*ndpoints*). U API metodama pozivaju se funkcije iz *models* Go paketa koje izvršavaju PostgreSQL naredbe za dohvaćanje podataka iz baze podataka (programski kod 4.6).

Programski kod 4.6: Funkcija za dohvaćanje podataka iz baze podataka

```go
func getStation(db *db.Database, query string, param string) (*Station, error) {
    s := &Station{}
    var sensorIDs []string

    err := db.Conn.QueryRow(query, param).Scan(&s.ID, &s.Name, pq.Array(&sensorIDs), &s.Latitude, &s.Longitude, &s.CreatedAt, &s.UpdatedAt)
    switch {
    case err == sql.ErrNoRows:
        return nil, sql.ErrNoRows
    case err != nil:
        return nil, errors.New("Error while getting station (" + err.Error() + ")")
    }

    s.Sensors, err = GetSensorsByIDArray(db, sensorIDs)
    if err != nil {
        return nil, errors.New("Error while getting sensors (" + err.Error() + ")")
    }

    return s, nil
}
```

4.2.2 **Web servis**

Web servis naziva *IoT Weather Station Service* (poglavlje 7) koristi bazu podataka u svrhu dohvaćanja i unosa podataka pomoću PostgreSQL upita, odnosno *select* SQL naredbu za provjeru podataka prikupljenih s MQTT brokera, SQL *insert* naredbu za unos podataka prikupljenih s MQTT brokera te *commit* i *rollback* SQL naredbe za pohranjivanje, odnosno poništavanje transakcija (*insert*, *delete* i *update*) na bazi podataka.

Web servis koristi bazu podataka u sličnu svrhu kao i aplikacijsko programsko sučelje te bi se pretpostavilo da i aplikacijsko programsko sučelje i web servis koriste iste Go pakete za izvršavanje operacija nad bazom podataka, no zbog modularnosti *IoT Weather Station* sustava, koriste se kopirani paketi razvijeni za aplikacijsko programsko sučelje, uz male promijene potrebne za unos podataka i povezivanje sa MQTT posrednikom.

Za potrebe web servisa kreirani su Go paketi koji odrađuju povezivanje s bazom podataka te izvršavaju PostgreSQL naredbe:

- *config*
    - dohvaća konfiguraciju web servisa zapisanu u *configuration.toml* konfiguracijskoj datoteci (programski kod 4.1 a)) te kreira *Database* objekt koji sadrži parametre potrebne za povezivanje na bazu podataka (programski kod 4.1 b))
- *db*
    - sadrži funkciju za povezivanje na bazu podataka (programski kod 4.4)
- *consts*

- sadrži konstante s PostgreSQL naredbama za unos i dohvaćanje podataka (programski kod 4.3)

- *models*
  - sadrži funkcije koje izvršavaju PostgreSQL naredbe (programski kod 4.6 i 4.7) zapisane u *consts* paketu

Web servis pri svakome pokretanju ostvaruje novu konekciju sa bazom podataka (programski kod 4.4), a nakon ostvarene konekcije na bazu podataka, pokreće se robot iz *gobot.io/x/gobot* Go paketa koji sadrži funkcije za prihvaćanje podataka s MQTT-a. Nakon što je podataka stigao s MQTT-a na web servis, podatci se provjeravaju i unose u bazu podataka (programski kod 4.7).

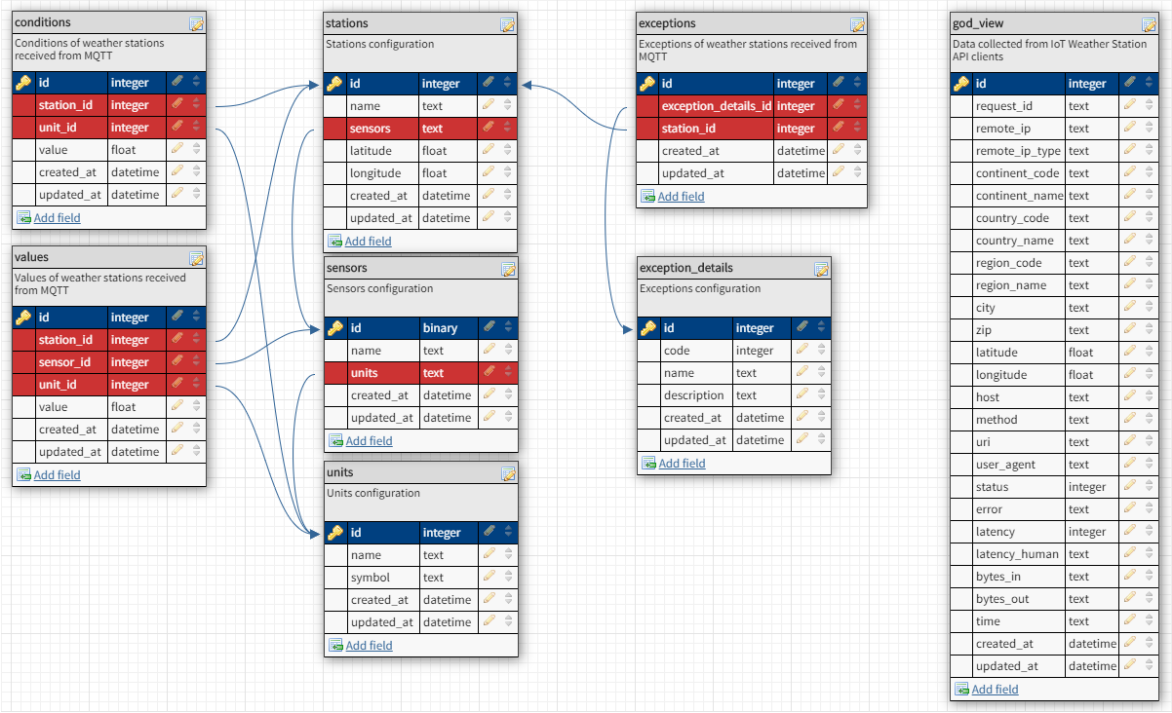
Programski kod 4.7: Funkcija za unos podataka u bazu podataka

```
func (mc *MqttController) handleConfiguration(msg mqtt.Message) {
    mc.Logger.Info("handleConfiguration (msg: " + string(msg.Payload()) + ") topic: " + msg.Topic())
    var config models.Configuration

    strJSON := string(msg.Payload())
    err := json.Unmarshal([]byte(strJSON), &config)
    if err != nil {
        mc.Logger.Error(err.Error())
    }

    result, err := models.InsertConfiguration(mc.Database, &config)
    if err != nil {
        mc.Logger.Error(err.Error())
        err := SendEmailAlert("Configuration exception", "handleConfiguration (msg: " + string(msg.Payload()) + ") topic: " + msg.Topic(), err.Error())
        if err != nil {
            mc.Logger.Error(err.Error())
        }
    } else if result != nil {
        mc.Logger.Info("Station successfully inserted.")
    }

}
```

4.2.3 **Model baze podataka**

Za izradu modela baze podataka *IoT Weather Station* sustava korištena je web aplikacija DB Designer koja omogućava jednostavno kreiranje tablica, redaka i atributa tablica (npr. *primary key*, *foreign key*, itd.) kroz sučelje te izvoženje (engl. *export*) modela u raznim izvedenicama SQL jezika (MySQL, MS SQL Server, SQLite, PostgreSQL, i Oracle).

IoT Weather Station sustav koristi osam tablica, a njihovi stupci, atributi i relacije prikazani su na slici 4.1.



Slika 4.1: Model baze podataka

5. PROGRAMSKI JEZIK GO

Programski jezik Go je relativno novi, statički pisani (engl. *statically typed*) i kompajliran programski jezik koji djeluje kao dinamički pisani (engl. d*ynamically typed*), interpreterski jezik (engl. *interpreted language*), a razvijen je u Google-u. Iako je dizajniran s namjerom za korištenje u sistemskom programiranju, Go je programski jezik opće namjene. Poput Java-e, C#-a ili Python-a, Go sadrži mehanizam za sakupljanje smeća (engl. g*arbage collection*) koji je ugrađen u sustav i automatski se pokreće po potrebi, a brine se za oslobađanje memorije koja više nije potrebna. Uz mehanizam sakupljanja smeća, Go sadrži i mehanizam za konkurentnost koji omogućava maksimalno iskorištavanje višejezgrenih i mrežno povezanih računala [13].

5.1 Općenito

Programski jezik Go razvijen je pod utjecajem programskog jezika C, ali s naglaskom na jednostavnost i sigurnost. Sintaksa i okolina programskog jezika Go usvaja mogućnosti koji su uobičajene za dinamičke jezike (engl. d*ynamic language*), a mogućnosti su sljedeće [14]:

- brzo kompajliranje
- udaljeno upravljanje paketima (engl. r*emote package management*) pomoću naredbe *go get* i dostupna dokumentacija paketa
- sažeta deklaracija i inicijalizacija varijabli kroz zaključivanje tipa varijabli (engl. t*ype inference)*
    - *primjer*: x := 0 umjesto int x = 0; ili var x = 0;

Uz gore navede mogućnosti, programski jezik Go također sadrži i prepoznatljive pristupe određenim problemima:

- ugrađena primitivna paralelnost:
    - *goroutines* – lagani procesi (engl. l*ight-weight processes*)
    - kanali (engl. *channels*)
    - naredba *select*
- sustav sučelja (engl. *interface*) umjesto virtualnog nasljeđivanja i ugrađivanje tipova (engl. *type embedding*) umjesto ne virtualnog nasljeđivanja

- alati koji prema zadanim postavkama stvaraju statički povezane izvorne binarne datoteke bez vanjskih ovisnosti

Cilj programskog jezika Go je postizanje programerima lako pamtljivog programskog jezika, dijelom kroz sintaksu i dijelom kroz izostavljanje značajki koje su uobičajene u sličnim programskim jezicima [15].

