Primjena .NET tehnologije za sustavnu evidenciju nazočnosti studenata na nastavnim aktivnostima

Frčko, Branimir

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Bjelovar University of Applied Sciences / Veleučilište u Bjelovaru**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:144:793101

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-09-14**

Repository / Repozitorij:

Repository of Bjelovar University of Applied Sciences - Institutional Repository

VELEUČILIŠTE U BJELOVARU
STRUČNI PRIJEDIPLOMSKI STUDIJ RAČUNARSTVO

Primjena .NET tehnologije za sustavnu evidenciju nazočnosti studenata na nastavnim aktivnostima

Završni rad br. 02/RAČ/2023

Branimir Frčko

Bjelovar, rujan 2023.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Branimir Frčko** JMBAG: **0246089709**

Naslov rada (tema): **Primjena .NET tehnologije za sustavnu evidenciju nazočnosti studenata na nastavnim aktivnostima**

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

Grana: **Primijenjeno računarstvo**

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. **Danijel Radočaj, mag. inž. meh., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 02/RAČ/2023

U sklopu završnog rada potrebno je:

1. Analizirati i opisati .NET razvojni okvir za izradu MVC aplikacija i usporediti ga s drugim razvojnim okvirima.
2. Analizirati i implementirati podršku za korištenje studentskih identifikacijih kartica u aplikaciji.
3. Predložiti i opisati relacijski model baze podataka koji će omogućiti pohranu svih podataka evidencije nazočnosti studenata.
4. Izraditi i opisati rješenje .NET aplikacije za evidenciju nazočnosti studenata na nastavnim aktivnostima.
5. Izraditi i opisati podsustav za komuniciranje s hardverskim dijelom sustava.

Datum: 09.06.2023. godine

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

Zahvala

Zahvaljujem se svom mentoru Krunoslavu Husaku, dipl. ing. rač. na potpori i vodstvu tijekom pisanja završnog rada. Također bi se želio zahvaliti ostalim djelatnicima Veleučilišta u Bjelovaru koji su me podržavali i pomogli mi u ostvarivanju mojih ciljeva. Naposljetku, želim se zahvaliti svojoj obitelji na neizmjernoj podršci te ohrabrivanju koji su mi pružili tijekom cijelog mog obrazovanja.

Sadržaj

1. UVOD

U današnjem digitalnom dobu tehnološki napredak oblikuje našu svakodnevnicu i potiče inovacije u različitim područjima. U području evidencije prisutnosti na nastavnim aktivnostima, posebno u obrazovanju, sve veći broj institucija okreće se automatiziranom modelu prijave prisutnosti pomoću *Internet of Things* (IoT) tehnologije. Zastarjele metode bilježenja prisutnosti imaju svoje mane, kao što su potpisivanje koje nastavnicima oduzima dosta vremena, dosta teška kontrola prisutnosti, potrošnja velike količine papira čime se zagađuje okoliš te gubitak tih istih papira. IoT rješenjem rješavaju se svi ti problemi na način da se dolasci automatski zbrajaju kada student provuče svoju studentsku ispravu te se dolaznost odmah sprema u bazu podataka koja je uvijek dostupna i ažurna. Takav način štedi puno vremena nastavnicima koji onda mogu provesti svoje vrijeme kako bi unaprijedili predavanja, a ne brojali potpise.

IoT tehnologija omogućuje povezivanje i komunikaciju različitih uređaja putem interneta. Kombinacijom IoT uređaja i pametnih identifikacijskih oznaka poput UUID-ova, RFID-ova, QR kodova, NFC-a, i dr. pruža se mogućnost preciznog i automatiziranog praćenja prisutnosti učenika na nastavi. Ova metoda omogućuje brzu i pouzdanu evidenciju prisutnosti koja zamjenjuje tradicionalne metode. IoT metodom izbjegavamo ljudsku pogrešku koja se uvijek zna dogoditi te mogućnost prevare sistema koju neki učenici koriste kako bi izbjegli svoje nastavne aktivnosti.

Cilj ovog završnog rada je istražiti i implementirati .NET rješenje za evidenciju prisutnosti na predmetima uz korištenje IoT uređaja za skeniranje UUID-a. Upotrebom .NET platforme, razvijena je aplikacija visokih performansi koja bilježi i provjerava prisutnost učenika na temelju podataka dobivenih putem skeniranja UUID-ova na IoT uređaju. Ista tehnologija također se koristi za upravljanje bazom podataka te za prikaz prisutnosti učenika. Krajnji rezultat je olakšavanje praćenja prisutnosti na predmetima na Veleučilištu u Bjelovaru. Kroz ovaj rad bit će prikazana korisnost i primjena .NET rješenja za evidenciju prisutnosti na predmetima uz pomoć IoT tehnologije.

