SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Diplomski studij

Pametno parkiralište

Seminarski rad

Matija Hajduković
Matej Kosić

Osijek, 2022.

SADRŽAJ

1. UVOD

U sklopu kolegija Internet Objekata potrebno je bilo osmisliti i izraditi projektni zadatak kojim će se demonstrirati rad pametnog javnog parkirališta. Parkiralište je zamišljeno kao otvoreno parkiralište kojem svaka osoba s automobilom ima pristup. Sustav se sastoji od 2 web stranice, jedna za korisnika, a druga za operatera, te ESP32 CAM modula koji na sebi ima ugrađenu web kameru kako bi se dobio prikaz parkirnih mjesta u stvarnom vremenu.

2. SPECIFIKACIJA ZAHTJEVA

2.1. Opis sustava

Pomoću pripadajuće web stranice omogućeno je operateru da neprestano ažurira zauzeta/slobodna mjesta pomoću video prijenosa sa ESP32 CAM modula.

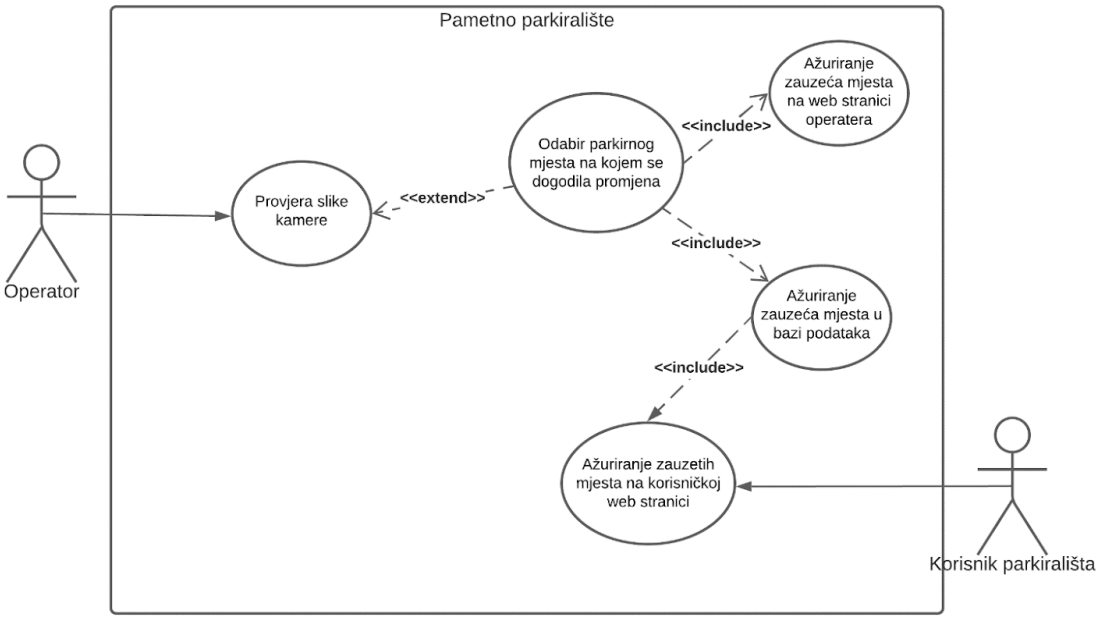
Klikom na određeno parkirno mjesto na web stranici operatera mijenja se stanje parkirališta iz slobodnog u zauzeto i obratno te se isto ažurira u bazi podataka. Napravljena je dodatna tablica koja omogućava operateru tablični pregled zauzetih/slobodnih mjesta.

Također je napravljena druga web stranica koja će služiti kao prikaz parkirnih mjesta samom korisniku na ulazu u parkiralište. Na web stranici se nalazi tlocrt samog parkirališta na kojem su zeleno obojana slobodna mjesta, a crveno obojana zauzeta mjesta. Podaci o zauzetim mjestima se dobivaju iz baze podataka u koju je web stranica operatera upisivala.

Kao dodatna mogućnost omogućeno je paljenje ugrađene LED diode koja simulira rasvjetno tijelo na parkiralištu kako bi sustav funkcionirao i po noći. Svjetlo će se paliti ukoliko razina svjetline padne ispod određene granice, a samu vrijednost svjetline mjeri fotootpornik.
Kroz sam projekt je omogućena buduća nadogradnja u vidu umjetne inteligencije (engl. *Artificial intelligence*) koja će zamijeniti samog operatera i biti u stanju prepoznati je li parkirno mjesto zauzeto ili slobodno.

2.2. Use case dijagram

Use case dijagram izrađen je pomoću Lucide web aplikacije. Sam use case dijagram je vrlo jednostavan. Sve počinje od operatera koji provjerava je li se dogodila ikakva promjena na parkiralištu, je li se neki auto isparkirao ili uparkirano neki novi. Ukoliko je na to parkirno mjesto i odabire. U tom trenutku se događaju dvije stvari. Ažurira se tablica na operaterskoj stranici koja prikazuje koja su točno mjesta zauzeta, a koja slobodna te se potom isto to radi i u bazi podataka. Nakon što je promjena upisana u bazu podataka korisnička web stranica će te nove podatke „povući“ te potom ažurirati slobodna mjesta.