5.2 Implementacija u IoT Weather Station sustavu

Tema završnog rada vezana je uz kolegij Internet stvari koji nam je dao uvid u mogućnosti i prednosti programskog jezika Go. Upravo iz tog razloga programski jezik Go odabran je za razvoj pozadinskog (engl. *backend*) dijela *IoT Weather Station* sustava, a to su: baza podataka (poglavlje 4), *IoT Weather Station API* (poglavlje 6) i *IoT Weather Station Service* (poglavlje 7).

IoT Weather Station Service, odnosno web servis implementira povezivanje na *WebSocket* aplikacijskog programskog sučelja, povezivanje na bazu podataka i pokretanje "robota" s funkcijama dohvata i provjere podataka s MQTT posrednika te pohrane istih u bazu podataka i slanje podataka putem *WebSocket-a*. Web servis dodatno sadrži funkcionalnost za prepoznavanje grešaka i iznimaka te obavještavanje razvojnih programera putem e-mail poslužitelja.

IoT Weather Station API, odnosno aplikacijsko programsko sučelje implementira povezivanje na bazu podataka i pokretanje HTTP poslužitelja koji sadrži metode s funkcijama pristupa podatcima u bazi podataka prikupljenih pomoću web servisa. Aplikacijsko programsko sučelje podržava *WebSocket* za razmjenu podataka između web servisa i grafičkog korisničkog sučelja u stvarnome vremenu (engl. *real-time*). Kao i web servis, dodatno sadrži funkcionalnost korištenja e-mail poslužitelja, ali za komunikaciju s korisnicima *IoT Weather Station* sustava.

Sve funkcionalnosti baze podataka, web servisa i aplikacijskog programskog sučelja ostvarene su upotrebom programskog jezika Go i pripadajućih paketa od kojih su najznačajniji: *github.com/labstack/echo, github.com/gorilla/websocket* i *gobot.io/x/gobot*.

6. IOT WEATHER STATION API

IoT Weather Station API je aplikacijsko programsko sučelje *IoT Weather Station* sustava. Njegova svrha je dohvaćanje podataka o vremenskim stanicama iz baze podataka pomoću HTTP poslužitelja na kojemu je dostupno aplikacijsko programsko sučelje koje pruža metode za dohvat i provjeru podataka.

U ovome poglavlju detaljno su opisane funkcionalnosti i mogućnosti *IoT Weather Station* aplikacijskog programskog sučelja.

6.1 API

API, odnosno aplikacijsko programsko sučelje je računalno sučelje koje definira interakcije između više softverskih posrednika. Aplikacijsko programsko sučelje definira vrste zahtjeva koji se mogu uputiti, kako ih uputiti, formate podataka koje treba koristiti, standarde koje treba primijeniti, itd. Također može pružiti mehanizme proširenja tako da korisnici mogu proširiti postojeće funkcionalnosti. Aplikacijsko programsko sučelje može biti prilagođeno za specifičnu komponentu sustava (aplikacije) ili može biti dizajnirano na temelju industrijskog standarda kako bi se osigurala interoperabilnost. Kroz skrivanje informacija, aplikacijsko programsko sučelje podržava modularno programiranje, što omogućava korisnicima korištenje aplikacijskog programskog sučelja neovisno o implementaciji [16].

6.1.1 **REST API**

REST je kratica za reprezentativni prijenos stanja (engl. *representational state transfer*), a predstavlja arhitekturu za razvoj distribuiranih hipermedijskih sustava koju je prvi put predstavio Roy Fielding 2000. godine [17].

Kao i svaka arhitektura, REST sadrži šest glavnih načela koja moraju biti zadovoljena kako bi se API mogao prozvati RESTful API-jem, a načela su sljedeća [18]:

1. klijent - poslužitelj
    - odvajanjem problema korisničkog sučelja od problema pohrane podataka poboljšavamo prenosivost korisničkog sučelja na više platformi i poboljšavamo skalabilnost pojednostavljivanjem komponenata poslužitelja

2. nepostojani sustav (engl. *stateless*)
    - svaki zahtjev klijenta prema poslužitelju mora sadržavati sve podatke potrebne za razumijevanje zahtjeva i ne može iskoristiti bilo koji pohranjeni kontekst na poslužitelju, stoga je stanje sesije u potpunosti na klijentu
3. mogućnost predmemoriranja (engl. *cacheable*)
    - ograničenja predmemorije zahtijevaju da se podatci unutar odgovora (engl. *response*) na zahtjeva (engl. *request*) implicitno ili eksplicitno označe kao predmemorirani ili nepredmemorirani
    - ako je odgovor moguće predmemorirati, tada se klijentskoj predmemoriji daje pravo ponove upotrebe tih podataka za kasnije, ekvivalentne zahtjeve
4. jedinstveno sučelje (engl. *uniform interface*)
    - primjenom načela općenitosti softverskog inženjerstva na sučelje komponenata pojednostavljuje se cjelokupna arhitektura sustava i poboljšava se vidljivost interakcija sa sustavom
    - da bi se ostvarilo jedinstveno sučelje, potrebna su ograničenja koja određuju ponašanje komponenata sustava
    - REST arhitektura je definirana s četiri ograničenja sučelja:
        1. identifikacija resursa
        2. manipulacija resursima kroz reprezentacije
        3. samoopisne poruke
        4. hipermedija kao osnova aplikacijskog stanja
5. slojeviti sustav
    - slojeviti stil sustava omogućava da se arhitektura sastoji od hijerarhijskih slojeva koji ograničavaju ponašanje komponenata tako da se svaka komponenta ne može “vidjeti” dalje od neposrednog sloja s kojim se komunicira
6. kôd na zahtjev (neobavezno)
    - REST omogućuje proširenja funkcionalnosti klijenta preuzimanjem i izvršavanjem koda u obliku apleta ili skripti
    - pojednostavljuje klijente smanjenjem broja značajki koje je potrebno unaprijed implementirati

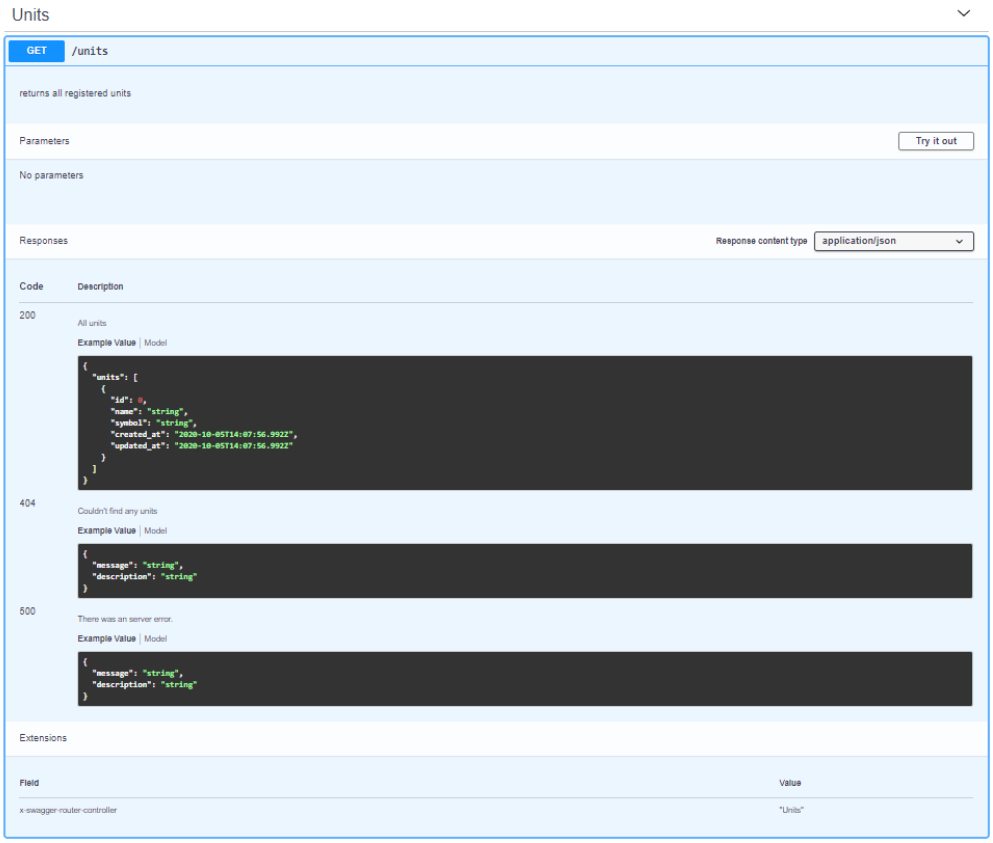
Tijekom razvoja *IoT Weather Station API-a* nastojali smo se pridržavati načela RESTful arhitekture kako bi cijeli *IoT Weather Station* sustav bio u skladu sa standardima i potrebama modernih web sustava.