U nastavku ovog rada detaljnije će se proći kroz proces razvoja aplikacije, korake koji su poduzeti pri implementaciji sustava, kao i prednosti i izazove s koji su nastali tijekom procesa. Također, analizirat će se dobiveni rezultati te raspravljati o daljnjem unaprjeđenju sustava.

Rad je podijeljen na šest poglavlja. Drugo poglavlje opisuje osnovne pojmove .NET-a, njegovu povijest te neke prednosti i nedostatke. Treće poglavlje govori detaljnije o značajkama .NET-a kao što su CLR, BCL i LINQ. Četvrto poglavlje opisuje neke naprednije koncepte koje sam koristio prilikom izrade rješenja. Peto poglavlje je fokusirano na implementaciju rješenja. U šestom poglavlju navodi se koja bi poboljšanja i izmjene mogle implementirati. Posljednja poglavlja sadržavaju zaključak i sažetak te literaturu i oznake.

2. UVOD U .NET

2.1 .NET

.NET je popularna platforma za razvoj softvera tvrtke Microsoft. Smatra se ”programskim okvirom” odnosno alatom koji programerima pomaže u razvijanju raznih aplikacija kao što su web stranice, desktop programi ili mobilne aplikacije. .NET je zapravo skup alata, tehnologija i jezika koji omogućuju razvoj, testiranje i izvođenje aplikacija na raznim sustavima. [1] Programeri svoj kod mogu pisati u C#, F# te Visual Basic jeziku iz razloga što se prevode u zajednički jezik izvođenja CIL. [3] Aplikacije funkcioniraju na računalima i uređajima različitih operativnih sustava kao što su Windows, Linux ili macOS. Također, .NET pruža različite okvire za specifične vrste aplikacija. Na primjer, ASP.NET je okvir koji se koristi za izgradnju web stranica i web aplikacija. [3]

Slika 2.1: Logo .NET-a

2.2 Povijest

Bogata povijest .NET-a seže unatrag više od 20 godina. Razvoj .NET-a započela je vodeća tehnološka tvrtka Microsoft kojoj je bilo potrebno suvremeno i fleksibilno okružje za razvoj softvera. Prva verzija, pod imenom .NET Framework 1.0 objavljena je u veljači 2002. godine. Najveća pozornost pridonijeta je *Common Language Runtime* (CLR), odnosno

izvršnom okruženju .NET aplikacija. Tijekom godina .NET platforma je prošla kroz brojne nadogradnje i izmjene. Svakom novom verzijom dodavane su nove značajke, poboljšanja performansi te podrške za nove tehnologije. Microsoft je 2016. godine prepoznao potrebu za razvojem .NET platforme za različite operativne sustave te tako predstavio .NET Core, modularnu verziju .NET-a koja je otvorenog koda i podržava razvoj na Windowsu, Linuxu i macOS-u. Sljedeći veliki pomak dogodio se 2018. godine kada su se .NET Core i .NET Framework integrirali u jedan okvir koji se nazvao .NET, a cilj je bio pojednostaviti i ujediniti razvojnu platformu. Suvremena verzija naziva se .NET 7.0.

2.3 Prednosti

Sintaksni jezici poput C#, F# te VB.NET pružaju intuitivan i jasan način pisanja koda čime programerima olakšavaju razvoj aplikacija. Također, bogata biblioteka klasa i alata omogućuje iskorištavanje gotovih funkcionalnosti te time smanjuje vrijeme potrebno za pisanje koda. Spomenuti .NET Core je prenosiva verzija .NET-a koja omogućuje razvoj na raznim platformama kao što su Windows, Linux i macOS te time omogućuje fleksibilnost. Programeri tako ciljaju na širi raspon korisnika bez obzira na korištenje raznih uređaja i operativnih sustava. [4]

Uz prenosivost, .NET također podržava integraciju s već postojećim sustavima čime olakšava razvoj aplikacija u različitim okruženjima što također doprinosi fleksibilnosti. ASP.NET omogućuje efikasno razvijanje sigurnih, skalabilnih web stranica s dobrim performansama. Programeri koristeći ASP.NET mogu kombinirati mogućnosti HTML-a, CSS-a te JavaScripta s naprednim mogućnostima MVC (Model-View-Controller) arhitekture kako bi izgradili modernu i funkcionalnu web stranicu s minimalnim naporom.