Slika 2.1.:*Use case dijagram*

2.3. Korištene tehnologije

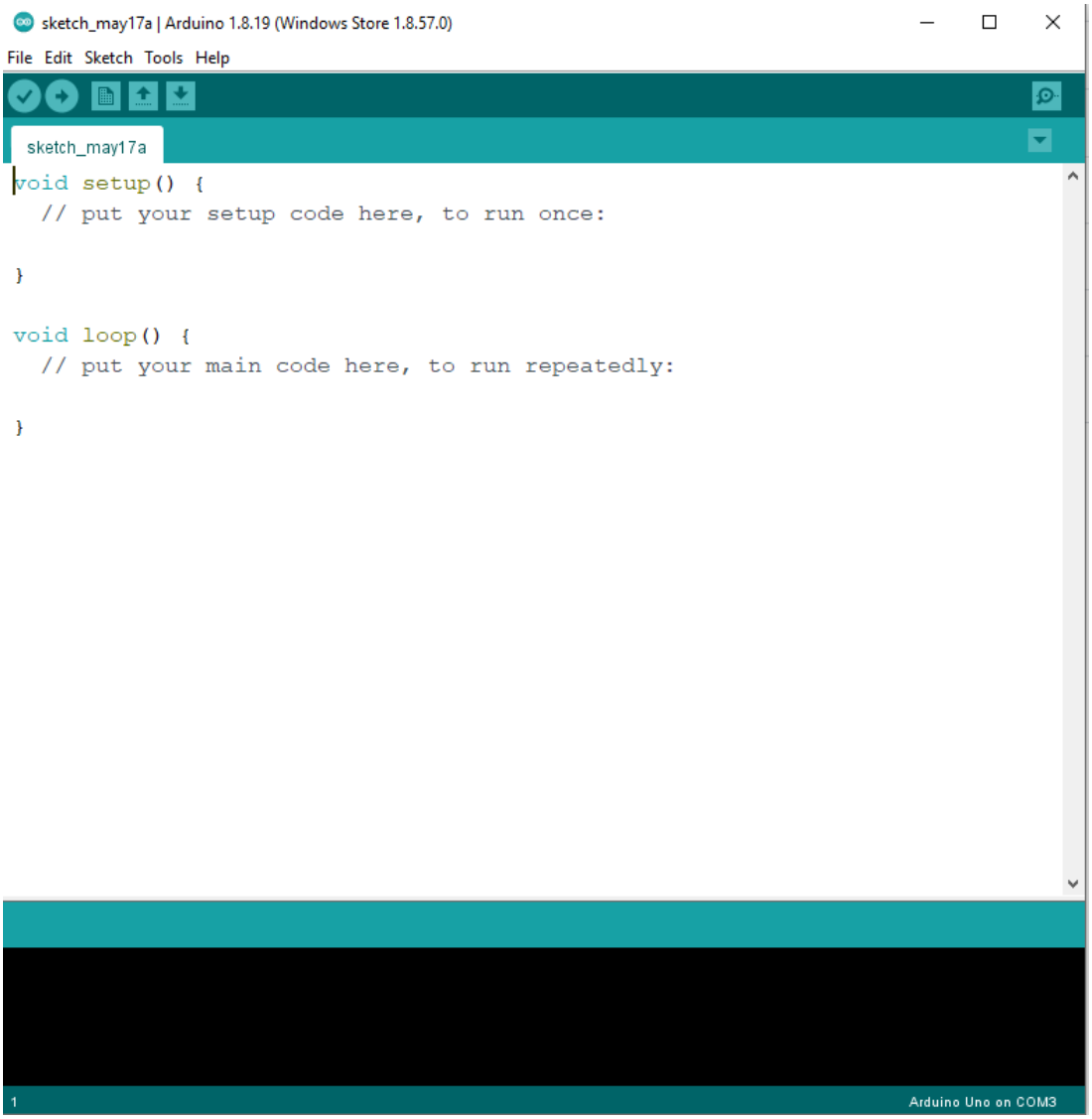
Prilikom preciziranja rješenja problema praćenja broja zauzetih mjesta na parkiralištu došlo se do zaključka da je najefikasnije rješenje vizualni prikaz parkirnih mjesta koji će biti omogućen preko kamere. Prvotna ideja uključivala je privatno parkiralište gdje bi praćenje broja zauzetih mjesta bilo obavljeno pomoću RFID čitača kartice koji bi prilikom prolaska automobila kroz rampu slao informacije o identitetu vlasnika vozila te vremenu njegova dolaska. Uz to bi se u bazi povećavao broj zauzetih mjesta. Prilikom izlaska vozila s parkirališta ponovno bi se isti podaci slali u bazu, a broj zauzetih mjesta bi se smanjivao. Od navedenog pristupa ipak se odustalo jer se došlo do zaključka da će novo rješenje imati šire područje primjene te je jednostavnije za implementaciju. Umjesto RFID čitača odabran je ESP32 CAM modul čija je slika trenutnog stanja zauzetosti parkirališta prikazana na jednoj web stranici te operater ukoliko uoči izmjenu situacije na parkiralištu može promijeniti stanje zauzetosti parkirališta klikom na parkirno mjesto prikazano na slici, a na drugoj web stranici je prikaz parkirališta preko kojeg korisnik može vidjeti koja su točno parkirna mjesta slobodna, a koja zauzeta. Za razvoj web stranica koristio se Visual Studio Code koristeći HTML, CSS, JS i PHP programske jezike, a za programiranje ESP32 CAM modula se koristio Arduino IDE.

