Internet aplikacija za evidenciju nazočnosti studenata pomoću RFID/NFC čitača iksica

Nedić, Josip

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni prijediplomski studij Računarstvo

INTERNET APLIKACIJA ZA EVIDENCIJU NAZOČNOSTI STUDENATA POMOĆU RFID/NFC ČITAČA IKSICA

Završni rad

Josip Nedić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Josip Nedić
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	R4686, 28.07.2021.
JMBAG:	0165091602
Mentor:	izv. prof. dr. sc. Ivan Aleksi
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Internet aplikacija za evidenciju nazočnosti studenata pomoću RFID/NFC čitača iksica
Znanstvena grana završnog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak završnog rada:	Temu rezervirao Josip Nedić. Potrebno je napraviti internet aplikaciju za evidenciju nazočnosti studenata u nastavi na temelju iksice. Aplikacija treba prikazivati sliku, ime, prezime, e-mail te ostale osnovne podatke o studentu. Potrebno je realizirati bazu podataka iz koje je moguće prikazati trenutni postotak nazočnosti. Potrebno je omogućiti izvoz baze u excel tablicu. Aplikaciju je potrebno testirati i dokumentirati rezultate testiranja.

Datum prijedloga ocjene završnog rada od strane mentora:	12.09.2024.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	25.09.2024.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	26.09.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 26.09.2024.

Ime i prezime Pristupnika:	Josip Nedić
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4686, 28.07.2021.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Internet aplikacija za evidenciju nazočnosti studenata pomoću RFID/NFC čitača iksica**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivan Aleksi

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

Na različitim fakultetima postoje različiti kriteriji za prisutnost na nastavi. Na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek (FERIT) studenti su obavezni prisustvovati najmanje 70 % predviđene nastave za svaki kolegij. U slučaju da student ne ispuni taj uvjet za određeni kolegij, mora ga ponovno upisati sljedeće akademske godine. Radi ispunjavanja ovog postotka, studenti često pribjegavaju pomoći kolega koji pohađaju ista predavanja, tako da oni u njihovo ime potpišu listu za evidenciju prisutnosti. Korištenje RFID (engl. Radio-Frequency Identification) tehnologije za evidenciju prisutnosti studenata može učinkovito spriječiti manipulaciju s potpisnim listama i osigurati točnu evidenciju prisustva. Ovaj završni rad bavi se izradom web aplikacije za praćenje prisutnosti studenata na nastavi, u kojoj svaki profesor može pratiti prisutnosti s obzirom na broj skeniranja studentske kartice putem RFID sustava. Svaki korisnik može pregledati podatke o prisutnosti i izostancima na temelju skeniranja kartice na ulazu u predavaonice. Prijava i spremanje podataka obavljaju se putem Supabase platforme. Za izradu aplikacije korišteni su moderni web razvojni alati, pri čemu je aplikacija razvijena u okviru ASP.NET razor, dok je za pozadinu korišten C#. Dizajn korisničkog sučelja definiran je s pomoću HTML-a (engl. HyperText Markup Language) i CSS-a (engl. Cascading Style Sheets). Za samo čitanje kartica korišten je RFID čitač kartica povezan je s Dasduino ESP32 ConnectPlus pločicom bežični prijenos podataka o skeniranju u stvarnom vremenu na backend sustav.

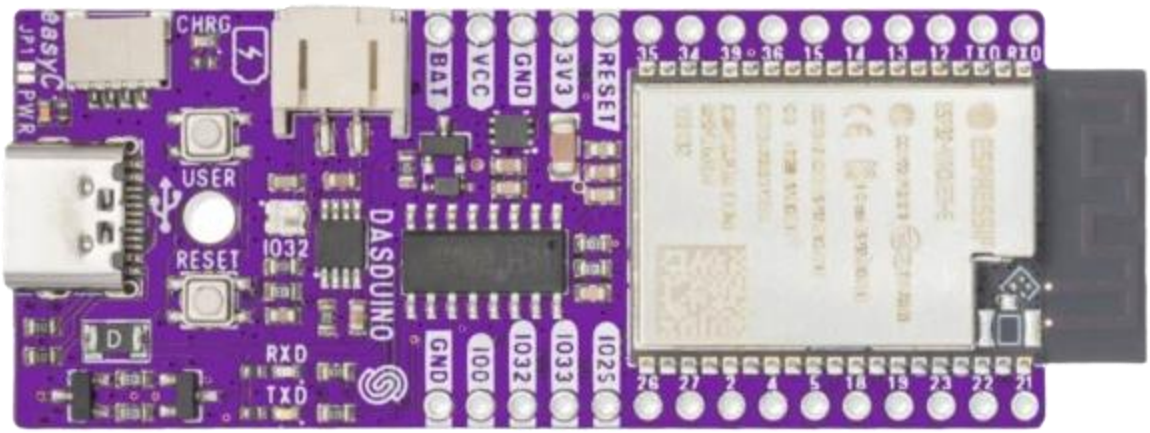
1.1 Zadatak završnog rada

Zadatak ovog završnog rada je izrada web aplikacije za praćenje prisutnosti studenata na nastavi koristeći RFID tehnologiju. Pomoću aplikacije profesor može pregledati podatke o studentskoj prisutnosti na predavanjima, s obzirom na broj skeniranja njihove studentske kartice prilikom ulaska u predavaonicu. RFID čitač kartica, povezan s Dasduino ESP32 ConnectPlus pločicom, prikuplja podatke o prisutnosti i bežično ih prenosi na pozadinu sustav aplikacije. Pozadina sustava je razvijena u C# .NET, dok je korisničko sučelje aplikacije izrađeno u Razoru, s dizajnom definiranim pomoću HTML-a i CSS-a. Podaci o korisnicima i njihovoj prisutnosti pohranjuju se na Supabase platformi. Glavni cilj rada je omogućiti studentima i nastavnicima jednostavan uvid u prisutnost na nastavi, uz pouzdano obilježavanje i pohranjivanje podataka korištenjem modernih tehnologija.

2. KORIŠTENE TEHNOLOGIJE

2.1 Dasduino ESP32 ConnectPlus pločica i PN532 RFID čitač kartica

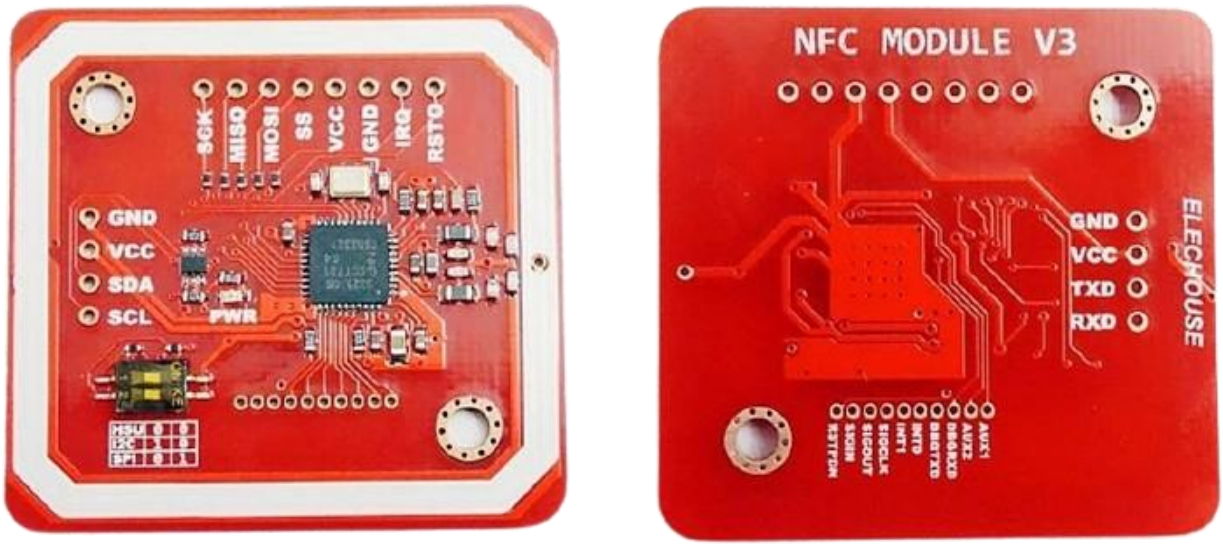
Dasduino ESP32 ConnectPlus pločica je razvojna platforma temeljena na popularnom ESP32 mikrokontroleru, koja nudi niz značajki pogodnih za projekte interneta stvari (engl. IoT) i bežičnu komunikaciju. ESP32 je snažan dual-core mikrokontroler koji integrira Wi-Fi i Bluetooth mogućnosti, omogućavajući jednostavnu implementaciju mrežnih aplikacija, kontrolu uređaja na daljinu, prikupljanje podataka i još mnogo toga. Dasduino ESP32 ConnectPlus dolazi s većinom potrebnih komponenti integriranih na pločicu, uključujući regulator napona, USB-to-serial čip te brojne GPIO (engl. general purpose input/output) pinove, koji omogućavaju povezivanje senzora, aktuatora i drugih perifernih uređaja. Ova pločica je kompatibilna s Arduino IDE-om (engl. Integrated Development Environment), što znači da je programiranje i razvoj softvera jednostavan, čak i za početnike. Korisnici mogu koristiti poznati Arduino programski jezik i biblioteku za razvijanje projekata. Jedna od ključnih prednosti ESP32 mikrokontrolera je njegova moćna obrada i mogućnost multitaskinga, što omogućava simultano izvođenje više zadataka. Pločica podržava Wi-Fi 802.11 b/g/n i Bluetooth LE (engl. Low Energy) 4.2, što je idealno za projekte koji zahtijevaju bežičnu povezivost, kao što su pametni kućni uređaji, senzorske mreže ili daljinski upravljani sustavi. Dasduino ESP32 ConnectPlus pločica također nudi širok raspon ulaznih i izlaznih opcija, uključujući PWM (engl. pulse-width modulation), ADC (engl. analog-to-digital converter), DAC (engl. digital-to-analog converter), I2C (engl. Inter-Integrated Circuit), SPI (engl. Serial Peripheral Interface) i UART (engl. Universal Asynchronous Receiver-Transmitter), što omogućava fleksibilnost u dizajnu hardvera. Tu su i dodatne značajke poput ugrađenog senzora za dodir, mogućnosti rada s baterijama i niske potrošnje energije, što je korisno za projekte koji trebaju raditi dulje vrijeme bez stalnog izvora napajanja. Ukratko, Dasduino ESP32 ConnectPlus pločica je svestrana i moćna razvojna platforma koja omogućava korisnicima da lako razvijaju i implementiraju raznovrsne IoT projekte i aplikacije. S obzirom na svoju bogatu funkcionalnost i kompatibilnost s Arduino ekosustavom, predstavlja odličan izbor za hobiste, studente i profesionalce koji žele istražiti mogućnosti ESP32 mikrokontrolera (slika 2.1).