2.4 Nedostatci

Većina alata, okruženja i usluga za razvoj .NET aplikacija se temelji na Microsoft tehnologijama što može ograničiti programere koji preferiraju druge tehnologije ili platforme. Također, ovisnost o Microsoft sustavu može stvoriti dodatne troškove zbog cijene licence ili podrške za neki alat. Veličina instalacije za pokretanje .NET aplikacija nešto je veća što može predstavljati problem za aplikacije koje trebaju biti distribuirane na uređajima

s ograničenim prostorom za pohranu. Veća veličina instalacije također može negativno utjecati na korisničko iskustvo prilikom preuzimanja ili instaliranja aplikacije. Zajednica otvorenog koda ne doseže razinu nekih drugih popularnih platformi kao što su Python ili Java što može ograničiti dostupnost otvorenog koda, resursa i zajedničkog dijeljenja znanja te rješavanja specifičnih problema. Manja zajednica također utječe na dostupnost gotovih biblioteka i rješenja koja se mogu iskoristiti kako bi se ubrzao rad. [5]

3. Značajke .NET-a

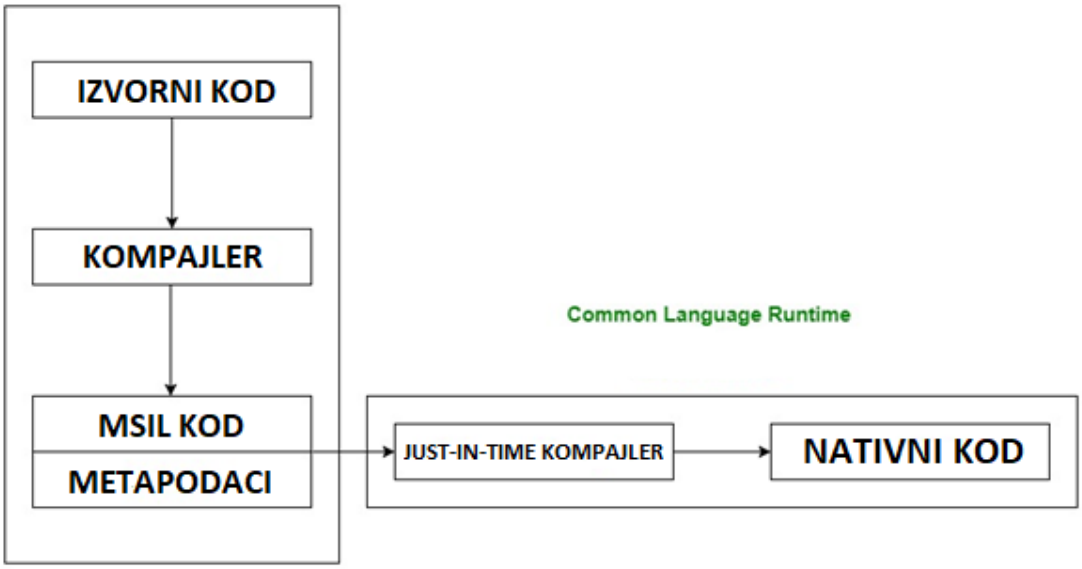
3.1 CLR

Common Language Runtime (CLR) je središnji izvršni okvir .NET platforme. CLR pruža važne značajke kao što su sakupljanje smeća (engl. garbage collector), *just-in-time* (JIT) kompilacija te sigurnost tipova. Također, omogućuje interoperabilnost između različitih jezika koji se izvode na .NET platformi. Pojednostavljen prikaz CLR-a prikazan je na slici 3.1. [6]

Takozvano sakupljanje smeća (engl. garbage collector) je automatizirano upravljanje memorijom implementirano na .NET platformi od strane CLR-a. *Garbage collector* olakšava programeru posao upravljanja memorijom tako što ne mora ručno pratiti i oslobađati memorijske resurse. Proces se vrši automatski i oslobađa dio memorije koja više nije u uporabi, čime se poboljšava produktivnost programera i smanjuje mogućnost grešaka koje se mogu dogoditi pri upravljanju memorijom.

Just-in-time kompilacija je proces kojim se kod .NET platforme pretvara iz izvornog u strojni jezik koji računalni sustav može izravno izvršavati. CLR koristi JIT kompilaciju kako bi povećao performanse izvršavanja .NET programa. Umjesto prevođenja cijelog koda prije izvršavanja, JIT kompilacija izvršava prevođenje neposredno prije izvršavanja dijela koda koji se treba izvesti. JIT kompilacija omogućuje optimizaciju koda za određenu platformu te poboljšava performanse i brzinu izvršavanja aplikacija.

CLR pruža sigurnost tipova koja osigurava izvođenje koda samo ako su ispunjeni sigurnosni uvjeti. Svaka vrijednost i objekt .NET platforme imaju informacije o tipu što se koristi za provjeru sigurnosti i smanjuje rizik od pogrešaka vezanih uz tipove. Samim time poboljšava stabilnost i pouzdanost aplikacije iz razloga što neće dolaziti do neispravnog korištenja objekata ili neispravnog pristupanja memoriji čime se štiti zdravlje računala posebno CPU-a i RAM-a.