2.3.1. Visual Studio Code

Visaul Studio Code je besplatni editor programskog koda razvijen od strane *Microsoft*-a. Služi za razvoj web stranica, mobilnih programskih izvedbi, razvoj računalnih igrica i sl. Nudi mogućnost otklanjanja pogrešaka (engl. *debugging*), refaktoriranje koda, inteligentno dovršavanje koda (engl. *IntelliSense*), korištenje git naredbi za verzioniranje iz samog editora itd. Podržava različite programske jezike kao što su: Python, C3, C/C++, JavaScript, TypeScript, HTML/CSS, Java, PHP i mnogi drugi.

2.3.2. Arduino IDE

Arduino IDE program je otvorenog koda, napisan u programskom jeziku Java, a koristi se kako bi se kôd napisan u programskom jeziku C ili C++ preveo u strojni jezik i poslao procesoru. Program napisan za Arduino naziva se skica (eng. *Sketch*). Svaka Arduino skica ima minimalno dvije funkcije: *setup()* i *loop()*. Na slici 2.2. prikazan je prozor Arduino IDE-a.



***Slika 2.2.**: Prazna Arduino skica*

Unutar *setup()* funkcije se inicijaliziraju varijable, pin modovi i sl. Kôd koji se napiše unutar ove funkcije se izvršava jednom i to odmah nakon pokretanja sustava.

Loop() funkcija se poziva nakon funkcije *Setup()* funkcije i ponavalja se sve dok se sustav ne ugasi. Ovo predstavlja „glavni“ dio kôda unutar kojeg se pišu same naredbe koje želimo da se izvrše.

Osim ove dvije funkcije, u skici se mogu nalaziti i druge, tzv. korisnički definirane funkcije koje se koriste za bolju preglednost kôda ili pojednostavljenje istog.

2.4. Troškovnik

Naziv komponente	**Količina**	**Cijena**
ESP32-CAM	1	105,29 kn
Kabel USB2.0 USB-A (M) na Micro USB-B (M), pleteni	1	15,00 kn
Arduino paket spojnih žica 150mm, žensko-muški	1	15,00 kn

Tablica 2.1.*: Troškovi korištenih komponenti*

Većina komponenti bila je u privatnom vlasništvu studenta prije izvođenja projekta pa su cijene komponenti usklađene s trenutnim cijenama na tržištu. U tablici 2.1 navedene su sve komponente koje će se koristiti, njihova cijena i količina.

3. REALIZACIJA SUSTAVA

3.1. Kôd za ESP32-CAM

Arduino kôd je preuzet od autora Rui Santosa [2] koji je baziran oko ESP32-CAM modula. Potrebno je instalirati ESP32 pločicu u Arduino IDE kako bi se uopće moglo povezati sa modulom.

U samome kodu poterbno je definirati koji model ESP32-CAM modula, u ovom projektu korišten je *AI thinker* te je iz tog razloga on i definiran.

```
#define CAMERA_MODEL_AI_THINKER
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WITHOUT_PSRAM

// Not tested with this model
//#define CAMERA_MODEL_WROVER_KIT
```

Slika 3.1.: *Definiranju ESP32-CAM modula*

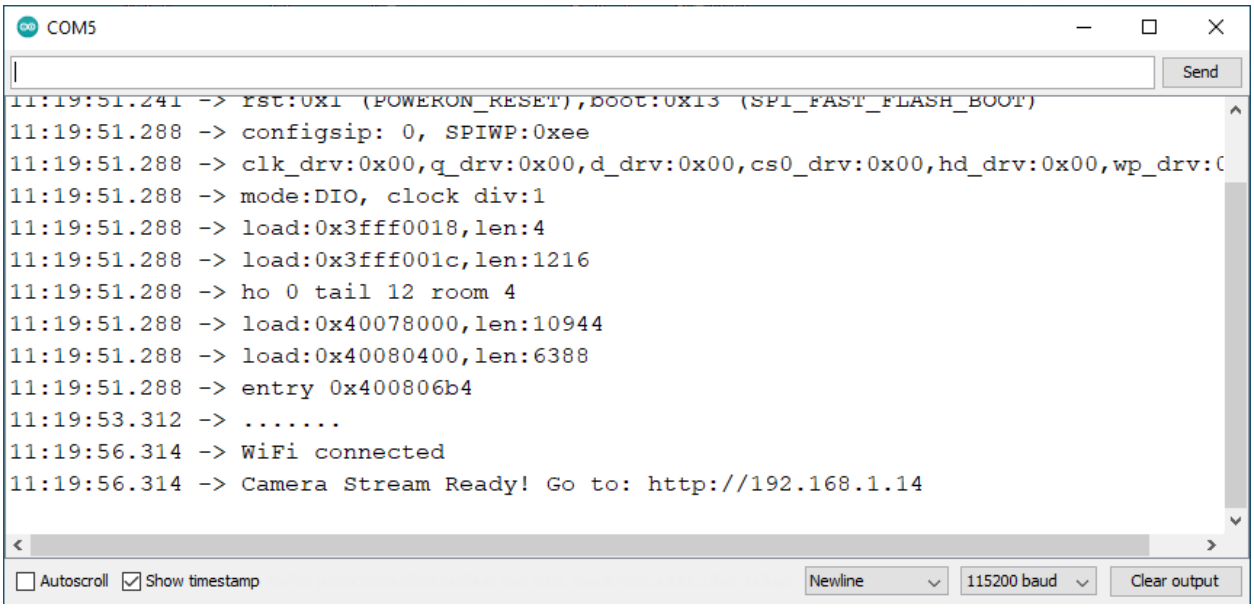
Postavljanje kôda na samu pločicu je moguće pomoću FTDI programera. Modul koji je kupljan u sklopu ovog projekta je imao FTDI programator koji se postavlja na isti način na koji se *shield*-ovi postavljaju na običnu Arduino pločicu.