Slika 2.1 Dasduino ESP32 ConnectPlus pločica

Izvor: https://soldered.com/product/dasduino-connectplus/

Posljednjih godina postupci automatske identifikacije (Auto-ID) postali su vrlo popularni u mnogim poslovima kao što su logistika nabave i distribucije, industrija, proizvodna poduzeća i sustavi protoka materijala[1]. PN532 RFID čitač kartica je popularan modul koji se koristi za komunikaciju s NFC (engl. Near Field Communication) i RFID karticama. Ovaj čitač temelji se na NXP-ovom PN532 čipu, koji podržava različite komunikacijske protokole, uključujući ISO/IEC 14443 za kartice tipa A i B te FeliCa standard. PN532 modul može raditi u nekoliko različitih načina, uključujući kao RFID/NFC čitač, pisac ili peer-to-peer uređaj. Ovaj modul omogućuje lako očitavanje informacija s RFID kartica ili tagova te njihovo slanje prema mikroprocesoru, što je korisno za aplikacije kao što su kontrole pristupa, identifikacija, praćenje inventara i slično. PN532 podržava različite metode komunikacije s mikrokontrolerima, uključujući I2C, SPI i UART, što ga čini fleksibilnim za integraciju u različite projekte. Često se koristi u kombinaciji s mikrokontrolerima kao što su Arduino, ESP32, Raspberry Pi i drugi. Jedna od prednosti PN532 modula je njegova mogućnost rada na maloj udaljenosti (obično do 10 cm), što ga čini sigurnim za aplikacije koje zahtijevaju blisku interakciju između čitača i kartice. Modul također podržava mogućnost emulacije NFC tagova, što omogućava da mikrokontroler s kojim je povezan djeluje kao NFC uređaj. Uz jednostavan dizajn i bogatu dokumentaciju, PN532 RFID čitač kartica je popularan izbor za izradu raznih projekata u IoT području, kao što su sustavi za beskontaktno plaćanje, pametni identifikacijski sustavi i interaktivne instalacije (slika 2.2).



Slika 2.2 PN532 RFID čitač kartica

Izvor: https://www.esp8266.com/viewtopic.php?p=83391

2.2 Arduino IDE razvojno sučelje

Arduino IDE je službeno razvojno okruženje za programiranje Arduino pločica i kompatibilnih mikrokontrolera. Ova aplikacija pruža jednostavno sučelje koje omogućava korisnicima da pišu, uređuju i učitavaju kod na Arduino uređaje. Arduino IDE koristi pojednostavljeni verziju C++ programskog jezika, što olakšava početnicima da brzo nauče i započnu razvijati projekte. IDE dolazi s integriranim editorom koda, serijskim monitorom za praćenje podataka i velikom bibliotekom gotovih funkcija koje olakšavaju rad sa senzorima, motorima, LE diodama i drugim komponentama. Arduino IDE također omogućava lako upravljanje bibliotekama i dodavanje novih, čime se proširuje funkcionalnost i omogućava rad s raznovrsnim hardverskim modulima. Osim toga, podržava rad s različitim pločicama iz Arduino ekosustava i kompatibilnim mikrokontrolerima te nudi opciju spajanja dodatnih alata za debugging i profiliranje koda. Sve ove značajke čine Arduino IDE moćnim, a istovremeno pristupačnim alatom za razvoj raznovrsnih elektroničkih i IoT projekata.

2.3 Razor

U ASP.NET Core MVC aplikaciji koristi se komponenta nazvana "view engine" za izradu sadržaja koji se šalje klijentima. Zadani "view engine" naziva se Razor i obrađuje HTML datoteke s oznakama za upute koje umetnu dinamički sadržaj u izlaz koji se šalje pregledniku[2]. Razor je šablonski pokretač razvijen od strane Microsofta za korištenje unutar ASP.NET okvira, koji omogućava

stvaranje dinamičkih web stranica. Glavna karakteristika Razor-a je njegova sposobnost da integrira C# kod unutar HTML stranica pomoću jednostavne i intuitivne sintakse. Prebacivanje između HTML-a i C# koda postiže se korištenjem @ znaka, što omogućava programerima lako uključivanje dinamičkog sadržaja, logike i podataka unutar statičkog HTML-a. Razor je posebno popularan zbog svoje čitljivosti i jednostavnosti. U usporedbi s drugim okvirnim jezicima, kao što su JSP (engl. JavaServer Pages) ili PHP(engl. Hypertext Preprocessor). Razor nudi jasniju i kraću sintaksu, što smanjuje mogućnost pogrešaka prilikom pisanja koda. Primjerice, u C# dijelu koriste se petlje, uvjetne izjave, pozivanje metode ili prikazivanje podataka direktno iz backend-a. Razor se koristi u nekoliko tehnologija unutar ASP.NET-a, uključujući Razor Pages i Blazor. Razor Pages pruža pojednostavljen način za izradu web aplikacija, gdje svaka stranica može imati svoj backend (pozadinski) kod, što čini razvoj bržim i lakšim. Blazor, s druge strane, omogućava korištenje Razor sintakse na klijentskoj strani pomoću WebAssembly tehnologije, što donosi prednosti brzine i učinkovitosti bez potrebe za pisanjem JavaScript-a. Razor se izvršava na strani servera (engl. server-side rendering), generira gotove HTML stranice koje se šalju klijentu. Ovo je izuzetno korisno za SEO optimizaciju, brže učitavanje stranica i sigurnost, budući da korisnik ne vidi C# kod. Ukratko, Razor kombinira najbolje od oba svijeta: moć server-side programiranja i jednostavnost kreiranja dinamičkih HTML stranica, što ga čini jednim od najpopularnijih alata za izradu modernih web aplikacija unutar ASP.NET okvira.

2.4 C# i Visual Studio

C# je moderan, objektno orijentirani programski jezik razvijen od strane Microsofta. Prvotno je predstavljen 2000. godine kao dio .NET platforme, a danas je jedan od najpopularnijih jezika za razvoj aplikacija, posebno u okruženju Windowsa. C# je dizajniran da bude jednostavan za učenje, ali istovremeno dovoljno moćan za razvoj kompleksnih aplikacija, uključujući desktop, web, mobilne aplikacije i igre. Jedna od glavnih značajki jezika je njegova integracija s .NET okvirom, što omogućuje jednostavan pristup širokom spektru biblioteka i alata koji olakšavaju razvoj aplikacija. C# je jezik tipiziran na vrijeme kompajliranja, što znači da većinu pogrešaka možete otkriti prije nego što se aplikacija pokrene. Podržava nasljeđivanje, enkapsulaciju i polimorfizam, ključne principe objektno orijentiranog programiranja. Također, C# nudi napredne funkcionalnosti poput lambda izraza, LINQ (engl. Language Integrated Query) za manipulaciju podacima i asinkronog programiranja, što ga čini vrlo fleksibilnim. Visual Studio je integrirano razvojno okruženje (engl. IDE) također razvijeno od strane Microsofta, koje se koristi za razvoj softverskih rješenja, posebno u

jezicima poput C#, VB.NET, F# i drugih. Visual Studio nudi veliki skup alata za pisanje, testiranje i debugiranje aplikacija te je posebno moćan za razvoj na .NET platformi. Visual Studio ima brojne značajke, poput inteligentnog automatskog dovršavanja koda (IntelliSense), vizualnog uređivanja sučelja i integriranih alata za upravljanje bazama podataka. Jedna od ključnih prednosti Visual Studia je njegova podrška za razvoj aplikacija za više platformi. Osim aplikacija za Windows, moguće je razvijati aplikacije za Android, iOS, Linux pa čak i igre koristeći Unity integraciju. Visual Studio podržava sve glavne programske jezike na .NET platformi, a zahvaljujući alatima za kolaboraciju, omogućuje timovima da rade zajedno na razvoju složenih projekata. U Visual Studiju je također ugrađen napredni debugger, koji omogućuje praćenje i rješavanje grešaka u kodu tijekom njegovog izvršavanja. Osim toga, Visual Studio podržava alat za kontrolu verzija poput Git-a i Azure DevOps-a, što olakšava upravljanje kodom i verzijama unutar timova. Uz bogatu ekosferu proširenja, korisnici mogu prilagoditi Visual Studio svojim specifičnim potrebama. C# i Visual Studio zajedno čine vrlo moćan alat za razvoj softverskih rješenja. C# omogućuje pisanje optimiziranog, modernog koda, dok Visual Studio pruža sve potrebne alate za upravljanje projektima, testiranje i distribuciju aplikacija. Kroz asinkrono programiranje, napredne funkcije poput paralelizma te jednostavan rad s bazama podataka putem ORM-a kao što je Entity Framework, C# se koristi za razvoj širokog spektra aplikacija, dok Visual Studio čini cijeli proces razvoja bržim i efikasnijim.

2.5 HTML i CSS

HTML (engl. HyperText Markup Language) je osnovni jezik za izgradnju web stranica koji strukturira sadržaj i organizira informacije u obliku koji web preglednici mogu prikazati. Koristi oznake (engl. tagove) kao što su ‘ ’, ‘’, ‘<h1>’, ‘<p>’, ‘<a>’ i ‘<img>’ za definiranje različitih vrsta sadržaja i njihovih odnosa. HTML pruža smislenu strukturu stranica, omogućujući programerima i dizajnerima da organiziraju tekst, slike, veze, forme i druge elemente na način koji je razumljiv i lako dostupan korisnicima.