6.1.2 **API dokumentacija**

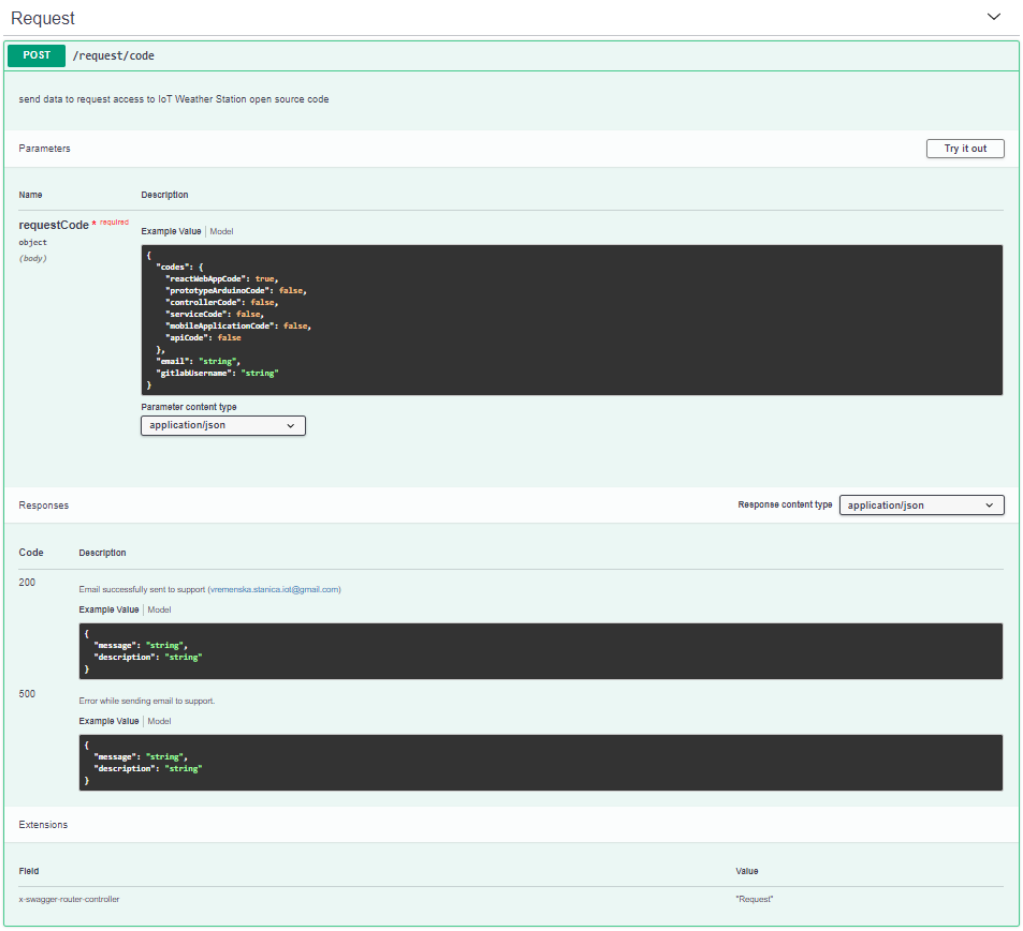
API omogućava korištenje funkcionalnosti sustava krajnjim korisnicima. Kako bi korisnici znali uputiti zahtjeve prema aplikacijskom programskom sučelju, potrebno je kreirati dokumentaciju, odnosno specifikaciju aplikacijskog programskog sučelja. U tu svrhu korišten je web aplikacija Swagger. Swagger je u suštini opisni jezik sučelja koji služi za opisivanje RESTful API-a u YAML i JSON obliku [19]. U dokumentaciji se opisuju metode aplikacijskog programskog sučelja, kako ih pozvati, koje podatke i u kojem obliku poslati uz zahtjev te kakav odgovor očekivati od aplikacijskog programskog sučelja.

Na slici 6.1 prikazan je primjer Swagger specifikacije *get* API metode *IoT Weather Station* sustava koja vraća sve mjerne jedinice upisane u bazu podataka.

Na slici 6.2 prikazan je primjer Swagger specifikacije *post* API metode *IoT Weather Station* sustava pomoću koje se zadaje zahtjev za pristup kodu dijelova *IoT Weather Station* sustava.



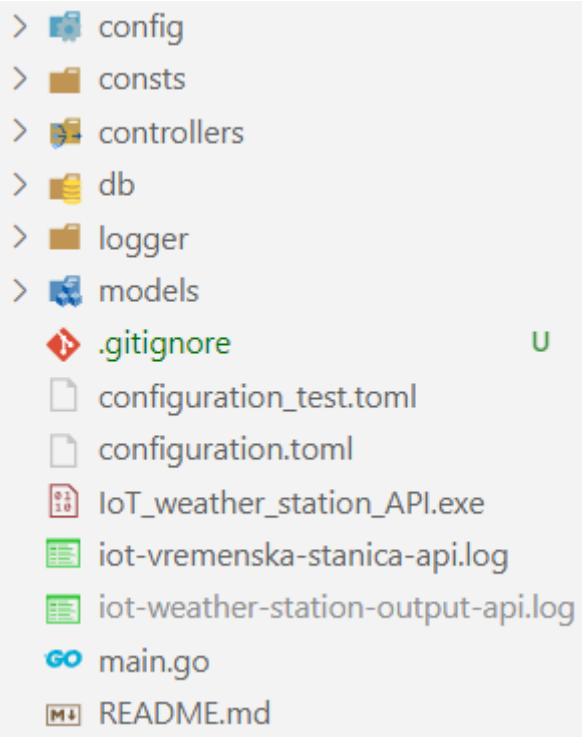
Slika 6.1: Swagger specifikacija GET metode



Slika 6.2: Swagger specifikacija POST metode

6.2 Funkcionalnosti IoT Weather Station API-a

Temeljna funkcionalnost *IoT Weather Station API-a* je posluživanje aplikacijskog programskog sučelja pomoću HTTP poslužitelja. Na slici 6.3 prikazano je stablo mapa i datoteka koje *IoT Weather Station API* koristi za realizaciju funkcionalnosti. Svaka mapa predstavlja zaseban Go paket koji obavlja neku od funkcionalnosti, npr. *logger* paket obavlja funkcionalnost zapisivanja grešaka u *iot-weather-station-output-api.log* datoteku. U poglavlju o bazama podataka (poglavlje 4) opisali smo funkcije i pakete koji odrađuju operacije nad bazom podataka, dok ćemo u ovome poglavlju detaljnije opisati pakete koje *IoT Weather Station API* koristi za posluživanje i izvršavanje metoda aplikacijskog programskog sučelja.



Slika 6.3: Stablo mapa i datoteka

6.2.1 **HTTP poslužitelj**

Na preporuku mentora za posluživanje HTTP poslužitelja odabran je *github.com/labstack/echo* Go paket naziva Echo. Echo je proširiva i minimalistička Go web razvojna cjelina (engl. *framework*) visokih performansi [20]. U programskom kodu 6.1 prikazano je pokretanje Echo HTTP poslužitelja, a koraci su sljedeći: Prije pokretanja Echo HTTP poslužitelja potrebna je inicijalizacija *Controller* objekta koji sadrži metode aplikacijskog programskog sučelja. Nakon inicijalizacije *Controller* objekta u varijablu *dc*, inicijalizira se i konfigurira *Echo* objekt koji sadrži konfiguraciju potrebnu za pokretanje HTTP poslužitelja. Kada je Echo objekt u varijabli *r* inicijaliziran i konfiguriran, preostaje nam povezivanje funkcija *Controller* objekta s odgovarajućim metodama i URL-ovima te pokretanje Echo HTTP poslužitelja. Slika 6.4 prikazuje pokrenuti Echo HTTP poslužitelj.

Programski kod 6.1: Pokretanje Echo HTTP poslužitelja

```
dc := controllers.NewController(db, l)

r := echo.New()
r.Use(middleware.CORS())
r.Use(middleware.Logger())
r.Use(middleware.Recover())

v1 := r.Group("/v1")
{
    hub := controllers.NewHub()
    go hub.Run()

    v1.GET("/stations", dc.GetStations)
    v1.GET("/stations/:stationID", dc.GetStation)
    v1.GET("/sensors", dc.GetSensors)
    v1.GET("/sensors/:sensorID", dc.GetSensor)
    v1.GET("/units", dc.GetUnits)
    v1.GET("/units/:unitID", dc.GetUnit)
    v1.GET("/conditions", dc.GetConditions)
    v1.GET("/exceptions", dc.GetExceptions)
    v1.GET("/values", dc.GetValues)
    v1.POST("/request/code", dc.RequestCodeEmail)

    v1.GET("/ws", func(c echo.Context) error {
        return dc.WsEndpoint(hub, c)
    })
}

l.Info("Starting HTTP server...")
r.Logger.Fatal(r.Start(c.Server.Address))
```

```
   ____    __
  / __/___/ /  ___
 / _// __/ _ \/ _ \
/___/\__/_//_/\___/ v4.1.11
High performance, minimalist Go web framework
https://echo.labstack.com
____________________________________O/_______
                                    O\
⇨ http server started on 127.0.0.1:8001
```

Slika 6.4: Sučelje pokrenutog Echo HTTP poslužitelja

6.2.2 **API metode**

Funkcionalnosti *IoT Weather Station API-a* dostupne su kroz metode aplikacijskog programskog sučelja, a koriste se dvije HTTP metode za dohvat i slanje podataka. *Get* HTTP metoda koristi se za dohvat podataka o vremenskim stanicama iz baze podataka uz jednu iznimku: API metoda */ws* koristi se za *WebSocket* komunikaciju te je zasebno opisana u poglavlju 6.2.3. *Post* HTTP metoda koristi se za slanje zahtjeva za pristup kodu dijelova *IoT Weather Station* sustava.