Slika 3.2.: *ESP32-CAM modul sa FTDI programatorom*

Nadalje je potrebno upisati naziv i lozinku WIFI mreže na koju se modul spaja i definirati IP adresu kao što je prikazano na slici 3.3. Na slici 3.4 prikazan je *serial monitior* nakon uspješnog povezivanja modula sa mrežom unutar kojeg je moguće vidjeti IP adresu modula. Adresa je statična i uvijek će biti ista, 192.168.140.30, odlaskom na tu IP adresu vidimo prikaz same kamere što je prikazano na slici 3.5..

```
//Replace with your network credentials
const char* ssid = "[redacted]";
const char* password = "[redacted]";

// Set your Static IP address
IPAddress local_IP(192, 168, 140, 93);
// Set your Gateway IP address
IPAddress gateway(192, 168, 140, 13);

IPAddress subnet(255, 255, 255, 0);
```

Slika 3.3.: *Definiranje WiFi podataka, IP adrese, gateway-a i subnet maske*



Slika 3.4.: *Serial monitor sa IP adresom modula*

Slika 3.5.: *Prikaz slike s kamere u web pregledniku*

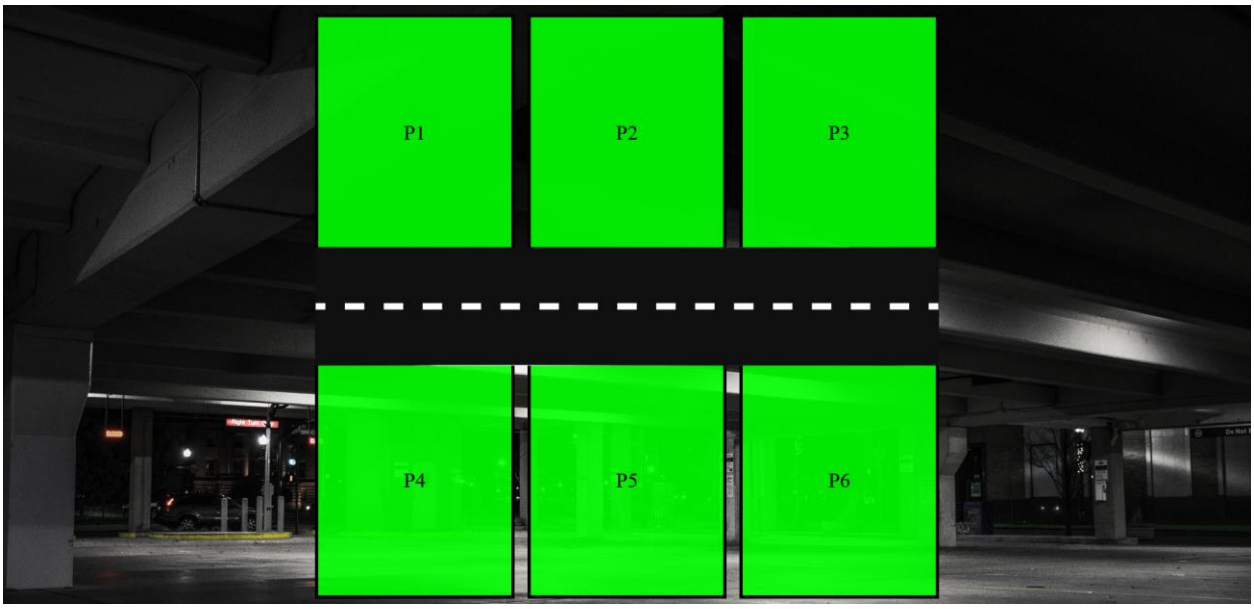
Rezolucija slike je 640x480 piksela, razlog ovakvoj maloj rezoluciji je što ukoliko se stavi veća rezolucija trebati će puno više vremena da se slika prikaže. Maksimalna moguća rezolucija ovog modula je 2560x1920 piksela.

```
if(psramFound()){
  //FRAMESIZE_XGA; //1024x768
  //FRAMESIZE_UXGA; // 1600x1200
  //FRAMESIZE_SVGA;  // 800x600

  config.frame_size =FRAMESIZE_VGA;  // 640x480
  config.jpeg_quality = 10;
  config.fb_count = 2;
} else {
  config.frame_size = FRAMESIZE_SVGA;
  config.jpeg_quality = 12;
  config.fb_count = 1;
}
```

Slika 3.6.: *Rezolucija ESP32-CAM modula*

3.2. Korisnička web stranica

Korisnička stranica je zamišljena da se nalazi ispred samog ulaza na parkiralište kako bi korisnici prilikom ulaza imali uvid u slobodna parkirna mjesta.