CSS (engl. Cascading Style Sheets) se koristi za oblikovanje i dizajniranje HTML elemenata, omogućavajući kontrolu nad njihovim izgledom i rasporedom. CSS definira kako HTML elementi izgledaju na stranici, uključujući boje, fontove, veličine, razmake, poravnanje i raspored elemenata. Stilovi se mogu primijeniti na cijelu web stranicu kroz vanjske CSS datoteke, koje se povezuju s HTML-om putem ‘<link>’ oznaka, čime se olakšava upravljanje stilovima i osigurava dosljednost dizajna. CSS omogućava stvaranje prilagodljivog dizajna koji se automatski prilagođavaju različitim

veličinama ekrana i uređajima putem medijskih upita (engl. media queries). Na primjer, možete definirati različite stilove za desktop računala, tablete i mobilne telefone zbog izgleda i optimalne funkcionalnosti web stranica na svim uređajima. Osim osnovnog oblikovanja, CSS također podržava napredne značajke poput animacija i prijelaza, koje mogu dodati dinamičnost i vizualni interes na web stranici. U kombinaciji, HTML i CSS omogućavaju stvaranje vizualno privlačnih i funkcionalnih web stranica koje su jednostavne za navigaciju i upotrebu. HTML pruža strukturu, dok CSS upravlja izgledom i stilom, čime se omogućava razvoj modernih i estetski ugodnih web aplikacija.

2.6 Supabase database

Supabase je open-source alternativa za Firebase, dizajnirana za izgradnju kompletnih aplikacija s real-time funkcionalnostima i skalabilnom infrastrukturom. U srcu Supabase-a nalazi se PostgreSQL baza podataka, koja omogućava napredne SQL funkcije, relacijske podatke i bogat skup alata za upravljanje podacima. Za razliku od Firebasea, koji koristi NoSQL pristup, Supabase koristi relacijski model baze podataka, što olakšava rad s kompleksnijim upitima i povezanim podacima. Supabase dolazi s real-time mogućnostima, gdje aplikacije mogu dobivati ažuriranja u stvarnom vremenu prilikom promjena u bazi podataka. Pored toga, ima ugrađen REST API koji se automatski generira na temelju vaše baze podataka, omogućujući jednostavnu interakciju s podacima kroz HTTP zahtjeve. Jedna od ključnih značajki je ugrađeni sustav autentifikacije, koji podržava različite metode prijave korisnika, poput e-maila, lozinke, OAuth (Google, GitHub) i drugih. Također, Supabase pruža integrirane alate za autorizaciju, što znači da možete kontrolirati pristup podacima na temelju korisničkih uloga. Supabase se lako integrira s frontend tehnologijama poput Reacta, Vuea ili Angulara, kao i s backend okruženjima poput Node.jsa ili C#-a. Korisnici ga često biraju zbog jednostavnosti, pristupačne cijene i snažne zajednice koja doprinosi njegovom razvoju.

3. RAZVOJ APLIKACIJE

Kako bi se uspješno razvila aplikacija za praćenje prisutnosti na nastavi uz pomoć RFID čitača kartica, ključno je unaprijed definirati izgled aplikacije i njezine funkcionalnosti. Prvi korak u procesu je razvoj backend sustava pomoću C# i .NET. Ovaj backend sustav upravlja podacima koji se prikupljaju s RFID čitača, uključujući jedinstvene ID-ove kartica i vrijeme skeniranja. Nakon što RFID čitač detektira karticu, podaci se šalju serveru, gdje se obrađuju i pohranjuju u Supabase bazu podataka. Supabase pruža skalabilnu PostgreSQL bazu podataka koja pohranjuje sve podatke o prisutnosti studenata i skeniranjima kartica. Ova baza omogućuje učinkovito upravljanje i pretraživanje podataka, kao i pružanje real-time ažuriranja. Na frontend strani, koristi se Razor u ASP.NET za izradu web stranica koje omogućuju korisnicima pregled i upravljanje podacima o prisutnosti. Razor omogućuje integraciju C# koda s HTML-om, stvarajući dinamičke i interaktivne stranice koje se automatski ažuriraju u skladu s promjenama u bazi podataka. Korisnici mogu vidjeti trenutne podatke o prisutnosti i izostancima na jednostavan i intuitivan način. Sveukupno, ovaj pristup omogućava besprijekorno praćenje prisutnosti na nastavi, uz precizno upravljanje podacima kroz snažan backend sustav i funkcionalno korisničko sučelje.

3.1 Povezivanje pločice sa web aplikacijom

Program započinje povezivanjem pločice na WiFi mrežu pomoću postavljenih SSID i lozinke. ‘WiFi.begin(ssid, password)’ inicijalizira pokušaj povezivanja s WiFi mrežom. Pločica provjerava status veze i čeka dok se ne poveže ili dok ne istekne vremensko ograničenje od 30 sekundi (programski kod 3.1).

```
void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);  // Set WiFi to station mode (client mode)
  WiFi.begin(ssid, password);

  unsigned long startTime = millis();
  unsigned long timeout = 30000; // 30 seconds timeout

  // Wait for the connection to be established or timeout
  while (WiFi.status() != WL_CONNECTED && millis() - startTime < timeout) {
    delay(1000);
    Serial.print("Connecting to WiFi: ");
    Serial.println(ssid);  // Print SSID name
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("Connected to WiFi");
  } else {
    Serial.println("Failed to connect to WiFi");
```

Programski kod 3.1 Povezivanje ESP32 pločice na WiFi

ESP32 provjerava da li je povezan s Wi-Fi mrežom. Ako je povezan, može slati podatke na server. ESP32 koristi HTTPClient biblioteku kako bi kreirao HTTP POST zahtjev. Postavlja URL servera na koji šalje podatke. Kreira se JSON format koji sadrži UID kartice, a taj podatak se šalje na web server. Podaci se šalju pomoću funkcije POST(). Nakon slanja podataka, ESP32 prima odgovor servera (ako ga ima) i prikazuje ga putem serijskog monitora (programski kod 3.2).

```
#include <HTTPClient.h>

void sendCardData(String uidStr) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    http.begin(serverUrl);
    http.addHeader("Content-Type", "application/json");

    // Create JSON payload
    String jsonPayload = "{\"uid\": \"" + uidStr + "\"}";

    int httpResponseCode = http.POST(jsonPayload);

    if (httpResponseCode > 0) {
      String response = http.getString();
      Serial.println("Response: " + response);
    } else {
      Serial.println("Error on sending POST: " + String(httpResponseCode));
    }

    http.end();
  } else {
    Serial.println("WiFi not connected. Unable to send data.");
  }
}
```

Programski kod 3.2 Slanje informacija na web stranicu

RFID čitač PN532 treba inicijalizirati na početku kako bi bio spreman za komunikaciju s ESP32 pločicom. To se radi pomoću funkcije ‘nfc.begin()’ koja pokreće komunikaciju između ESP32 i RFID čitača. Pomoću ‘getFirmwareVersion()’ se provjerava radi li čitač ispravno. Nakon toga šalje upit RFID čitaču kako bi se dobio njegov firmware. Ako čitač nije pronađen, program se zaustavlja. Nakon što je uspješno pronađen i inicijaliziran, RFID čitač se postavlja u SAM konfiguracijski način rada

(engl. Secure Access Module) kako bi mogao detektirati RFID kartice. Funkcija nfc.SAMConfig() omogućava čitaču da radi u pasivnom načinu rada gdje čeka na skeniranje kartice. Pločica ulazi u petlju gdje stalno provjerava postoji li kartica u blizini čitača. Funkcija readPassiveTargetID() pokušava očitati karticu u blizini koristeći ISO14443A protokol (koji koriste mnoge MIFARE kartice). Ako je kartica detektirana, funkcija vraća uspješan rezultat i UID kartice (jedinstveni identifikator). uid[ ] je polje koje sadrži očitani jedinstveni identifikator kartice. uidLength pokazuje koliko je dug UID (obično između 4 i 7 bajtova). Nakon što je kartica očitana, UID se pretvara iz bajtova u heksadecimalni oblik i sprema u string, kako bi se mogao prikazati ili poslati na server (programski kod 3.3).