Kako sve *get* metode imaju sličnu funkcionalnost, za primjer ćemo proći kroz tok (engl. *flow*) */units/:unitID* API metode. Prema URL-u API metode možemo zaključiti da je njena funkcija dohvaćanje mjernih jedinica prema jedinstvenom identifikatoru (engl. *ID*). Prilikom poziva */units/:unitID* API metode poziva se *GetUnit* funkcija prikazana u programskom kodu 6.2.

Programski kod 6.2: GetUnit funkcija

```
// GetUnit get information about specific unit
func (dc *Controller) GetUnit(c echo.Context) error {
    return ExecuteEndpoint(dc.getUnit, dc, c)
}
```

Sljedeći korak u toku izvođenja API metode je poziv *ExecuteEndpoint* funkcije (programski kod 6.3), čija je uloga pozivanje stvarne funkcije API metode, u ovome slučaju *getUnit* funkcija (programski kod 6.4), i izvršavanje *InsertGodView* funkcije u novoj Go dretvi (engl. *goroutine*).

Programski kod 6.3: ExecuteEndpoint funkcija

```
// ExecuteEndpoint executes function with God View insertion
func ExecuteEndpoint(fn Endpoint, dc *Controller, c echo.Context) (error){
    start := time.Now()
    err := fn(c)
    stop := time.Now()
    go InsertGodView(dc, c, start, stop, err)
    return err
}
```

Funkcija *getUnit* (programski kod 6.4), koja je dio *controller* paketa, odrađuje provjeru valjanosti jedinstvenog identifikatora mjerne jedinice i poziva funkciju

GetUnitByID, iz *models* paketa (poglavlje 4.2.1), za dohvat mjerne jedinice iz baze podataka po jedinstvenom identifikatoru. Funkcija *GetUnitByID* vraća objekt tipa *Units* koji predstavlja podatke o mjernim jedinicama, nakon čega se taj objekt serijalizira u JSON format i šalje pozivatelju API metode.

Programski kod 6.4: getUnit funkcija

```
func (dc *Controller) getUnit(c echo.Context) error {
    strUnitID := c.Param(consts.UnitID)
    unitID, isValid := IsIDValid(strUnitID)
    if !isValid {
        return c.JSON(400, ErrorResponse{"Bad request", "Unit ID must be an integer and bigger then 0"})
    }
    s, err := models.GetUnitByID(dc.db, unitID)

    switch {
    case s != nil:
        return c.JSON(200, s)
    case err != nil:
        dc.logger.Error(err.Error())
        return c.JSON(500, ErrorResponse{"Error while getting unit", "There was an server error while getting unit"})
    default:
        return c.JSON(404, ErrorResponse{"Unit not found", "Unit with specified ID was not found"})
    }
}
```

API metoda */request/code* koja je se poziva pomoću *post* HTTP metode, prolazi kroz isti proces kao i *get* metode, uz dodatno primanje *RequestCode* objekta potrebnog za odrađivanje funkcionalnosti metode. A funkcionalnost */request/code* API metode je slanje *RequestCode* objekta na e-mail adresu podrške *IoT Weather Station* sustava (*vremenska.stanica.iot@gmail.com*) putem SMTP poslužitelja.

Pri svakome izvršavanju API metode poziva se *InsertGodView* funkcija (programski kod 6.3), a njena uloga je upisivanje podataka o korisniku i pozvanoj API metodi u bazu podataka, u svrhu optimiziranja rada *IoT Weather Station* sustava i sprječavanja DoS i DDoS napada.

6.2.3 **WebSocket**

WebSocket je računalni komunikacijski protokol koji omogućava potpuni dvosmjerni (engl. *full-duplex*) komunikacijski kanal preko jedne TCP veze, odnosno omogućava dvosmjernu komunikaciju između klijentske aplikacije i web poslužitelja [21].

Za potrebe *WebSocket* komunikacije kreirana je posebna API metoda: */ws*. Njena uloga je registracija klijenata prilikom poziva */ws* API metode te pozivanje funkcija za čitanje i slanje poruka na aplikacijskom programskom sučelju.

Uloga *WebSocket-a* u *IoT Weather Station* sustavu je slanje podataka, pristiglih sa senzora vremenskih stanica, s *IoT Weather Station Service-a* prema *IoT Weather Station API-u*. I slanje istih podataka s *IoT Weather Station API-a* prema klijentskim aplikacijama, u našem slučaju prema *Weather App* web aplikaciji.

Korištenjem *WebSocket-a* ostvaruje se razmjena podataka u stvarnome vremenu (engl. *real-time*).

7. IOT WEATHER STATION SERVICE

IoT Weather Station Service je web servis (engl. *web service*) *IoT Weather Station* sustava. U suštini predstavlja glavni dio teme završnog rada jer je njegova svrha posluživanje web servisa za dohvaćanje podataka o vremenskim stanicama s MQTT posrednika te provjera i unos (skladištenje) istih u bazu podataka. *IoT Weather Station Service* predstavlja zasebnu cjelinu čija je funkcionalnost unos podataka u bazu podataka te ga stoga možemo nazvati i mikroservisom (engl. *microservice*) *IoT Weather Station* sustava.

U ovome poglavlju detaljno su opisane funkcionalnosti i mogućnosti *IoT Weather Station* web servisa.

7.1 Web servis

Web servis je softverski sustav dizajniran za podršku interoperabilnoj interakciji stroj prema stroju preko mreže. Sadrži sučelje opisano u strojno obradivom formatu (WSDL). Ostali sustavi komuniciraju s web servisom na način propisan njegovim opisom, odnosno SOAP porukama koje se prenose pomoću HTTP-a s XML, u današnje vrijeme JSON, serijalizacijom zajedno s drugim web standardima [22].

Iz definicije web servisa možemo zaključiti da *IoT Weather Station Service* nema ulogu uobičajenog web servis, ta uloga je prepuštena *IoT Weather Station API-u*. *IoT Weather Station Service* se nalazi na poslužitelju, komunicira s MQTT posrednikom koji služi kao opskrbljivač podatcima i komunicira sa bazom podataka u svrhu dohvata i unosa podataka. To su jedine značajke web servisa koje *IoT Weather Station Service* primjenjuje.

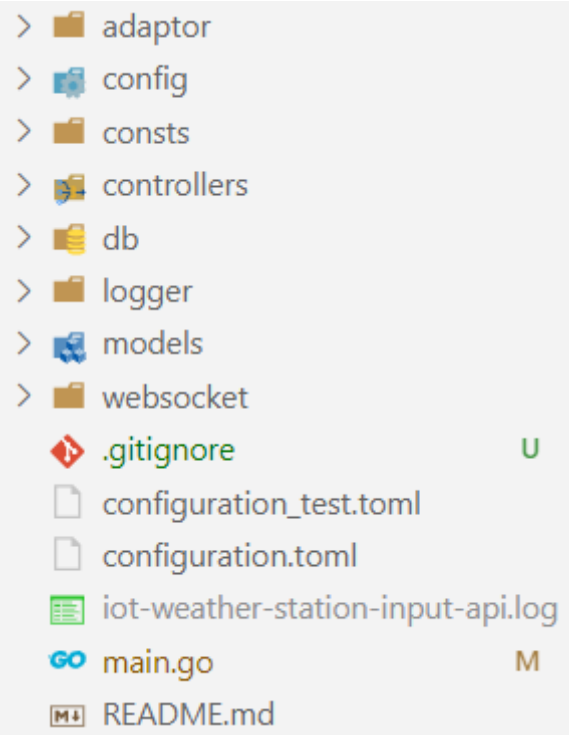
7.2 Mikroservis

Mikroservis je arhitektura, odnosno varijanta servisne arhitekture (SOA) strukturnog stila. Mikroservis uređuje aplikaciju kao zbirku povezanih usluga koje obavljaju određene funkcionalnosti i koriste lagane (engl. *lightweight*) protokole [23].

IoT Weather Station Service kao web servis obavlja jednu funkcionalnost *IoT Weather Station* sustava što ga ujedno čini i web servisom i mikroservisom.

7.3 Funkcionalnosti IoT Weather Station Service-a

Temeljna funkcionalnost *IoT Weather Station Service-a* je posluživanje web servisa pomoću Gobot razvojne cjeline (engl. *framework*). Na slici 7.1 prikazano je stablo mapa i datoteka koje *IoT Weather Station Service* koristi za realizaciju funkcionalnosti. Svaka mapa predstavlja zaseban Go paket koji obavlja neku od funkcionalnosti, npr. *logger* paket obavlja funkcionalnost zapisivanja grešaka u *iot-weather-station-input-api.log* datoteku. U poglavlju o bazama podataka (poglavlje 4) opisali smo funkcije i pakete koji odrađuju operacije nad bazom podataka, dok ćemo u ovome poglavlju detaljnije opisati pakete koje *IoT Weather Station Service* koristi za posluživanje i izvršavanje funkcija provjere i unosa podataka u bazu podataka.