Slika 3.7. : *Prikaz korisničke web stranice*

```
const PARKING_SPOT_STATUS = [];

// get data from db on page load
$(document).ready(function() {
    $.ajax({
        url: "getDatabaseData.php",
        type: "POST",
        success: function(jsonList) {
            data = JSON.parse(jsonList);

            for (i = 0; i < data.length; i++) {
                if (data[i] == true) {
                    PARKING_SPOT_STATUS[i] = true;
                } else {
                    PARKING_SPOT_STATUS[i] = false;
                }
            }
            changeBackground();
        },
        async: false
    });
});
```

Slika 3.8. : *Prikaz kôda za učitavanje stranice*

Ova stranica ništa ne upisuje u bazu podataka već samo čita iz nje i to samo prilikom učitavanja iste. Napravljena je varijabla *PARKING_SPOT_STATUS* u koju se uspije stanje parkirnih mjesta te se kasnije u funkciji *changeBackground()* mijenjaju boje parkirnih mjesta, crvena znači zauzeto, a zelena znači slobodno.

```
function changeBackground() {
    var table = document.getElementById("parking-lot");

    console.log("changeBackgorund: " + PARKING_SPOT_STATUS);

    for (i = 0; i < PARKING_SPOT_STATUS.length; i++) {
        //Parking spot 1
        if (i == 0 && PARKING_SPOT_STATUS[i] == false) {
            table.rows[0].cells[0].classList.remove("occupied");
        } else if (i == 0 && PARKING_SPOT_STATUS[i] == true) {
            table.rows[0].cells[0].classList.add("occupied");
        }

        //Parking spot 2
        if (i == 1 && PARKING_SPOT_STATUS[i] == false) {
            table.rows[0].cells[1].classList.remove("occupied");
        } else if (i == 1 && PARKING_SPOT_STATUS[i] == true) {
            table.rows[0].cells[1].classList.add("occupied");
        }

        //Parking spot 3
        if (i == 2 && PARKING_SPOT_STATUS[i] == false) {
            table.rows[0].cells[2].classList.remove("occupied");
        } else if (i == 2 && PARKING_SPOT_STATUS[i] == true) {
            table.rows[0].cells[2].classList.add("occupied");
        }

        //Parking spot 4
        if (i == 3 && PARKING_SPOT_STATUS[i] == false) {
            table.rows[2].cells[0].classList.remove("occupied");
        } else if (i == 3 && PARKING_SPOT_STATUS[i] == true) {
            table.rows[2].cells[0].classList.add("occupied");
        }

        //Parking spot 5
        if (i == 4 && PARKING_SPOT_STATUS[i] == false) {
            table.rows[2].cells[1].classList.remove("occupied");
        } else if (i == 4 && PARKING_SPOT_STATUS[i] == true) {
            table.rows[2].cells[1].classList.add("occupied");
        }

        //Parking spot 6
        if (i == 5 && PARKING_SPOT_STATUS[i] == false) {
            table.rows[2].cells[2].classList.remove("occupied");
        } else if (i == 5 && PARKING_SPOT_STATUS[i] == true) {
            table.rows[2].cells[2].classList.add("occupied");
        }
    }
}
```

Slika 3.9. : *Prikaz funkcija changeBackground()*

Funkcija *changeBackground()* prolazi kroz polje *PARKING_SPOT_STATUS* te provjerava je li vrijednost *true* ili *false*. Ako je *true* to se može interpretirati kao da je mjesto zauzeto te se određenom mjestu dodaje klasa *occupied* koja mijenja boju pozadine u crvenu.

Komunikacija između korisničke i operaterske web stranice je omogućena korištenje *websocket*-a. Na korisničkoj web stranici napravljene su 3 funkcije koje služe za slanje/primanje podataka. Potrebno je implementirati *echo* poslužitelj (engl. *echo server*) koristeći Python programski jezik.

Server se kreira na TCP portu 5000, također se kreira *connected* varijabla pomoću *set()* metode koja pretvara niz ili listu u iterabilnu sekvencu u kojoj su svi podaci jedinstveni (nema duplikata).

Kreiranjem asinkrone funkcije *server()* koja prvobitno dodaje novu konekciju te onda čeka poruku. Ukoliko se primi poruka šalje se svima koji su trenutno „preplaćeni“ ne *WebSocket* poslužitelja, ali ne i osobi koja je tu poruku poslala.

```
# https://github.com/Vuka951/tutorial-code/blob/master/py-websockets/app.py
import asyncio
import websockets

connected = set()

async def server(websocket, path):
    # Register.
    connected.add(websocket)
    try:
        async for message in websocket:
            for conn in connected:
                if conn != websocket:
                    await conn.send(message)
    finally:
        # Unregister.
        connected.remove(websocket)


start_server = websockets.serve(server, "localhost", 5000)

asyncio.get_event_loop().run_until_complete(start_server)
asyncio.get_event_loop().run_forever()
```