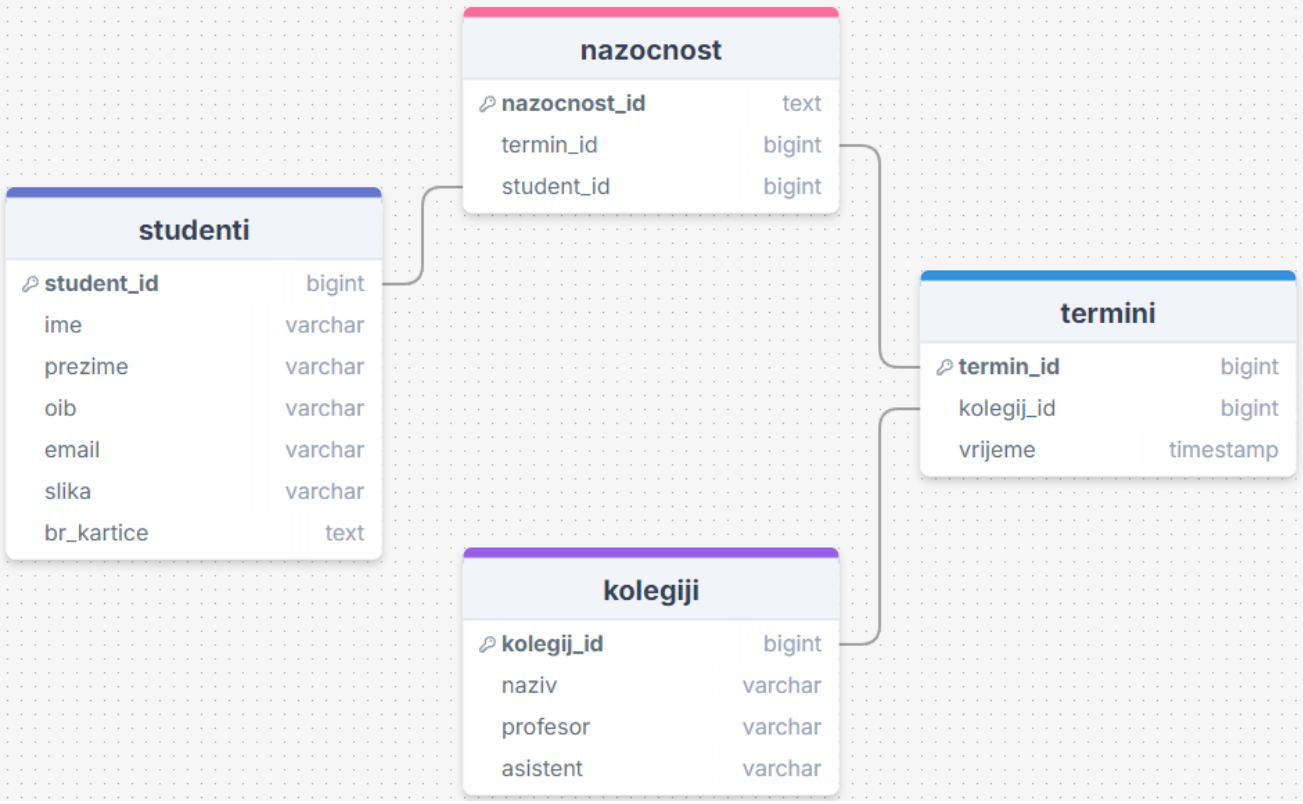
```
#define SDA_PIN 21
#define SCL_PIN 22

// Create an instance of the PN532
Adafruit_PN532 nfc(SDA_PIN, SCL_PIN);

void setup() {
  Serial.begin(115200);
  // Initialize WiFi connection as described above

  nfc.begin();
  uint32_t versiondata = nfc.getFirmwareVersion();
  if (!versiondata) {
    Serial.print("Didn't find PN53x board");
    while (1);
  }

  nfc.SAMConfig();
  Serial.println("Waiting for an NFC card...");
}

void loop() {
  uint8_t success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };
  uint8_t uidLength;

  // Wait for an RFID card
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength);

  if (success) {
    Serial.println("RFID card detected!");

    // Convert UID to a string
    String uidStr = "";
    for (uint8_t i = 0; i < uidLength; i++) {
      uidStr += String(uid[i], HEX);
    }

    // Send HTTP POST request to server with card UID
    sendCardData(uidStr);
```

Programski kod 3.3 Pokretanje RFID čitača

3.2 Izgradnja web aplikacije

3.2.1 Struktura baze podataka

Relacijski model je niži model. Koristi zbirku tablica za predstavljanje podataka i odnosa među tim podacima. Njegova konceptualna jednostavnost dovela je do njegove široke primjene; danas je velika većina baza podataka zasnovana na relacijskom modelu[3]. U do sada navedenom se spominje da je korišten Supabase jer omogućuje jednostavnu integraciju s bazom podataka i upravljanje podacima putem RESTful API-ja, što je idealno za aplikacije poput praćenja prisutnosti. Također, Supabase nudi real-time mogućnosti, što je korisno za praćenje i ažuriranje podataka u stvarnom vremenu, poput evidencije skeniranja RFID kartica (slika 3.1).



Slika 3.1 Dijagram baze podataka

3.2.2 Dohvaćanje skeniranih podataka

Iz prethodnog Arduino koda (programski kod 3.2) se može vidjeti kako se šalju podaci na web aplikaciju, u ovom slučaju su se podaci slali na krajnju točku RFID/scan.

'ScanRfid' funkcija služi za obradu RFID skeniranja studenata i evidentiranje njihove prisutnosti na nastavi. Prvo provjerava da li je zahtjev ispravan, odnosno sadrži li valjani UID kartice i ispravni identifikator dvorane. Ako te informacije nedostaju ili su neispravne, vraća poruku o grešci. Zatim se traži student u bazi podataka prema broju kartice (engl. UID). Ako student nije pronađen, vraća se poruka da student ne postoji. Nakon toga, provjerava se postoji li termin nastave koji je započeo unutar posljednjih 15 minuta i još traje i to u dvorani navedenoj u zahtjevu. Ako takav termin ne postoji, vraća se poruka da termin nije pronađen. Ako postoji termin, provjerava se da li je već zabilježena prisutnost za tog studenta na tom terminu. Ako već postoji zapis, vraća se poruka o grešci, koja govori da je prisutnost već evidentirana. Ako ne postoji zapis, kreira se novi zapis prisutnosti u bazi podataka, s podacima o studentu, terminu i trenutnom vremenu skeniranja. Na kraju, ako je sve uspješno, vraća se poruka da je zapis uspješno unesen (programski kod 3.4).

```
public async Task<IActionResult> ScanRfid([FromBody] RfidRequest request)
{
    if (request == null || string.IsNullOrEmpty(request.Uid) || request.DvoranaId <= 0)
        return BadRequest("Request is not valid");
    var student = await _supabaseClient
        .From<Student>()
        .Where(x => x.BrKartice == request.Uid)
        .Single();
    if (student is null)
        return NotFound(new { error = "Student not found" });
    var now = DateTime.UtcNow;
    var startTime = now.AddMinutes(-15);
    var endTime = now;
    var termin = await _supabaseClient
        .From<Termin>()
        .Where(x => x.StartTime >= startTime && x.StartTime <= endTime)
        .Where(x => x.EndTime >= now)
        .Where(x => x.DvoranaId == request.DvoranaId)
        .Single();
    if (termin is null)
        return NotFound(new { error = "No termin found within the last 15 minutes" });
    // Check if the record already exists for the given student and termin
    var existingAttendance = await _supabaseClient
        .From<Nazocnost>()
        .Where(x => x.StudentId == student.StudentId && x.TerminId == termin.TerminId)
        .Single();
    if (existingAttendance != null)
    {
        return BadRequest(new { error = "Attendance record already exists for this student and termin." });
    }
    await _supabaseClient
        .From<Nazocnost>()
        .Insert(new Nazocnost
        {
            TerminId = termin.TerminId,
            StudentId = student.StudentId,
            DateScanned = DateTime.Now,
        });
    return Ok(new { message = "Record inserted successfully" });
```

Programski kod 3.4 API za praćenje skenirane kartice i upisivanje prisutnosti u bazu

3.2.3 Spremanje podataka u excel

‘ExportAttendanceByKolegij’ omogućava izvoz podataka o prisutnosti studenata za određeni kolegij u Excel datoteku. Prvo dohvaća podatke o prisutnosti studenata putem funkcije ‘FetchAttendanceAsync’, a zatim koristi knjižnicu ClosedXML za izradu Excel radnog lista s kolonama za ime, prezime, OIB i postotak prisustva. Ako je OIB nepoznat, unosi se "N/A". Excel datoteka se sprema u memorijski tok i vraća kao preuzimanje putem HTTP odgovora, s nazivom "attendance.xlsx" (programski kod 3.5).

```
// GET /kolegij/{kolegijId}/export
[HttpGet("kolegij/{kolegijId:int}/export")]
0 references
public async Task<IActionResult> ExportAttendanceByKolegijId(int kolegijId)
{
    var attendanceData = await FetchAttendanceAsyncByKolegijId(kolegijId);
    using var workbook = new XLWorkbook();
    var worksheet = workbook.Worksheets.Add("Attendance");

    worksheet.Cell(1, 1).Value = "Ime";
    worksheet.Cell(1, 2).Value = "Prezime";
    worksheet.Cell(1, 3).Value = "OIB";
    worksheet.Cell(1, 4).Value = "Postotak Prisustva";

    int row = 2;
    foreach (var student in attendanceData)
    {
        worksheet.Cell(row, 1).Value = student.ime;
        worksheet.Cell(row, 2).Value = student.prezime;
        worksheet.Cell(row, 3).Value = string.IsNullOrEmpty(student.oib) ? "N/A" : student.oib.Trim();
        worksheet.Cell(row, 4).Value = $"{student.percentage}%";
        row++;
    }

    using var memoryStream = new MemoryStream();
    workbook.SaveAs(memoryStream);
    memoryStream.Seek(0, SeekOrigin.Begin);
    return File(memoryStream.ToArray(), "application/vnd.openxmlformats-officedocument.spreadsheetml.she
}
```

Programski kod 3.5 Kod za spremanje podataka u excel

Preuzimanje Excel datoteke koja sadrži podatke o prisutnosti studenata za određeni kolegij. Funkcija ‘ExportAttendance’ šalje zahtjev API-ju koristeći HTTP GET metodu kako bi preuzela Excel datoteku vezanu uz prisutnost studenata. Ako API uspješno vrati odgovor, podaci iz odgovora se pretvaraju u niz bajtova koji predstavljaju sadržaj datoteke. Ovaj niz bajtova se zatim pretvara u memorijski tok (engl. MemoryStream), koji se koristi za preuzimanje datoteke. Također, koristi se ime datoteke ‘Prisutnost_po_kolegiju.xlsx’ za pohranu. Nakon toga, funkcija ‘DownloadFile’ preuzima generiranu datoteku. Ova funkcija koristi .NET klasu ‘DotNetStreamReference’ kako bi tok podataka proslijedila