Slika 7.1: Stablo mapa i datoteka

7.3.1 **Servis**

Na preporuku mentora za posluživanje web servisa odabran je *gobot.io/x/gobot* Go paket naziva Gobot. Gobot je razvojna cjelina za robotiku, fizičko računanje i Internet stvari (IoT) napisana u programskom jeziku Go. "Robot" je glavna softverska apstrakcija Gobot razvojne cjeline koja olakšava razvoj funkcionalnosti visoke razine za podržane uređaje [24]. U svrhu *IoT Weather Station Service-a* razvijen je robot, odnosno Go program koji izvršava tri glavne funkcionalnosti: dohvaćanje konfiguracije i podataka s

vremenskih stanica putem MQTT posrednika, provjera valjanosti podataka i objavljivanje lažiranih podataka na MQTT posrednik kako bismo simulirali rad vremenskih stanica. U programskom kodu 7.1 prikazano je pokretanje Gobot robota, a koraci su sljedeći: Prije pokretanja Gobot robota potrebna je inicijalizacija *Mqtt* objekta koji sadrži konfiguraciju potrebnu za konekciju na MQTT posrednik. Nakon inicijalizacije *Mqtt* objekta u varijablu *ad*, inicijalizira se i kreira *Websocket* objekt koji sadrži konfiguraciju potrebnu za konekciju na *WebSocket IoT Weather Station API-a*. Nakon ostvarene konekcije na *WebSocket* kreira se *MqttController* objekt koji sadrži funkcije koje robot izvršava. Za kraj nam preostaje povezivanje funkcija *MqttController* objekta s robotom te pokretanje Gobot robota. Slika 7.2 prikazuje pokrenuti Gobot robot.

Programski kod 7.1: Pokretanje Gobot robota

```
ad, err := adaptor.Create(c.Mqtt)
if err != nil {
    l.Fatal(err.Error())
}

ws, err := websocket.Connect(c.Server.Address)
if err != nil {
    l.Fatal(err.Error())
}

mc := controllers.NewMqttController(ad, db, l, ws)

work := func() {

    mc.HandleConfiguration()
    mc.HandleStationData()

    /*************************************************/
    // FAKE INPUT DATA
    /*************************************************/

    gobot.Every(10*time.Second, func() {
        mc.PublishStationData("ws/18/data")
    })

    gobot.Every(20*time.Second, func() {
        mc.PublishStationData("ws/19/data")
    })

    /*************************************************/

    go mc.CheckDataHealth();

}

robot := gobot.NewRobot(
    []gobot.Connection{ad.Adaptor},
    work,
)

robot.Start()
```

```
2020/10/12 22:37:06 Initializing connection MQTT ...
2020/10/12 22:37:06 Robot 7C3D71288DE60AB0 initialized.
2020/10/12 22:37:06 Starting Robot 7C3D71288DE60AB0 ...
2020/10/12 22:37:06 Starting connections...
2020/10/12 22:37:06 Starting connection MQTT on port mqtt.vub.zone:1883...
2020/10/12 22:37:06 Starting devices...
2020/10/12 22:37:06 Starting work...
```

Slika 7.2: Sučelje pokrenutog Gobot robota

7.3.2 **Funkcije**

IoT Weather Station Service ima tri glavne funkcionalnosti koje ćemo opisati u ovome poglavlju. Sve tri funkcionalnosti nalaze se unutar *controllers* Go paketa razvijenog za potrebe *IoT Weather Station Service-a*.

Prva i najbitnija funkcionalnost je dohvaćanje podataka sa MQTT posrednika te provjera i unos istih u bazu podataka. Kao što smo naveli u poglavlju o implementaciji MQTT-a u *IoT Weather Station* sustavu (poglavlje 3.3), *IoT Weather Station Service* pretplaćuje se na dvije teme dostupne na MQTT posredniku: *ws/{stationId}/config* i *ws/{stationId}/data*. Za potrebe provjere i unosa podataka u bazu podataka te sukladno temama, razvijene su dvije funkcije: *HandleConfiguration* i *HandleStationData* (programski kod 7.2).

Programski kod 7.2: HandleConfiguration i HandleStationData funkcije

```
// HandleConfiguration handles configuration for stations
func (mc *MqttController) HandleConfiguration() {
    mc.Mqtt.Adaptor.OnWithQOS("ws/+/config", 0, mc.handleConfiguration)
}

// HandleStationData handles data from stations
func (mc *MqttController) HandleStationData() {
    mc.Mqtt.Adaptor.OnWithQOS("ws/+/data", 0, mc.handleData)
}
```

Kako *IoT Weather Station Service* funkcionira kao servis (stalno je pokrenut) funkcije *HandleConfiguration* i *HandleStationData* pozivaju se prilikom svake objave podataka s vremenske stanice (hardvera) na MQTT posrednik.

Funkcije *HandleConfiguration* i *HandleStationData* imaju sličnu funkcionalnost, razlika je u tome što *HandleConfiguration* funkcija odrađuje provjeru i unos konfiguracijskih podataka vremenske stanice, dok *HandleStationData* funkcija odrađuje provjeru i unos podataka prikupljenih sa senzora vremenskih stanica. Dodatno *HandleStationData* funkcija nakon unosa podataka u bazu podataka šalje iste prema *IoT Weather Station API-u* putem *WebSocket* konekcije ostvarene prilikom pokretanja Gobot robota.

Za primjer ćemo opisati tok *HandleConfiguration* funkcije. Prilikom objave podataka vremenske stanice na temu *ws/{stationId}/config*, poziva se *handleConfiguration* funkcija (programski kod 7.3). Funkcija odrađuje kreiranje *Configuration* objekta iz JSON

teksta (engl. *string*) dohvaćenog sa MQTT posrednika i poziva funkciju *InsertConfiguration*, iz *models* paketa, za dodatnu provjeru i unos konfiguracijskih podataka vremenske stanice u bazu podataka.

Programski kod 7.3: handleConfiguration funkcija

```
func (mc *MqttController) handleConfiguration(msg mqtt.Message) {
    mc.Logger.Info("handleConfiguration (msg: " + string(msg.Payload()) + ") topic: " + msg.Topic())
    var config models.Configuration

    strJSON := string(msg.Payload())
    err := json.Unmarshal([]byte(strJSON), &config)
    if err != nil {
        mc.Logger.Error(err.Error())
    }

    result, err := models.InsertConfiguration(mc.Database, &config)
    if err != nil {
        mc.Logger.Error(err.Error())
        err := SendEmailAlert("Configuration exception", "handleConfiguration (msg: " + string(msg.Payload()) + ") topic: " + msg.Topic(), err.Error())
        if err != nil {
            mc.Logger.Error(err.Error())
        }
    } else if result != nil {
        mc.Logger.Info("Station successfully inserted.")
    }

}
```

Ako se *InsertConfiguration* funkcija izvrši uspješno, odnosno nema grešaka, *Configuration* objekt je uspješno unesen u bazu podataka te se logira informacija o uspješnom unosu. A ukoliko *InsertConfiguration* funkcija vrati grešku, koja se mogla dogoditi ili prilikom provjere ili prilikom unosa podataka u bazu podataka, tada se logira greška i poziva se *SendEmailAlert* funkcija koja šalje e-mail sa sadržajem greške na e-mail adresu podrške *IoT Weather Station* sustava (*vremenska.stanica.iot@gmail.com*) putem SMTP poslužitelja.

Druga funkcionalnost *IoT Weather Station Service-a* je lažiranje podataka senzora vremenskih stanica. Ova funkcionalnost je privremena i koristi se samo za svrhu završnog rada jer smo izgubili podršku za hardver *IoT Weather Station* sustava. Za potrebe lažiranja podataka senzora vremenskih stanica kreirana je funkcija *publishStationData* prikazana u programskom kodu 7.4. Njena funkcija je kreiranje nasumičnih vrijednosti vremenskih parametara (temperatura, tlak zraka, vlaga, itd.) i objava vremenskih parametara na MQTT posrednik u JSON formatu.

Programski kod 7.4: publishStationData funkcija

```
/*************************************************/
// INPUT FAKE DATA (TEMP SOLUTION FOR STUDENT PROJECT)
/*************************************************/

func publishStationData(mc *MqttController, topic string) {
    rand.Seed(time.Now().UnixNano())
    pressure := rangeIn(1030, 1045)
    temperature := 17 + rand.Float64()*(22-17)
    humidity := rangeIn(55, 60)
    gas := rangeIn(60, 80)
    altitude := rangeIn(80, 110)

    strJSON := fmt.Sprintf("{ \"DHT22\": { \"Temperature\": \"%.2f\", \"Humidity\":\"%v\" },
    data := []byte(strJSON)
    mc.Mqtt.Adaptor.Publish(topic, data)
}

func rangeIn(low, hi int) int {
    return low + rand.Intn(hi-low)
}
```

Trenutno *IoT Weather Station* sustav sadrži dvije vremenske stanice, pa sukladno s time *IoT Weather Station Service* izvršava objavu lažiranih podataka za obje stanice, kao što je prikazano u programskom kodu 7.5.