Slika 3.10. : *WebSocket poslužitelj*

Nakon toga je potrebno na web stranici kreirati novi *WebSocket* objekt koristeći *WebSocket* konstruktor koji prima URL (engl. *Uniform Resource Locator*) na koji će *WebSocket* server odgovoriti.

```
const socket = new WebSocket('ws://localhost:5000');
```

Slika 3.11. : *Kreiranje WebSocket objekta*

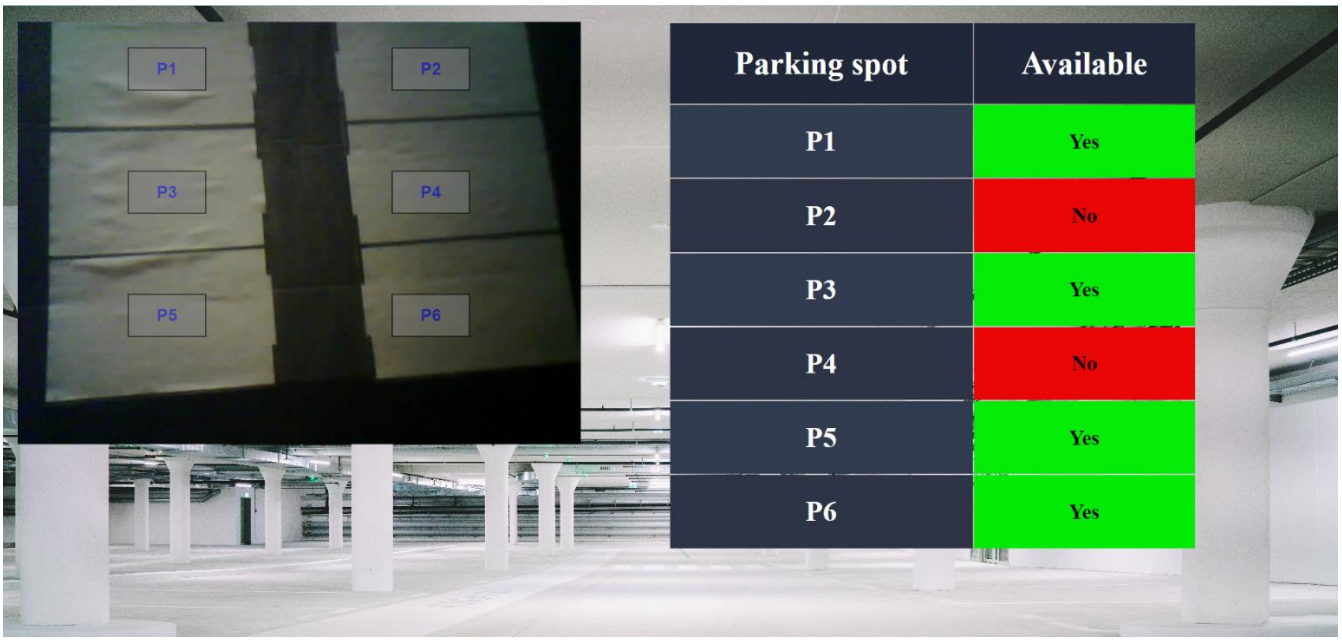
Nakon kreiranja objekta potrebno je implementirati funkcije koje omogućuju razmjenu podataka. Prva i druga funkcija nam samo ispisuju poruku ukoliko se uspješno povežemo ili odspojimo sa *WebSocket* servera. Treća funkcija predstavlja funkciju u kojoj se primaju podaci i obrađuju. Podaci se šalju u JSON (engl. *JavaScript Object Notation*) obliku te se kasnije prebacuju u *string* oblik podatka koji se onda pridodjeljuju *PARKING_SPOT_STATUS* polju podataka. Nakon što se prime i obrade podaci poziva se već spomenuta *changeBackgorund()* funkcija koja mijenja pozadinu parkirnih mjesta na web stranici.

```
// https://github.com/Vuka951/tutorial-code/blob/master/py-websockets/clients/index.html
// Connection opened
socket.addEventListener('open', function(event) {
    console.log('Connected to the WS Server!')
});

// Connection closed
socket.addEventListener('close', function(event) {
    console.log('Disconnected from the WS Server!')
});

// Listen for messages
socket.addEventListener('message', function(event) {
    //console.log('Message from server ', event.data);
    //store data from Operator website
    socketData = JSON.parse(event.data);

    for (i = 0; i < socketData.length; i++) {
        PARKING_SPOT_STATUS[i] = socketData[i];
    }

    changeBackground();
});
```

Slika 3.12. : *Implementirane WebSocket funkcije*

3.3. Operaterska web stranica

Operaterska stranica zamišljena je kao sučelje preko kojega je operateru omogućeno da uživo prati situaciju na parkiralištu te pritiskom na odgovarajući gumb označi parkirno mjesto kao zauzeto ili slobodno ovisno o situaciji. Pored live streama s parkirališta nalazi se tablica koja ažurno prikazuje stanje dostupnosti za svako parkirano mjesto.