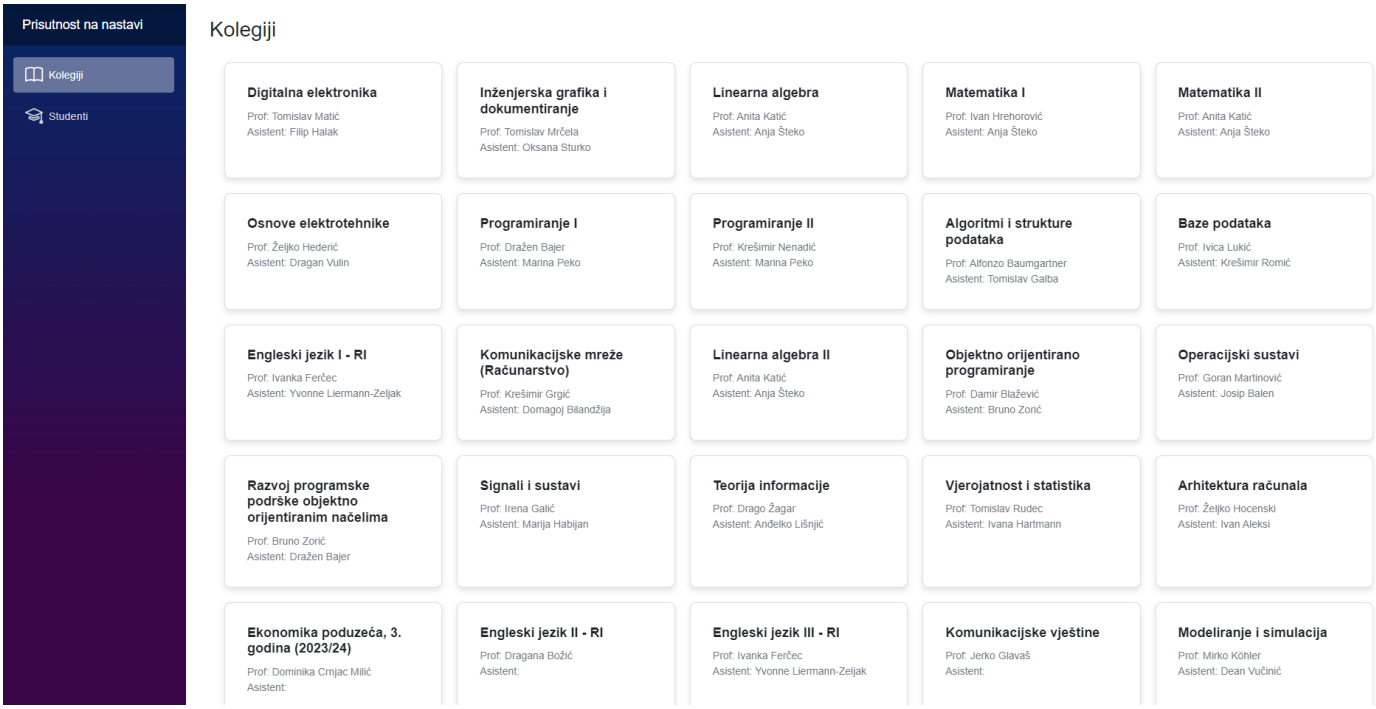
JavaScript kodu, gdje se koristi JavaScript interop da bi pokrenula preuzimanje datoteke u pregledniku. JavaScript funkcija 'downloadFileFromStream' se koristi za stvarno preuzimanje datoteke na klijentovom računalu. U slučaju da dođe do pogreške prilikom preuzimanja datoteke, ili prilikom izvođenja JavaScript koda, kod hvata iznimku (engl. Exception ili JSException) i prikazuje poruku o grešci u konzoli (programski kod 3.6).

```
private async Task ExportAttendance()
{
    try
    {
        // Call the API to download the attendance Excel file
        var response = await _httpClient.GetAsync($"api/Attendance/kolegij/{id}/export");

        if (response.IsSuccessStatusCode)
        {
            // Get the file content as a byte array
            var fileContent = await response.Content.ReadAsByteArrayAsync();
            var fileName = "Prisutnost_po_kolegiju.xlsx";

            // Log to check if file content is received
            Console.WriteLine("File content size: " + fileContent.Length);

            // Convert byte array to memory stream
            using var stream = new MemoryStream(fileContent);

            // Trigger the download of the file using JavaScript interop
            await DownloadFile(fileName, stream);
        }
        else
        {
            Console.WriteLine("Error exporting attendance: Response was not successful.");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error exporting attendance: {ex.Message}");
    }
}

private async Task DownloadFile(string fileName, Stream stream)
{
    try
    {
        // Use a DotNetStreamReference for passing the stream to JavaScript
        var streamReference = new DotNetStreamReference(stream);

        // Invoke the JavaScript function to download the file
        await JS.InvokeVoidAsync("downloadFileFromStream", fileName, streamReference);
    }
    catch (JSException jsEx)
    {
        Console.WriteLine($"JavaScript Interop Error: {jsEx.Message}");
```

Programski kod 3.6 Funkcija koju poziva tipka Spremi podatke

3.2.4 Početna stranica

Početna stranica treba dati pregled onoga što stranica nudi – i sadržaj ("Što mogu pronaći ovdje?") i funkcionalnosti ("Što mogu učiniti ovdje?") – te kako je sve organizirano. Ovo se obično rješava pomoću trajne navigacije[4]. Kada se dizajnira web aplikacija uvijek je potrebno imati neku vrstu navigacije preko koje korisnik može navigirati na različite stranice naše aplikacije. Za početnu stranicu napravljeno je da to ujedno bude i stranica s listom svih kolegija s obzirom da je to najbitnije u ovom slučaju da profesori mogu vidjeti prisutnost na bilo kojem kolegiju.



Slika 3.2 Izgled početne stranice

S lijeve strane se nalazi već spomenuta navigacija koja ima mogućnosti navigiranja do studenti stranice kao i do kolegiji stranice (slika 3.2).

HTML kod prikazuje navigacijski izbornik za web stranicu. U gornjem dijelu se nalazi naslov "Prisutnost na nastavi" unutar navbar klase. Ispod toga je skriveni navigacijski izbornik koji se može otvoriti klikom na checkbox s klasom navbar-toggler. Ovaj checkbox služi kao prekidač za prikaz ili sakrivanje izbornika. Navigacijski izbornik sadrži dvije stavke, svaka s ikonom i tekstom: "Kolegiji" (s ikonom knjige) i "Studenti" (s ikonom akademske kape). NavLink komponenta omogućava

navigaciju na različite stranice ili sekcije. Svi navedeni elementi koriste Bootstrap ikone i klase za stiliziranje, a ikone su prilagođene veličini putem klase fs-4 (programski kod 3.7).

```
<div class="top-row ps-3 navbar navbar-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="">Prisutnost na nastavi</a>
    </div>
</div>

<input type="checkbox" title="Navigation menu" class="navbar-toggler" />

<div class="nav-scrollable" onclick="document.querySelector('.navbar-toggler').click()">
    <nav class="flex-column">
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span class="bi bi-book fs-4" aria-hidden="true"></span> Kolegiji
            </NavLink>
        </div>

        <div class="nav-item px-3">
            <NavLink class="nav-link" href="studenti">
                <span class="bi bi-mortarboard fs-4" aria-hidden="true"></span> Studenti
            </NavLink>
        </div>
    </nav>
</div>
```

Programski kod 3.7 HTML za izgled navigacije

'OnInitializedAsync' metoda poziva se kada se komponenta inicijalizira. Koristi HttpClient za dohvat podataka o kolegijima iz API-ja i obrađuje eventualne greške. 'OnCardClick' metoda navigira na detaljnu stranicu kolegija kad se klikne na karticu kolegija (programski kod 3.8).

```
@code {
    private HttpClient? _httpClient;
    private List<Kolegij>? kolegiji;
    private bool isLoading = true;
    private bool hasError = false;

    protected override async Task OnInitializedAsync()
    {
        try
        {
            // Use the named HttpClient ("API") to call your API
            _httpClient = ClientFactory.CreateClient("API");

            // Fetch the kolegiji data from your API
            kolegiji = await _httpClient.GetFromJsonAsync<List<Kolegij>>("api/Kolegij");
        }
        catch (Exception ex)
        {
            hasError = true;
            Console.WriteLine($"Error fetching kolegiji: {ex.Message}");
        }
        finally
        {
            isLoading = false;
        }
    }

    // Navigate to the details page for the selected kolegij
    private void OnCardClick(Kolegij kolegij)
    {
        Navigation.NavigateTo($"/kolegij-detalji/{kolegij.KolegijId}");
    }
}
```

Programski kod 3.8 C# kod početne stranice

<PageTitle> postavlja naslov stranice na "Kolegiji". Stanje učitavanja i greške prikazuje poruku dok se podaci učitavaju ili ako dođe do greške. Ako su podaci uspješno dohvaćeni, prikazuje ih u obliku kartica s nazivom kolegija, imenom profesora i asistenta. Kartice su interaktivne i omogućuju navigaciju na detaljnu stranicu klikom (programski kod 3.9).

```
<PageTitle>Kolegiji</PageTitle>

<h3>Kolegiji</h3>

@if (isLoading)
{
    <p>Loading...</p>
}
else if (hasError)
{
    <p>Error loading kolegiji.</p>
}
else if (kolegiji is not null && kolegiji.Any())
{
    <div class="grid-container">
        @foreach (var kolegij in kolegiji)
        {
            <div class="card" @onclick="() => OnCardClick(kolegij)">
                <div class="card-body">
                    <h5 class="card-title">@kolegij.Naziv</h5>
                    <p class="card-text text-muted">
                        Prof: @kolegij.Profesor <br />
                        Asistent: @kolegij.Asistent
                    </p>
                </div>
            </div>
        }

    </div>
}
else
{
    <p>No kolegiji available.</p>
}
```

Programski kod 3.9 HTML kod početne stranice

CSS definira stilove za prikaz kartica i njihovih elemenata. ‘grid-container’ postavlja raspored kartica u grid s automatskim prilagođavanjem veličine kartica. ‘card’ stilizira kartice s pozadinom, sjenkom i prijelazima pri hover efektu. ‘card-body’ dodaje unutarnje razmake unutar kartica, dok ‘card-title’ i ‘card-text’ stiliziraju naslove i tekst unutar kartica. Ovi stilovi poboljšavaju vizualni dojam i interaktivnost kartica na stranici (programski kod 3.10).