Programski kod 7.5: Izvršavanje PublishStationData funkcije

```
/*************************************************/
// FAKE INPUT DATA
/*************************************************/

gobot.Every(10*time.Second, func() {
    mc.PublishStationData("ws/18/data")
})

gobot.Every(20*time.Second, func() {
    mc.PublishStationData("ws/19/data")
})

/*************************************************/
```

Treća funkcionalnost *IoT Weather Station Service-a* je provjera valjanosti podataka pristiglih sa senzora vremenskih stanica. Za tu potrebu razvijena je *checkDataHealth* funkcija (programski kod 7.6) koja se izvršava svakih pet minuta. Njena uloga je provjera stanja vremenskih stanica. Ako je posljednji podataka, sa senzora vremenskih stanica,

unesen u bazu podataka prije više od dva sata od trenutka poziva funkcije, tada se pomoću *SendEmailAlert* funkcije obavještava podrška *IoT Weather Station* sustava (*vremenska.stanica.iot@gmail.com*) o neispravnoj vremenskoj stanici.

Programski kod 7.6: checkDataHealth funkcija

```
func(mc *MqttController) checkDataHealth() {

    gobot.Every(5*time.Minute, func() {

        stations, err := models.GetAllStations(mc.Database)
        if err != nil {
            mc.Logger.Error(err.Error())
        }

        for _, station := range stations.StationsList {

            values, err := models.GetValuesByStationID(mc.Database, station.ID)
            if err != nil {
                mc.Logger.Error(err.Error())
            }

            timeThreshold := time.Now().Add(-2 * time.Hour)

            // fmt.Println(timeThreshold)
            // fmt.Println(values.ValuesList[0].UpdatedAt)

            if values.ValuesList[0].UpdatedAt.Before(timeThreshold)  {
                err = SendEmailAlert("Data exception", "Station (ID: " + fmt.Sp
                if err != nil {
                    mc.Logger.Error(err.Error())
                }
                mc.Logger.Error("Data exception Station (ID: " + fmt.Sprint(sta
            }
        }
    })
}
```

8. Weather app

Weather App je web aplikacija *IoT Weather Station* sustava razvijena pomoću JavaScript biblioteke (engl. *library*) React. Svrha *Weather App* web aplikacije je prikaz podataka prikupljenih pomoću *IoT Weather Station Service-a* i dohvaćenih pomoću *IoT Weather Station API-a*. *Weather App* je u direktnoj komunikaciji s *IoT Weather Station API-em*, i koristi sve metode aplikacijskog programskog sučelja.

U ovome poglavlju detaljno je opisano grafičko korisničko sučelje i mogućnosti *Weather App* web aplikacije.

8.1 React

React (poznat i kao React.js ili ReactJS) je otvorena (engl. *open-source*) JavaScript biblioteka (engl. *library*) za izgradnju korisničkih sučelja. Održavaju ga Facebook i zajednica pojedinačnih programera i tvrtki, a koristi se kao osnova u razvoju web ili mobilnih aplikacija. Glavna funkcija React-a je prikazivanje podataka u DOM-u, pa razvoj React aplikacija obično zahtijeva upotrebu dodatnih biblioteka za upravljanje i usmjeravanje stanja kao što su Redux i React Router, koji su korišteni i u razvoju *Weather App* web aplikacije [25].

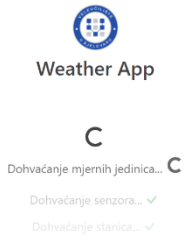
8.2 Grafičko korisničko sučelje

Grafičko korisničko sučelje *Weather App* web aplikacije sastoji se od šest glavnih React komponenti, odnosno web stranica. Za kontrolu podataka i komponenti korištena je Redux biblioteka, koja služi kao glavni spremnik stanja podataka i komponenti u *Weather App* web aplikaciji. Redux koristimo za dohvaćanje podataka s *IoT Weather Station API-a* te za prikaz određenih komponenti web aplikacije. Dodatno je korišten i React Router koji služi za navigaciju i povezivanje glavnih komponenti u jednu kompleksnu web aplikaciju.

8.3 Učitavanje web aplikacije

Prilikom otvaranje Weather App web aplikacije pokreće se učitavanje aplikacije (slika 8.1). Prilikom učitavanja aplikacije dohvaćaju se podatci potrebni za rad Weather App web aplikacije. Odrađuje se poziv IoT Weather Station API-a za dohvat vremenskih stanica, senzora, mjernih jedinica, posljednjih vrijednosti vremenskih stanica i spajanje na

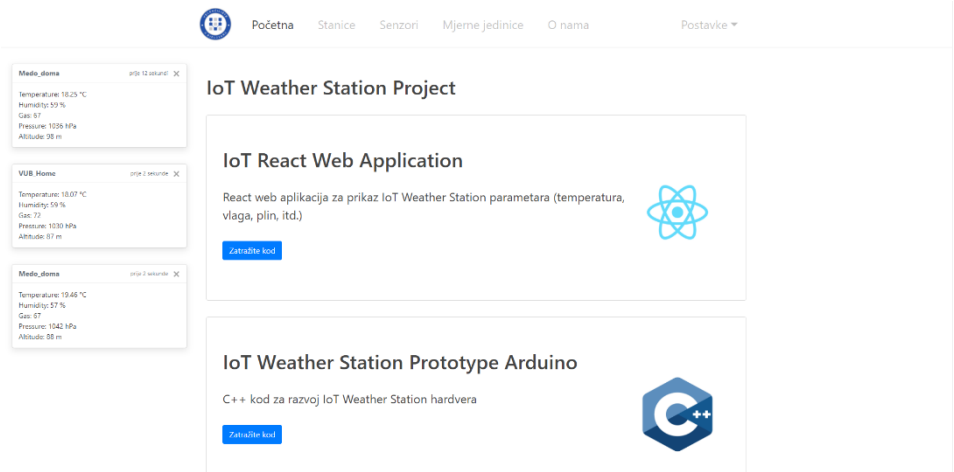
WebSocket. Nakon učitavanja korisnik se preusmjerava na početnu stranicu web aplikacije.



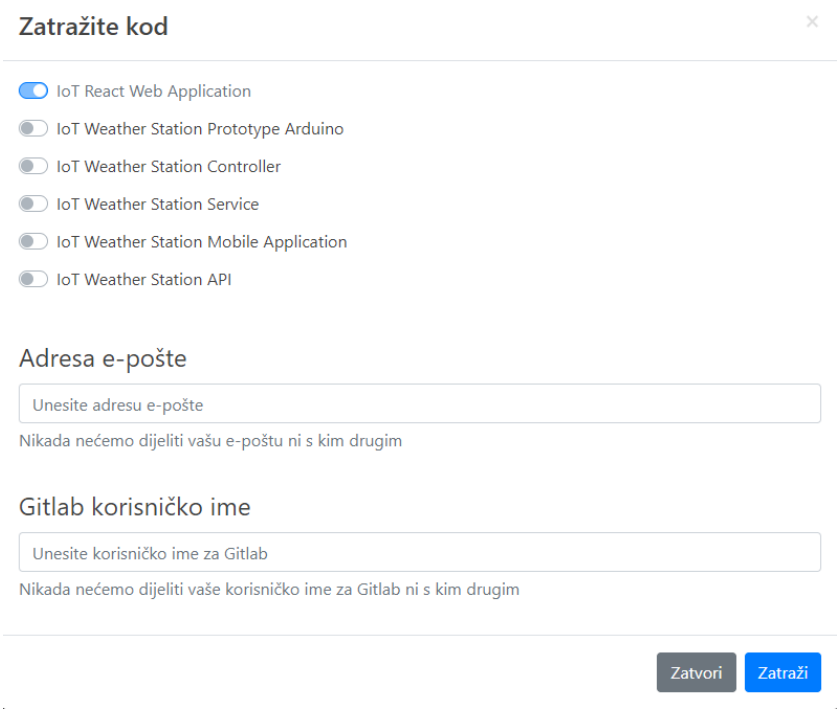
Slika 8.1: Učitavanje Weather App web aplikacije

8.4 Početna stranica

Početna stranica web aplikacije (slika 8.2) sadrži opise dijelova *IoT Weather Station* sustava i funkcionalnost zatraživanja koda za jedan ili više dijelova *IoT Weather Station* sustava. Kako bi korisnik zatražio pristup kodu nekome dijelu sustava, mora unijeti e-mail adresu i Gitlab korisničko ime te odabrati željene dijelove sustava kao što je prikazano na slici 8.3.