***Slika 3.13**: Prikaz operaterske web stranice*

Prilikom učitavanja stranice najprije se učitavaju podaci iz baze podataka o trenutnom stanju zauzetosti parkirališta pomoću SQL upita napisanih u *getDatabaseData.php (Slika 3.14)* . Prilikom učitavanja podataka iz baze odmah se ažurira i tablica zauzetosti parkirnih mjesta te označava odgovarajućom bojom (*Slika 3.15*).

getDatabaseData.php > ...

```php
<?php
require_once 'connect.php';


if ($_SERVER['REQUEST_METHOD'] === "POST") {
    $sql = "SELECT * FROM smartparkinglot;";

    $query = $conn->prepare($sql);
    $query->execute();

    if ($query->rowCount() == 0) {
        echo ("Nothing like that in database.");
    } else {
        // convert from object to associative field
        $query->setFetchMode(PDO::FETCH_ASSOC);

        foreach ($query as $row) {
            // store parking spot state in array
            $parking_spot = array(
                $row["P1"],
                $row["P2"],
                $row["P3"],
                $row["P4"],
                $row["P5"],
                $row["P6"]
            );
        }

        unset($conn);
    }

    echo json_encode($parking_spot);
} else {
    echo ("Error!");
}
```

Slika 3.14: *getDatabaseData.php*

```
$(document).ready(function() {
        $.ajax({
            url: "getDatabaseData.php",
            type: "POST",
            success: function(jsonList) {
                data = JSON.parse(jsonList);
                //console.log(data);

                //copy database data to array
                for (i = 0; i < data.length; i++) {
                    index=i+1;
                    var y = document.getElementById(index);
                    if (data[i] == true) {
                        PARKING_SPOT_STATUS[i] = true;
                        y.classList.add("occupied");
                        y.innerHTML="No"

                    } else {
                        PARKING_SPOT_STATUS[i] = false;
                        y.classList.remove("occupied");
                        y.innerHTML="Yes"

                    }
                }
            },
            async: false
        });
    });
```

Slika 3.15: *Učitavanje podataka na stranici i ažuriranje tablice*

Prikaz streama kamere na operaterskoj stranici omogućen je korištenjem *iframe* HTML elementa koji omogućuje prikaz jedne web stranice unutar druge pri čemu je kao izvor navedena statična IP adresa koju smo prethodno dobili pomoću napisanog Arduino koda. Preko *iframea* su zatim „prevučeni“ gumbi za svako parkirno mjesto.

```
<div id="parent" class="grid-child-element">
<iframe src="http://192.168.135.10/"></iframe>
    <button id="P1" class="button">P1</button>
    <button id="P4" class="button">P4</button>
    <button id="P2" class="button">P2</button>
    <button id="P5" class="button">P5</button>
    <button id="P3" class="button">P3</button>
    <button id="P6" class="button">P6</button>
</div>
```

Slika 3.16: *Realizacija HTML prikaza streama kamere i gumba koji su prevučeni preko*

Prilikom klika na određeni gumb mijenja se vrijednost *PARKING_SPOT_STATUS(Slika 3.19)* na temelju kojeg se zatim ažurira vrijednost u bazi podataka korištenjem *ajax* upita (*Slika 3.16*) koji koristi SQL logiku napisanu u *UpdateDatabaseData.php (Slika 3.18)* te šalje podatke, a ujedno se ažurira i prikaz u tablici za to parkirno mjesto.

```
$.ajax({
    url: "UpdateDatabaseData.php",
    type: "POST",
    data: {
        spot: spot
    },
    success: function(msg) {
        console.log(msg);
    },
    async: false
```

Slika 3.17: *Slanje upita za upisivanje podataka u bazu*

```php
<?php

require_once 'connect.php';

if ($_SERVER['REQUEST_METHOD'] === "POST") {
    $parkingSpots = $_POST["spot"];

    $P1 = $parkingSpots[0];
    $P2 = $parkingSpots[1];
    $P3 = $parkingSpots[2];
    $P4 = $parkingSpots[3];
    $P5 = $parkingSpots[4];
    $P6 = $parkingSpots[5];

    $sql = "UPDATE smartparkinglot SET P1='$P1', P2='$P2', P3='$P3', P4='$P4', P5='$P5', P6='$P6' WHERE ID=1;";

    $query = $conn->prepare($sql);
    $query->execute();
    unset($conn);

    echo ("Success");
} else {
    echo ("Error!");
}
```

Slika 3.18: *UpdateDatabaseData.php*

Kao što je već spomenuto komunikacija između korisničke i operaterske web stranice je omogućena korištenje *websocket*-a. Na operaterskoj web stranici kao i na korisničkoj napravljene su 3 funkcije koje služe za slanje/primanje podataka. Potrebno je implementirati *echo* poslužitelj (engl. *echo server*) koristeći Python programski jezik.

```
// Change PARKING_SPOT_STATUS array on btn press
$("button").click(function(e) {
    e.preventDefault();
    if (this.id == 'P1') {
        console.log("Button P1 pressed");
        PARKING_SPOT_STATUS[0] = !PARKING_SPOT_STATUS[0];
    } else if (this.id == 'P2') {
        console.log("Button P2 pressed");
        PARKING_SPOT_STATUS[1] = !PARKING_SPOT_STATUS[1];
    } else if (this.id == 'P3') {
        console.log("Button P3 pressed");
        PARKING_SPOT_STATUS[2] = !PARKING_SPOT_STATUS[2];
    } else if (this.id == 'P4') {
        console.log("Button P4 pressed");
        PARKING_SPOT_STATUS[3] = !PARKING_SPOT_STATUS[3];
    } else if (this.id == 'P5') {
        console.log("Button P5 pressed");
        PARKING_SPOT_STATUS[4] = !PARKING_SPOT_STATUS[4];
    } else if (this.id == 'P6') {
        console.log("Button P6 pressed");
        PARKING_SPOT_STATUS[5] = !PARKING_SPOT_STATUS[5];
    }
    console.log(PARKING_SPOT_STATUS);
    sendMsg.apply(this, PARKING_SPOT_STATUS);
    for (i = 0; i < PARKING_SPOT_STATUS.length; i++) {
        index=i+1;
        var y = document.getElementById(index);
        if ( PARKING_SPOT_STATUS[i] == true) {
            spot[i] = 1;
            y.classList.add("occupied");
            y.innerHTML="No"

        } else {
            spot[i] = 0;
            y.classList.remove("occupied");
            y.innerHTML="Yes"
        }
    }
```