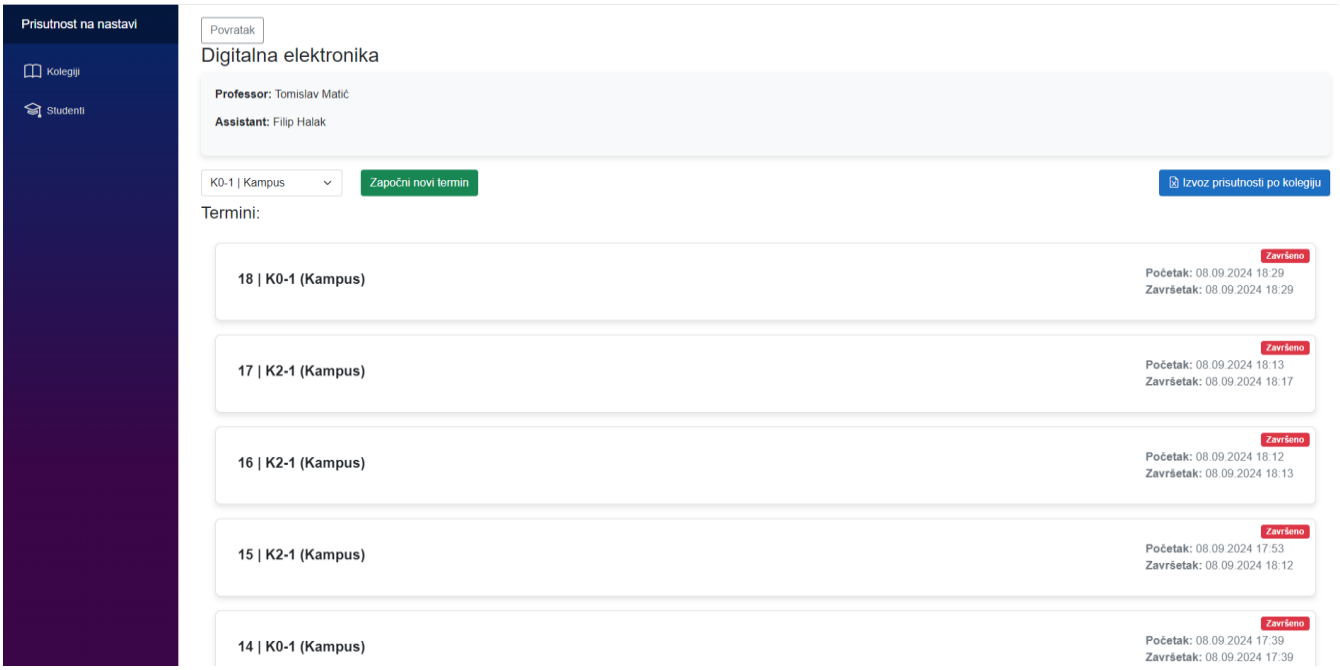
```
<style>
    .grid-container {
        display: grid;
        grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
        gap: 20px;
        padding: 20px;
    }

    .card {
        background-color: #fff;
        border-radius: 8px;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        transition: transform 0.2s ease, box-shadow 0.2s ease;
        cursor: pointer;
        padding: 15px;
        pointer-events: all;
    }

    .card:hover {
        transform: scale(1.05);
        box-shadow: 0 6px 12px rgba(0, 0, 0, 0.15);
    }

    .card-body {
        padding: 15px;
    }

    .card-title {
        font-size: 18px;
        font-weight: bold;
        margin-bottom: 10px;
    }

    .card-text {
        font-size: 14px;
        color: #6c757d; /* Bootstrap's muted text color */
    }

    .text-muted {
        color: #6c757d !important;
    }

</style>
```

Programski kod 3.10 CSS kod početne stranice

3.2.5 Detaljni prikaz kolegija

Samim klikom na jednu od kartica koje označavaju kolegij, aplikacija nas dovodi na detaljniji prikaz informacija o kolegiju koja sadrži listu termina sa statusom aktivnosti

Ova stranica ima više funkcionalnosti. S lijeve strane kao i na prethodnoj stranici korištena je ista navigacija. U gornjem dijelu stranice se nalaze osnovne informacije o kolegiju kao što su ime, profesor i asistent. Ispod toga se nalaze dvije tipke te jedan padajući izbornik za odabir dvorane u kojoj se nalazi predavanje. Ova funkcionalnost je nužna za praćenje aktivnosti s obzirom na to da u jednom trenutku može postojati više termina, na taj način nakon skeniranja kartice aplikacija zna na koji termin treba upisati studenta (slika 3.3).



Slika 3.3 Izgled stranice kolegij detalji

Metoda 'StartTermin' služi za kreiranje i spremanje novog termina u Supabase bazu podataka. Prvo, metoda koristi POST zahtjev kako bi primila podatke putem 'StartTerminRequest' objekta, koji sadrži informacije o ID-u kolegija i dvorane. U bloku try, metoda stvara novi objekt Termin. Početno vrijeme termina se postavlja na trenutni trenutak, dok se vrijeme završetka postavlja na sat i pol kasnije. Ovi podaci, zajedno s ID-om kolegija i dvorane, koriste se za definiranje novog termina. Nakon toga, metoda koristi Supabase klijent za umetanje novog termina u bazu podataka. Ako je umetanje uspješno i odgovor je pozitivnog statusa, metoda vraća poruku koja potvrđuje uspjeh. Ako dođe do pogreške prilikom umetanja, metoda čita sadržaj odgovora, ispisuje informacije o grešci na konzolu i vraća problem status s odgovarajućom porukom. U slučaju specifičnih iznimki poput PostgrestException, metoda pokušava čitati detalje greške iz odgovora kako bi pružila dodatne

informacije. Također, metoda obrađuje iznimku ObjectDisposedException ako dođe do neočekivane zatvorenosti objekta. Ako dođe do bilo koje druge iznimke, ispisuje generičku poruku greške i vraća status 500. Sve u svemu, metoda je dizajnirana za kreiranje novog termina, unos u bazu podataka i obrada mogućih grešaka (programski kod 3.11).

```
[HttpPost("start")]
0 references
public async Task<IActionResult> StartTermin(StartTerminRequest request)
{
    try
    {
        //Create a new Termin with the current date and time
        var newTermin = new Termin
        {
            KolegijId = request.KolegijId,
            StartTime = DateTime.Now,
            EndTime = DateTime.Now.AddHours(1.5),
            DvoranaId = request.DvoranaId
        };

        // Insert the new Termin into Supabase
        var response = await _supabaseClient.From<Termin>().Insert(newTermin);

        // Ensure response content is fully read before disposing the response object
        if (response.ResponseMessage.IsSuccessStatusCode)
        {
            return Ok(new { message = "Termin started successfully" });
        }
        else
        {
            var responseContent = await response.ResponseMessage.Content.ReadAsStringAsync();
            Console.WriteLine($"Error starting termin: {response.ResponseMessage.StatusCode}");
            Console.WriteLine($"Response Content: {responseContent}");
            return Problem("Error starting termin");
        }
    }

    catch (PostgrestException ex)
    {
        // Read the content from the response early to avoid disposed exceptions
        if (ex.Response != null && !ex.Response.Content.Headers.ContentLength.HasValue)
        {
            var errorContent = await ex.Response.Content.ReadAsStringAsync();
            Console.WriteLine($"Supabase Error Details: {errorContent}");
        }

        return StatusCode(500, new { error = ex.Message });
    }
    catch (ObjectDisposedException ex)
    {
        // Handle ObjectDisposedException explicitly
        Console.WriteLine($"Object disposed unexpectedly: {ex.Message}");
        return StatusCode(500, new { error = "Object disposed unexpectedly: " + ex.Message });
    }
    catch (Exception ex)
    {
        // General error logging
        Console.WriteLine($"Unexpected error: {ex.Message}");
        return StatusCode(500, new { error = ex.Message });
```

Programski kod 3.11 API za pokretanje termina

Metoda 'EndTermin' koristi HTTP PUT zahtjev za ažuriranje termina u bazi podataka. Na temelju varijable 'terminId', dohvaća termin, postavlja trenutno vrijeme kao vrijeme završetka (EndTime) i ažurira zapis u bazi. Ako termin ne postoji, vraća NotFound. Ako je uspješno ažuriran, vraća Ok sa ažuriranim informacijama o terminu (programski kod 3.12).

```
[HttpPut("{terminId:int}/end")]
0 references
public async Task<IActionResult> EndTermin(int terminId)
{
    var termin = await _supabaseClient
        .From<Termin>()
        .Where(x => x.TerminId == terminId)
        .Single();

    if (termin == null)
        return NotFound();

    termin.EndTime = DateTime.Now;

    await termin.Update<Termin>();

    return Ok(termin);
```

Programski kod 3.12 API za završavanje termina

Tipka ‘Spremi prisutnost’ sprema željene podatke u .xlsx format odnosno excel (slika 3.4). Ovaj proces je detaljnije opisan u podnaslovu Spremanje podataka u excel.