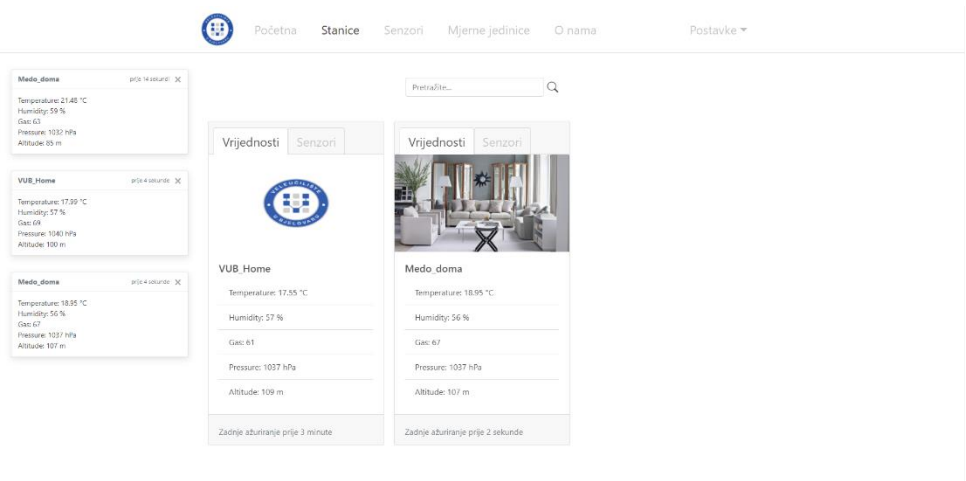
Slika 8.2: Početna stranica



Slika 8.3: Forma zahtjeva pristupu kodu

8.5 Stanice

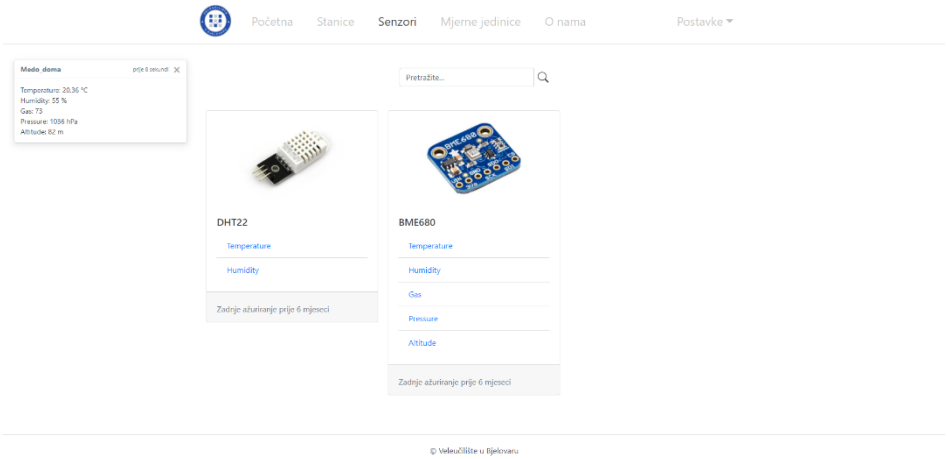
Stranica o vremenskim stanicama prikazuje dostupne vremenske stanice *IoT Weather Station* sustava u obliku kartica (slika 8.4). Kartice sadrže dvije informacije. Prva informacija su vrijednosti stanica dohvaćene preko *WebSocket-a* ili preko baze podataka ako je *WebSocket* komunikacija nedostupna, a druga informacija su senzori koje vremenska stanica koristi za dohvaćanje vrijednosti vremenskih parametara (temperatura, vlaga, tlak zraka, plinovi, itd.).



Slika 8.4: Web stranica Stanice

8.6 Senzori

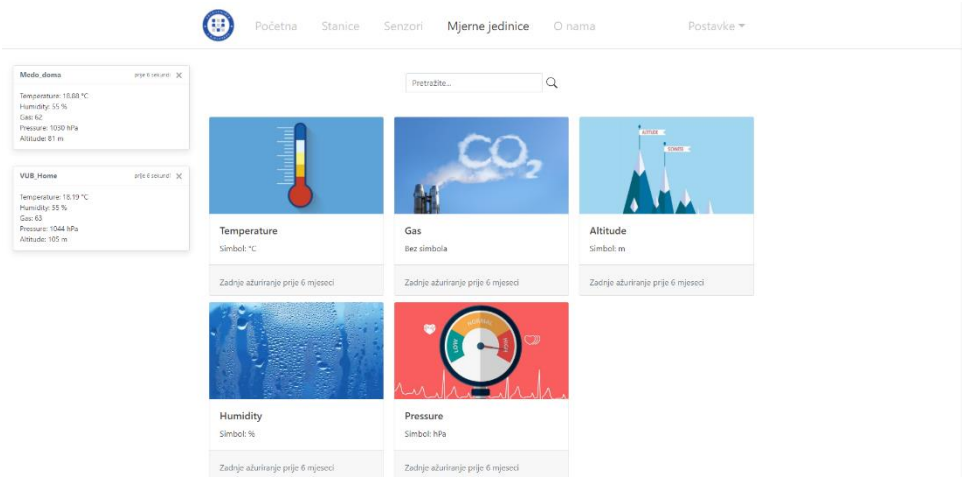
Stranica o senzorima vremenskih stanica prikazuje dostupne senzore vremenskih stanica *IoT Weather Station* sustava u obliku kartica (slika 8.5). Kartice sadrže informacije o mjernim jedinicama senzora.



Slika 8.5: Web stranica Senzori

8.7 Mjerne jedinice

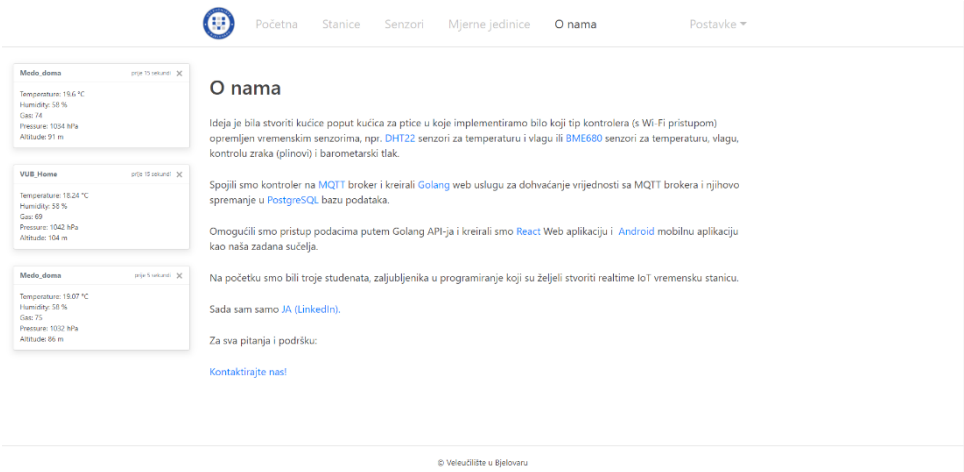
Stranica o mjernim jedinicama prikazuje mjerne jedinice koje podržavaju senzori vremenskih stanica *IoT Weather Station* sustava u obliku kartica (slika 8.6). Kartice sadrže informacije o nazivu i simbolu mjerne jedinice.



Slika 8.6: Web stranica Mjerne jedinice

8.8 O nama

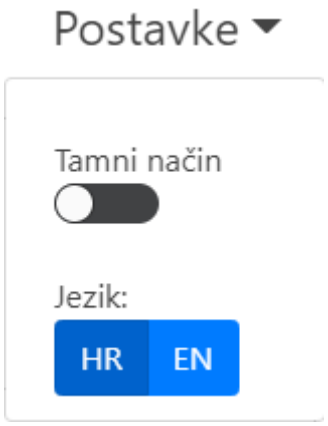
Stranica o nama prikazuje članak s osnovnim informacijama o *IoT Weather Station* sustavu (slika 8.7).



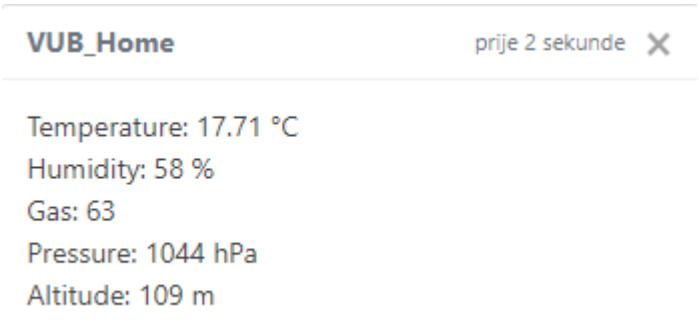
Slika 8.7: Web stranica O nama

8.9 Dodatne funkcionalnosti

Dodatne funkcionalnosti *Weather App* web aplikacije su postavke i *WebSocket* notifikacije. Postavke (slika 8.8 a)) omogućavaju promjenu teme i jezika. Dostupne su svijetla i tamna tema te hrvatski i engleski jezik. Obje postavke mijenjaju se pomoću gumba za prebacivanje (engl. s*witch button*). Iz slika web stranica može se vidjeti kako su *WebSocket* notifikacije (slika 8.8 b)) dostupne kroz sve stranice *Weather App* web aplikacije, a prikazuju osvježene vrijednosti vremenskih stanice u stvarnome vremenu (engl. *real-time*).