***Slika 3.19**: Funkcija za promjenu statusa parkirnog mjesta pritiskom na pojedini gumb*

4. UPUTE ZA KORIŠTENJE

Korisnik parkirališta otvaranjem stranice dobiva vizualni prikaz parkirališta gdje su zelenom bojom označena slobodna parkirna mjesta, a crvenom zauzeta parkirna mjesta te se u skladu s tim parkira gdje može. Operater parkirališta na svojoj stranici ima live stream situacije na parkiralištu te klikom na gumb na odgovarajućem parkirnom mjestu označava je li parkirno mjesto zauzeto ili ne pri čemu mu se ujedno i ažurira stanje zauzetosti parkirnog mjesta u tablici smještenoj pored streama s parkirališta. Od operatera se očekuje da redovito nadzire stanje na parkiralištu i pri svakoj promjeni zauzetosti parkirnog mjesta promjeni status u bazi podataka kako bi korisnik imao točan uvid koja su parkirna mjesta trenutno slobodna.

ZAKLJUČAK

Trenutno rješenje predstavlja pokaznu maketu sustava. Kamera koja se napaja pomoću USB priključak sa naponom od 5V nije dovoljna kako bi mogla sa zadovoljavajućom kvalitetom nadzirati parkirna mjesta. Kamera je zapravo prvi dio koje je potrebno unaprijediti ukoliko se ova maketa planira doista i implementirati. Nakon unaprjeđenja potrebno je voditi računa o načinu postavljanja kamere. Kameru je potrebno postaviti dovoljno visoko i pod određenim kutom kako bi se osiguralo da operater (ili nekakva umjetna inteligencija) može vidjeti ukoliko se promjena desi na samome parkiralištu. Jedan od problema koji se mora svladati sa pozicioniranjem kamere je da npr. jedan veliki auto (npr. džip) prekrije manji auto i time onemogući ažuriranje zauzeća mjesta od strane operatera.

Drugi problem koji se susreće kod samog sustava, a vezan je uz kameru su vanjski uvjeti. Noć, snijeg, kiša, vjetar i sl. mogu pomaknuti kameru i time utjecati na točnost očitanja samog sustava ili mogu prekriti kameru (npr. snijeg može „zatrpati“ ili zalediti leću kamere). Prilikom implementacija ovakvog sustava potrebno je uvesti dodatni senzor koji očitava je li noć te upaliti svijetla, koja se također moraju postaviti na parkiralište. Što se tiče drugih uvjeta kao što su snijeg i kiša tu je potrebno uvesti nekakav zaštitni pokrov koji će štiti samu kameru.

Nadalje veliku slabost ovog sustava predstavlja internet. Sustav je zamišljen sa dvije web stranice. Veza između te dvije web stranice je baza podataka, operaterska web stranica upisuje u bazu, a korisnička iz te iste baze čita podatke. Ukoliko dođe do prekida interneta podaci se neće moći upisivati i samim time korisnička web stranica neće moći prikazati koja su točno mjesta zauzeta, a koja slobodna.

Posljednji nedostatak ovog sustava možemo reći da je ljudski faktor. Predloženo rješenje uvelike ovisi o operateru. Ukoliko on ne ažurira slobodna mjesta sustav postaje beskoristan jer neće prikazivati točne informacije samom korisniku. Ovo je moguće riješiti uvođenjem umjetne inteligencije koja će konstantno provjeravati je li došlo do promjene i te promjene upisivati u bazu podataka, a koju i početku može „trenirati“ operater.

LITERATURA

[1] Lucid web aplikacija - https://lucid.app (15.5.2022.)

[2] ESP32-CAM Video Streaming Web Server (works with Home Assistant) - https://randomnerdtutorials.com/esp32-cam-video-streaming-web-server-camera-home-assistant/ (26.5.2022.)

[3] Visual Studio Code - https://code.visualstudio.com/ (15.5.2022.)

[4] WebSockets tutorial, Aymeric Augustin and contributors - https://websockets.readthedocs.io/en/stable/index.html (24.5.2022.)

[5] WebSocket API, MDN contributors - https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (24.5.2022.)

[6] WebSocket kod, Vuka951 - https://github.com/Vuka951/tutorial-code/blob/master/py-websockets/clients/index.html (24.5.2022.)