Ime	Prezime	OIB	Postotak Prisustva
Josip	Nedić	74392534885	66.67%
Marko	Novak	23456789012	0.00%
Ana	Kovač	34567890123	0.00%
Petar	Marić	45678901234	0.00%
Marija	Babić	56789012345	0.00%
Karlo	Šimić	67890123456	0.00%
Lana	Božić	78901234567	0.00%
Luka	Radić	89012345678	0.00%
Nina	Petrović	90123456789	0.00%
Ivan	Jurić	01234567890	0.00%
Juraj	Đurčević	12345678901	0.00%
Gabrijela	Džebić	56734445131	33.33%

Slika 3.4 Primjer excel tablice

Na svakoj stranici osim početne se nalazi gumb za povratak na prethodnu stranicu. (programski kod 3.13).

```
// Navigate back to the main page
private void GoBack()
{
    Navigation.NavigateTo("/");
}
```

Programski kod 3.13 Tipka povratak

Metoda ‘OnInitializedAsync’, koja se automatski poziva kada se komponenta učitava. Prvo se inicijalizira ‘HttpClient’ korištenjem ‘ClientFactory’ kako bi se kreirala instanca ‘HttpClient’ s nazivom "API". Zatim se pomoću tog klijenta poziva API da se dobiju detalji o kolegiju na temelju proslijeđenog ‘id’, a rezultati se pohranjuju u varijablu ‘kolegij’. Nakon toga, poziva se drugi API kako bi se dohvatili podaci o prisustvu studenata za odabrani kolegij, a rezultati se pohranjuju u ‘studentAttendance’. Ako se tijekom dohvaćanja podataka pojavi iznimka, varijabla ‘hasError’ se postavlja na ‘true’ i ispisuje se poruka greške u konzoli. Na kraju, bez obzira na ishod, varijabla ‘isLoading’ se postavlja na ‘false’, što signalizira da je proces učitavanja završen (programski kod 3.14).

```
protected override async Task OnInitializedAsync()
{
    try
    {
        _httpClient = ClientFactory.CreateClient("API");

        // Fetch kolegij details
        kolegij = await _httpClient.GetFromJsonAsync<Kolegij>($"api/Attendance/kolegiji/{id}");

        // Fetch students' attendance data for the selected kolegij
        studentAttendance = await _httpClient.GetFromJsonAsync<List<StudentAttendance>>($"api/A
    }
    catch (Exception ex)
    {
        hasError = true;
        Console.WriteLine($"Error fetching kolegij details or students: {ex.Message}");
    }
    finally
    {
        isLoading = false;
```

Programski kod 3.14 Pozivanje api metode za postotak nazočnosti

Metoda ‘GetAttendancePercentage” poziva funkciju ‘FetchAttendanceAsync’ i vraća samo ispravne podatke. (programski kod 3.15)

```
// GET /attendance-percentage/{kolegijId}
[HttpGet("attendance-percentage/{kolegijId:int}")]
0 references
public async Task<IActionResult> GetAttendancePercentage(int kolegijId)
{
    var attendanceData = await FetchAttendanceAsync(kolegijId);
    return Ok(attendanceData);
}
```

Programski kod 3.15 API za dohvaćanje prisustva na nastavi

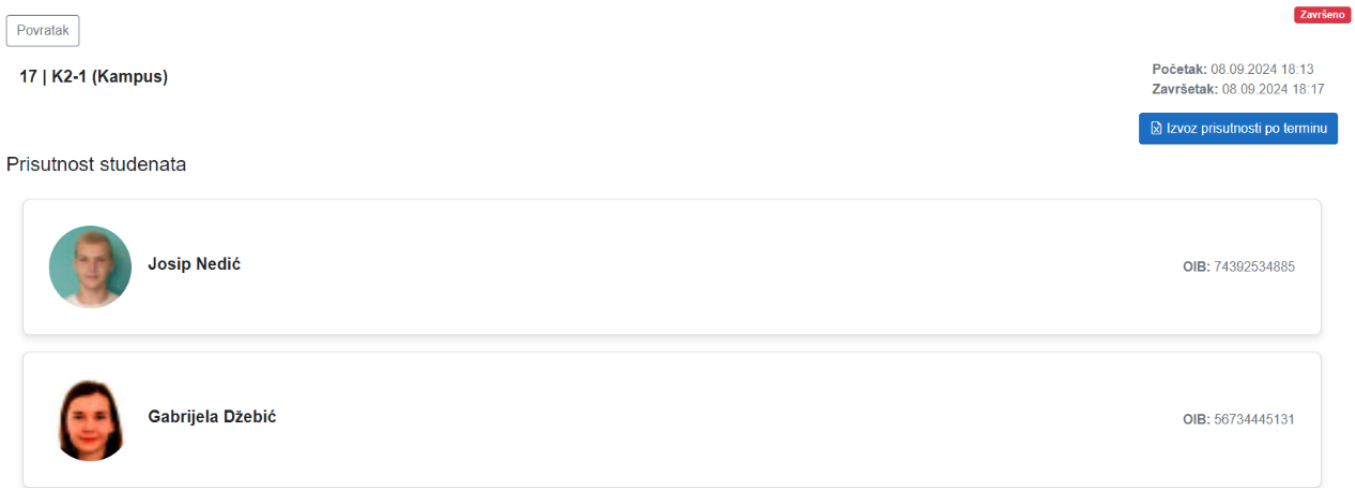
Metoda ‘FetchAttendanceAsync’ prikuplja podatke o prisustvu studenata za zadani kolegij pomoću varijable ‘kolegijId’. Ona prvo dohvaća ukupan broj termina za kolegij, zatim sve studente te zapise o prisustvu (nazočnosti) i termine specifične za kolegij. Nakon toga izračunava koliko je svaki student prisustvovao nastavi i postotak prisustva, vraćajući anonimne objekte s tim informacijama. Metoda ‘GetTotalTerminiForKolegijAsync’ dohvaća ukupan broj termina za određeni kolegij. Postupak uključuje pozivanje Supabase klijenta kako bi se dohvatili svi termini za zadanu varijablu ‘kolegijId’, provjerava se uspješnost odgovora i vraća broj termina. Metoda ‘GetAllStudentsAsync’ dohvaća sve studente iz baze podataka. Supabase klijent se poziva kako bi dohvatili svi studenti, a nakon provjere uspješnosti odgovora, vraća se lista studenata. Metoda ‘GetAttendanceRecordsAsync’ dohvaća sve zapise o prisustvu (nazočnosti). Supabase klijent se koristi za dohvaćanje svih zapisa o prisustvu, nakon čega se provjerava uspješnost odgovora i vraća lista zapisa o prisustvu. Na kraju, metoda ‘GetTerminiForKolegijAsync’ dohvaća sve termine specifične za zadani kolegij. Poziva se Supabase klijent kako bi se dohvatili termini za zadanu varijablu ‘kolegijId’, provjerava se uspješnost odgovora, a potom se vraća lista termina (programski kod 3.16).

```
private async Task<IEnumerable<dynamic>> FetchAttendanceAsync(int kolegijId)
        {
            var totalCount = await GetTotalTerminiForKolegijAsync(kolegijId);
            var students = await GetAllStudentsAsync();
            var nazocnosti = await GetAttendanceRecordsAsync();
            var termini = await GetTerminiForKolegijAsync(kolegijId);

            var result = students.Select(student =>
            {
                var attendedCount = nazocnosti
                    .Where(n => n.StudentId == student.StudentId && termini.Any(t => t.TerminId == n.TerminId))
                    .Count();

                var percentage = totalCount > 0 ? (attendedCount / (double)totalCount) * 100 : 0;

                return new
                {
                    student_id = student.StudentId,
                    ime = student.Ime,
                    prezime = student.Prezime,
                    oib = student.Oib,
                    slikaUrl = student.Slika, // Add the Slika (photo URL) field here
                    attended_count = attendedCount,
                    total_count = totalCount,
                    percentage = percentage.ToString("F2", CultureInfo.InvariantCulture)
                };
            });

            return result;
        }
```

Programski kod 3.16 FetchAttendance metoda

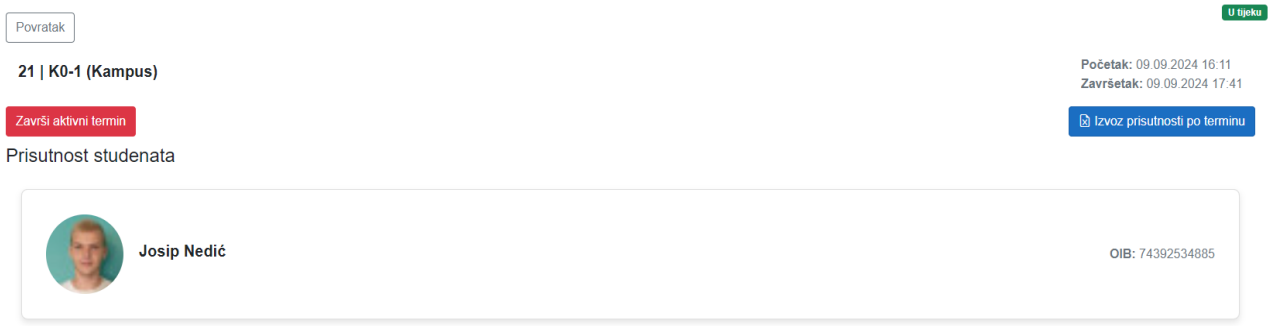
3.2.6 Prikaz prisutnosti na terminu

Pomoću ove stranice profesori imaju mogućnost uvida studenata koji su prisustvovali na određenom terminu te mogućnost preuzimanja popisa studenata koji su bili prisutni na tom terminu (slika 3.5).



Slika 3.5 Izgled stranice termina

Gore u desnom kutu se nalazi status aktivnosti, kada je termin u tijeku tada je status zelen i na njemu piše ‘U tijeku’. Također kada je termin u tijeku postoji mogućnost završavanja termina s te stranice (slika 3.6).



Slika 3.6 Izgled stranice kada je termin u tijeku

‘ExportAttendance’ metoda se malo razlikuje u odnosu na prošlu export metodu. Metoda ‘GetAttendanceByTerminId’ u backend API-ju dohvaća i vraća podatke o prisustvu za određeni termin na temelju varijable ‘terminId’. Ona vraća podatke o prisustvu u JSON format (programski kod 3.17).