a) Postavke Weather App web aplikacije



b) WebSocket notifikacija

Slika 8.8: Dodatne funkcionalnosti Weather App web aplikacije

9. ZAKLJUČAK

Tema ovog završnog rada je skladištenje, odnosno pohrana podataka u bazu podataka prikupljenih pomoću MQTT posrednika. Završni rad se temelji na kolegiju Internet stvari. Ciljevi završnog rada su sljedeći: opis hardvera za prikupljanje podataka o vremenskim uvjetima baziranog na ESP32 mikroupravljaču s implementiranim DHT22 i BME680 senzorima, opis MQTT protokola za objavu podataka prikupljenih s hardvera na MQTT posrednik, opis i izrada baze podataka potrebne za pohranu prikupljenih podataka, opis i izrada web servisa za dohvaćanje podataka sa MQTT posrednik i pohranu istih u bazu podataka, opis i izrada aplikacijskog programskog sučelja za dohvat prikupljenih podataka i opis i izrada grafičkog korisničkog sučelja u obliku web aplikacije za prikaz prikupljenih podataka. Pri izradi sustava korišteni su: ESP32 mikroupravljač sa DHT22 i BME680 senzorima, DB Designer web aplikacija, PostgreSQL baza podataka, Programski jezik Go, Swagger web aplikacija, JavaScript biblioteka React i znanja i istraživanja vezana uz kolegije Web programiranje 1, Web programiranje 2, Baze podataka i Internet stvari. Temeljni dio sustava čine baza podataka, *IoT Weather Station Service* (web servis) i *IoT Weather Station API* (aplikacijsko programsko sučelje). U radu su opisani softverski dijelovi *IoT Weather Station* sustava, a prijedlozi za daljnje unaprjeđenje *IoT Weather Station* sustava su: izrada pouzdanijeg hardvera za prikupljanje podataka i korištenje senzora s više mogućnosti kako bi se prikupilo što više podataka o vremenskim uvjetima i osigurala pouzdanost sustava, upotreba enkripcije na svim razinama sustava kako ne bi došlo do ugrožavanja sigurnosti podataka, osiguravanje aplikacijskog programskog sučelja (najranjivija točka) upotrebom API posrednika (engl. *API gateway*) za filtriranje zahtjeva i obranu od zlonamjernih napada poput DoS i DDoS napada i opširnije dokumentiranje cijelog *IoT Weather Station* sustava. Ovaj završni rad predstavlja sustav za manipulaciju podataka vremenskih uvjeta. Uz preinake, navede u ovome poglavlju, *IoT Weather Station* sustav može postati stvaran sustav za mjerenje podataka vremenskim uvjetima u stvarnome vremenu (engl. *real-time*).

10. LITERATURA

[1] ISO/IEC 20922:2016 Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1 [Online]. 2016. Dostupno na: https://www.iso.org/standard/69466.html (10.10.2020)

[2] MQTT Version 5.0 OASIS Standard Specification [PDF]. 2019. Dostupno na: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf (10.10.2020)

[3] Getting Started with MQTT [Online]. 2020. Dostupno na: https://www.hivemq.com/blog/how-to-get-started-with-mqtt/ (10.10.2020)

[4] MQTT, službena stranica [Online]. 2020. Dostupno na: https://mqtt.org/ (10.10.2020)

[5] Publish Subscribe Architecture [PDF]. Dostupno na: https://www.student.cs.uwaterloo.ca/~cs446/1171/Arch_Design_Activity/PublishSubscribe.pdf (10.10.2020)

[6] Mosquitto, službena stranica [Online]. 2020. Dostupno na: https://mosquitto.org/ (10.10.2020)

[7] PostgreSQL, službena stranica [Online]. 2020. Dostupno na: https://www.postgresql.org/ (10.10.2020)

[8] Difference Between Oracle and PostgreSQL [Online]. 2020. Dostupno na: https://www.educba.com/oracle-vs-postgresql/ (10.10.2020)

[9] Romanowski J.: Which Major Companies Use PostgreSQL? What Do They Use It for? [Online]. 2020. Dostupno na: https://learnsql.com/blog/companies-that-use-postgresql-in-business/ (10.10.2020)

[10] SQLite vs MySQL vs PostgreSQL: A comparison Of Relational Database Management System [Online]. 2019. Dostupno na: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems (10.10.2020)

[11] Stonebraker M., A. Rowe L.: The design of POSTGRES [PDF]. 1986. Dostupno na: https://dsf.berkeley.edu/papers/ERL-M85-95.pdf (10.10.2020)

[12] PostgreSQL 9.4.26 Documentation, službena dokumentacija [Online]. 2020. Dostupno na: https://www.postgresql.org/docs/9.4/ (10.10.2020)

[13] Adamović T., Husak K., Mutka A. Konkurentnost u programskom jeziku Go [PDF]. 2018. Dostupno na: https://hrcak.srce.hr/202072 (10.10.2020)

[14] Kincaid J.: Google's Go: A New Programming Language That's Python Meets C++ [Online]. 2009. Dostupno na: https://techcrunch.com/2009/11/10/google-go-language/?guccounter=1&guce_referrer=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnLw&guce_referrer_sig=AQAAAB6EFe9Z87bY--ctX65RHtc24K_jlrDZMi3hPSJ-a9zadNNVKj0F9cgUsg7bCt88A9mdfaT8XsaDHN44xvLvQ4p0ZQcrxsd2xDuWYOSG3pM0_lDtkxM-_9evn41vaU0XEgV6Veh5JLGFSvh3HutMAxEWNM-CL9yo5LtVPnFzoqku (10.10.2020)

[15] Go, službena stranica [Online]. 2020. Dostupno na: https://golang.org/ (10.10.2020)

[16] API-fication, www.hcltech.com stranica [Online]. 2014. Dostupno na: https://www.hcltech.com/sites/default/files/apis_for_dsi.pdf (10.10.2020)

[17] Fielding T. R.: Architectural Styles and the Design of Network-based Software Architectures [Disertacija]. 2000. Dostupno na: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (10.10.2020)

[18] RESTful API, službena stranica [Online]. 2020. Dostupno na: https://restfulapi.net/ (10.10.2020)

[19] Swagger, službena stranica [Online]. 2020. Dostupno na: https://swagger.io/ (10.10.2020)

[20] Echo, službena stranica [Online]. 2020. Dostupno na: https://echo.labstack.com/ (10.10.2020)

[21] Fette I., Melnikov A.: The WebSocket Protocol [Online]. 2011. Dostupno na: https://tools.ietf.org/html/rfc6455 (10.10.2020)

[22] W3C Working Group: Web Services Architecture [Online]. 2004. Dostupno na: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest (10.10.2020)

[23] Microservices vs. Web Services [Online]. 2020. Dostupno na: https://www.velvetech.com/blog/microservices-vs-web-services/#:~:text=Microservices%20are%20a%20software%20architecture,processes%20and%20communicate%20through%20APIs. (10.10.2020)

[24] Gobot, službena stranica [Online]. 2020. Dostupno na: https://gobot.io/ (10.10.2020)

[25] ReactJS, službena stranica [Online]. 2020. Dostupno na: https://reactjs.org/ (10.10.2020)

11. OZNAKE I KRATICE

API – engl. *Application programming interface*

DDL – engl. *Data Definition Language*

DDoS – engl. *Distributed Denial of Service*

DML – engl. *Data Manipulation Language*

DOM – engl. *Document Object Model*

DoS – engl. *Denial of Service*

DQL – engl. *Data Query Language*

HTTP – engl. *Hypertext Transfer Protocol*

IoT – engl. *Internet of things*

JSON – engl. *JavaScript Object Notation*

MQTT – engl. *MQ Telemetry Transport or Message Queuing Telemetry Transport*

OASIS – engl. *Organization of the Advancement of Structured Information Standards*

SMTP – engl. *Simple Mail Transfer Protocol*

SOA – engl. *Service-Oriented Architecture*

SOAP – engl. *Simple Object Access Protocol*

SQL – engl. *Structured Query Language*

TCL - engl. *Transaction Control Language*

TCP - engl. *Transmission Control Protocol*

URL - engl. *Uniform Resource Locator*

VUB – Veleučilište u Bjelovaru

WSDL – engl. *Web Service Definition Language*

YAML – engl. *YAML Ain't Markup Language*

12. SAŽETAK

Naslov: Web servis za skladištenje podataka dobivenih pomoću MQTT-a

Ovaj rad predstavlja razvoj *IoT Weather Station* sustava za manipulaciju podataka vremenskih uvjeta koji omogućava prikupljanje, pohranu, dohvat i prikaz podataka vremenskih uvjeta u stvarnome vremenu. Dijelovi *IoT Weather Station* sustava su: hardver realiziran pomoću ESP32 mikroupravljača sa DHT22 i BME680 senzorima, PostgreSQL baza podataka, MQTT posrednik, *IoT Weather Station Service* (web servis) za unos podataka u bazu podataka i *IoT Weather Station API* (aplikacijsko programsko sučelje) za dohvat podataka iz baze podataka, oboje razvijeni pomoću Go programskog jezika i *Weather App* web aplikacija razvijena pomoću JavaScript biblioteke React. Rad se temelji na kolegiju Internet stvari, odnosno tehnologiji Interneta stvari, uz primjenu baze podataka i web tehnologija poput web servisa, aplikacijskog programskog sučelja i web aplikacija. U radu je opisana softverska realizacija dijelova *IoT Weather Station* sustava.

Ključne riječi: uvjeti, MQTT, servis, API, Go, React.

13. ABSTRACT

Title: Web service for storing data obtained using MQTT

This paper presents the development of the *IoT Weather Station* weather conditions manipulation system that allows real-time weather data collection, storage, retrieval and display. The parts of the *IoT Weather Station* system are: hardware implemented using ESP32 microcontroller with DHT22 and BME680 sensors, PostgreSQL database, MQTT broker, *IoT Weather Station Service*, i.e., web service for data input into the database and *IoT Weather Station API*, i.e., application programming interface for retrieving data from the database, both developed using the Go programming language and the *Weather App* web application developed using the React JavaScript library. The paper is based on the course Internet of Things, i.e, Internet of Things technology, with the application of a database and web technologies such as a web service, application programming interface and web application. The paper describes the software implementation of parts of the *IoT Weather Station* system.

Keywords: conditions, MQTT, service, API, Go, React.

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 19.10.2020.	IVAN BREZIĆ	Ivan Brezić

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

IVAN BREZIĆ

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 19.10.2020.

Ivan Brezić

potpis studenta/ice