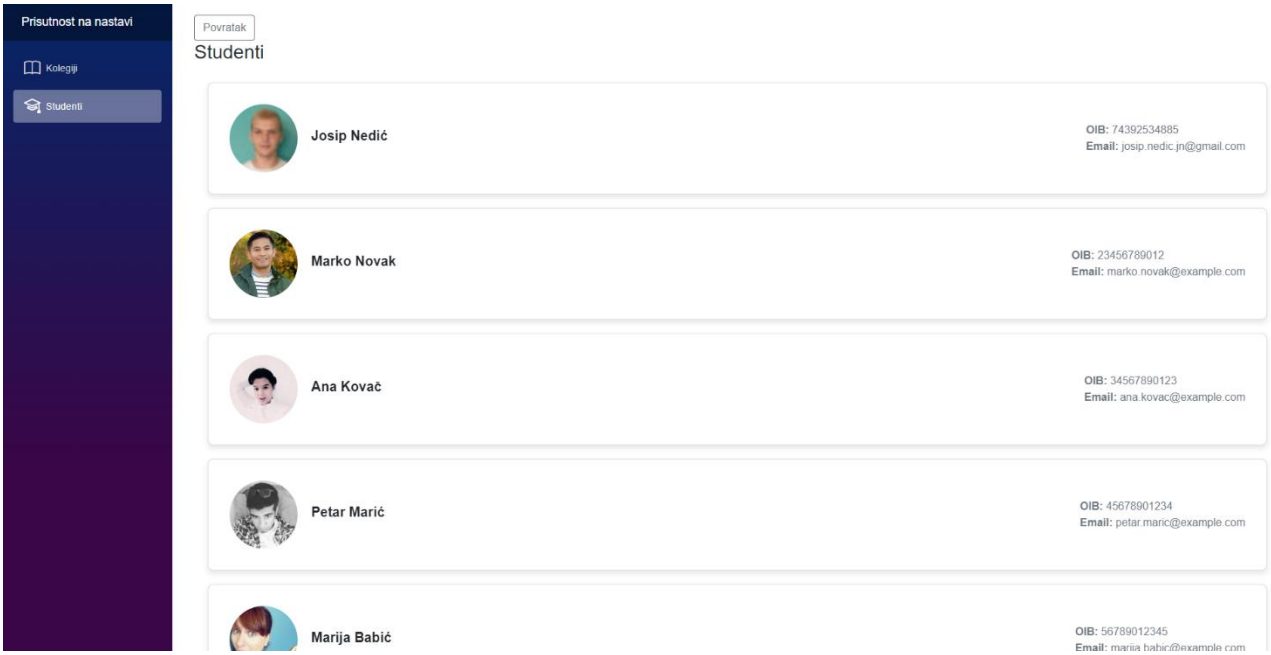
```
private async Task ExportAttendance()
{
    try
    {
        // Call the API to download the attendance Excel file
        var response = await _httpClient.GetAsync($"api/Attendance/termin/{id}

        if (response.IsSuccessStatusCode)
        {
            // Get the file content as a byte array
            var fileContent = await response.Content.ReadAsByteArrayAsync();
            var fileName = "Pristnost_po_terminu.xlsx";

            // Log to check if file content is received
            Console.WriteLine("File content size: " + fileContent.Length);

            // Convert byte array to memory stream
            using var stream = new MemoryStream(fileContent);

            // Trigger the download of the file using JavaScript interop
            await DownloadFile(fileName, stream);
        }
        else
        {
            Console.WriteLine("Error exporting attendance: Response was not su
        }
    }
```

Programski kod 3.17 Export po terminu

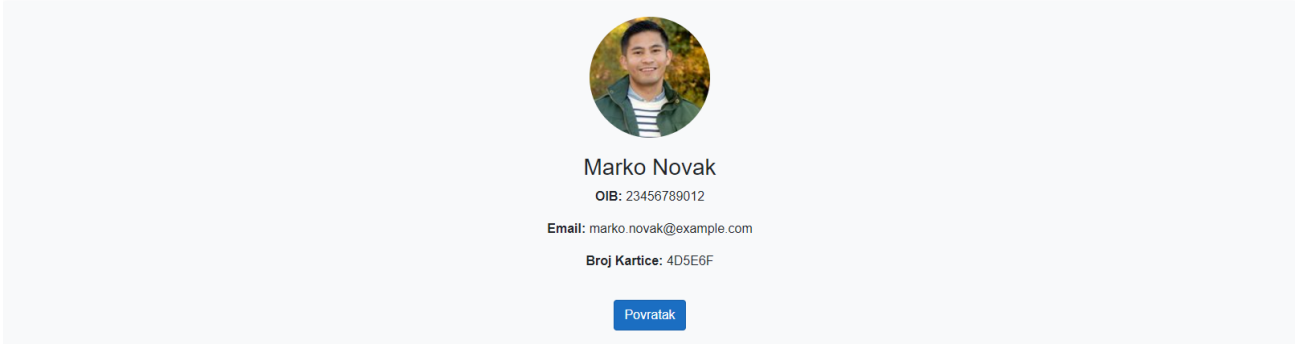
JSON je postao popularna alternativa XML-u. Iako mu nedostaju napredne značajke XML-a (poput imenskih prostora, prefiksa i shema), prednost mu je jednostavnost i preglednost, s formatom sličnim onome što biste dobili pretvaranjem JavaScript objekta u string[5].

3.2.7 Popis studenata

Na stranici student korisnici imaju uvid u bitne informacije o studentima (slika 3.7). Svaka studentska kartica se može kliknuti te otvara stranicu studentskog profila u kojemu se nalaze detaljnije informacije o studentu (slika 3.8).



Slika 3.7 Izgled Studenti stranice



Slika 3.8 Izgled studentskih detalja

Ovaj kod sadrži dva API-endpointa za rad s podacima o studentima. Prvi endpoint ([HttpGet("")]), 'GetAllStudents', koristi se za dohvaćanje svih studenata iz baze podataka. Kada se pozove, metoda šalje zahtjev prema Supabase klijentu za preuzimanje svih studenata. Ako je odgovor uspješan, vraća popis svih studenata, ako nije vraća statusni kod 500 s porukom o grešci. Ako dođe do iznimke tijekom dohvaćanja, metoda bilježi grešku i također vraća statusni kod 500 s porukom o grešci. Drugi endpoint ([HttpGet("{id:int}")]), 'GetStudent', koristi se za dohvaćanje podataka o pojedinačnom studentu na temelju njegovog identifikatora (id). Metoda šalje zahtjev prema Supabase klijentu za dohvaćanje podataka o studentu sa specifičnim studentskim identifikatorom. Ako student postoji, vraća podatke o njemu, ako student s tim identifikatorom ne postoji, vraća statusni kod 404 s porukom o grešci (programski kod 3.18).

```
[HttpGet("")]
0 references
public async Task<IActionResult> GetAllStudents()
{
    try
    {
        // Fetch all students from the "studenti" table
        var studentsResponse = await _supabaseClient.From<Student>().Get();

        // Check if the request was successful
        if (!studentsResponse.ResponseMessage.IsSuccessStatusCode)
            return StatusCode(500, new { error = "Error fetching students." });

        var students = studentsResponse.Models;

        // If no students found, return an empty list
        if (students == null || !students.Any())
            return Ok(new List<Student>());

        return Ok(students);
    }
    catch (Exception ex)
    {
        // Log the exception and return an error
        Console.WriteLine($"Error fetching students: {ex.Message}");
        return StatusCode(500, new { error = "An error occurred while fetching students." });
    }
}

// GET /student/{id}
[HttpGet("{id:int}")]
0 references
public async Task<IActionResult> GetStudent(int id)
{
    var student = await _supabaseClient.From<Student>().Where(x => x.StudentId == id).Single();

    if (student is null)
        return NotFound(new { error = "Student not found" });

    return Ok(student);
}
```

Programski kod 3.18 API za dohvaćanje studenata

4. ZAKLJUČAK

U ovom radu je napravljena aplikacija koja omogućava učinkovito upravljanje podacima o studentima i njihovom prisustvu na kolegijima. Pomoću aplikacije profesori mogu izraditi, pratiti i prekinuti termin za kolegije, kao i da završavaju termine kada je potrebno. Također, nudi funkcionalnost za izvoz podataka o prisustvu u Excel formatu, što omogućava jednostavnu izradu izvještaja o prisustvu studenata za pojedine termine ili kolegije. Uključene metode za dohvaćanje svih studenata ili pojedinačnih podataka o studentu omogućuju fleksibilan pristup informacijama. Aplikacija je dizajnirana s naglaskom na pouzdano rukovanje greškama, uključujući detaljno bilježenje i pružanje jasnih poruka o greškama. Sve ove značajke zajedno osiguravaju učinkovito praćenje i upravljanje podacima o prisustvu studenata, uz visoku razinu korisničke podrške. Trenutna verzija aplikacije je kreirana za profesore, ali dodatnim implementiranjem funkcionalnosti kao što su prijava u sustav i autentifikacija korisnika, aplikacija je pogodna i za studente koji žele uvid u svoje aktivnosti na fakultetu, Implementacijom ISSP REST API-ja koji sadrži metode kao što su dohvat studenta preko beskontaktnog broja kartice, dohvat termina predavanja (rasporeda) i druge metode koje koriste bazu podataka kojom se omogućuje korištenje aplikacije na fakultetima.

LITERATURA

[1] K. Finkenzeller, RFID Handbook: Principles and Applications, 2010.

[2] A. Freeman, Pro ASP.NET Core MVC 2, 2017.

[3] A. Silberschatz, H. Korth i S. Sudarshan, Database System Concepts-7th edition, 2019.

[4] S. Krug, Don't Make Me Think: A Common Sense Approach to Web Usability, 2014.

[5] J. Albahari i B. Albahari, C# 9.0 in a Nutshell: The Definitive Reference, 2021.

SAŽETAK

Cilj ove aplikacije bio je razviti web sustav za praćenje prisutnosti studenata na nastavi korištenjem RFID tehnologije. Pomoću aplikacije profesori mogu pratiti dolaske studenata na predavanja putem skeniranja RFID kartica u realnom vremenu. Web aplikacija koristi Supabase platformu za pohranu podataka, gdje se bilježe podaci o prisustvu, uključujući ime i prezime studenata te postotak prisutnosti na predavanjima. Izrađena je s modernim alatima za web razvoj, pri čemu je korišten ASP.NET Razor za frontend, a C# za backend. RFID skener je povezan s Dasduino ESP32 ConnectPlus pločicom koja omogućava bežični prijenos podataka o skeniranju.

Ključne riječi: prisutnost studenata, RFID tehnologija, web aplikacija, ASP.NET, Supabase, Dasduino

ABSTRACT

Internet application for student attendance tracking using RFID/NFC student card reader

The goal of this application was to develop a web system for tracking student attendance using RFID technology. The application allows professors to monitor student arrivals at lectures in real time through RFID card scanning. The web application uses the Supabase platform for data storage, where attendance data, including students' names and attendance percentages, is recorded. It was created with modern web development tools, using ASP.NET Razor for the frontend and C# for the backend. The RFID scanner is connected to a Dasduino ESP32 ConnectPlus board, which enables wireless transmission of scanning data.

Keywords: student attendance, RFID technology, web application, ASP.NET, Supabase, Dasduino