Slika 3.1: Pojednostavljen prikaz kompajliranja koda u .NET-u

3.2 BCL

Base Class Library (BCL) je .NET-ova velika kolekcija klasa, sučelja i metoda koje pružaju neke gotove funkcionalnosti za razvoj aplikacija. Postoje razne biblioteke koje se koriste za obradu teksta, rad s datotekama, mrežno programiranje, upravljanje bazama podataka, grafičko korisničko sučelje, itd. Korištenje BCL-a programerima omogućava bržu i efikasniju izradu aplikacija zbog korištenja već gotovih komponenti koje su testirane i optimizirane.

BCL sadrži bogatu kolekciju klasa za upravljanje skupovima podataka kao što su liste, nizovi, mape, redovi, itd. Također uključuje generičke tipove koje koriste programeri kako bi stvorili sigurne tipove podataka koji se mogu koristiti s različitim vrstama skupova podataka. Ove značajke znatno olakšavaju manipulaciju podacima te organizaciju struktura podataka. Postoje razne klase i metode za rad s datotekama, direktorijima, mrežnim protokolima te za komunikaciju putem mreža.

Značajke se koriste kako bi olakšale čitanje, pisanje, brisanje i manipulaciju datotekama te za uspostavljanje stabilnih mrežnih veza i sigurnu razmjenu podataka između aplikacija. Ugrađene komponente za grafičko korisničko sučelje (GUI) omogućuju intuitivnu i interaktivnu izgradnju kvalitetnih korisničkih sučelja. GUI sadrži gotove klase za

upravljanje prozorima, kontrolama, događajima te vizualnim elementima kao što su gumbovi, polja za unos teksta, slike i drugo. Klase i sučelja za upravljanje bazama podataka iznimno olakšavaju uspostavljanje veze s bazom podataka, izvršavanje upita, upravljanje transakcijama, dohvat podataka, ažuriranje zapisa i druge operacije koje uključuju baze podataka. Korištenjem tih klasa i sučelja ubrzava se razvoj aplikacija te omogućuje sigurnije i lakše održavanje kako aplikacije, tako i baze podataka.

3.3 LINQ

Language Integrated Query (LINQ) tehnologija je sastavni dio .NET platforme koja pruža jednostavnu sintaksu za izvršavanje upita nad različitim izvorima podataka. LINQ omogućuje izvršavanje upita nad kolekcijama objekata, bazama podataka, XML dokumentima te drugim izvorima podataka. Upiti se mogu pisati izravno unutar programskog jezika bez potrebe za vanjskim alatima koji usporavaju sustav i razvoj aplikacija. Jednostavna sintaksa upita omogućuje lako razumijevanje i pisanje upita za selektiranje, filtriranje, grupiranje te sortiranje podataka. Uz jednostavnu sintaksu LINQ također sadrži i tipizirane upite što znači da se provjera tipova vrši tijekom kompilacije čime se ubrzava otkrivanje grešaka i povećava sigurnost aplikacije. Podržavanje raznih izvora podataka omogućuje korištenje iste sintakse za izvršavanje upita nad različitim izvorima podataka. [7]

LINQ podržava razne tipove upita, kao što je LINQ to *Objects* koji se koristi za izvršavanje upita nad kolekcijama objekata u memoriji pružajući funkcionalnosti kao što su grupiranje, filtriranje, sortiranje i prikaz podataka. Također podržava LINQ TO SQL i LINQ to Entities koji omogućuju mapiranje relacijskih baza podataka čime se olakšava manipulacija podacima iz baza podataka. LINQ to XML pruža programerima da koriste XML dokumente kao kolekcije objekata za jednostavno pretraživanje i transformaciju XML podataka.

Ekstenzije i lambda izrazi ključni su koncepti LINQ-a. Ekstenzijske metode omogućuju dodavanje novih metoda kolekcijama što znači pisanje čitljivog koda koristeći lančane pozive metoda. Lambda izrazi pružaju programerima definiranje anonimnih funkcija unutar upita koje se najčešće koriste za filtriranje i prikaz podataka.

Jedan od bitnijih koncepta LINQ-a je odgođeno izvršavanje (engl. Deferred Execution) koji pruža odgađanje izvršavanja upita sve dok podaci zaista nisu potrebni. Izbjegavanje

nepotrebnog izvršavanje upita doprinosi optimizaciji performansi iako je upite moguće odmah izvršiti koristeći trenutačno izvršavanje (engl. Immediate Execution) kada je to potrebno. Kako bi još više poboljšao performanse i optimiziranost, LINQ podržava i asinkroni pristup pisanja upita.

LINQ pojednostavljuje razvoj i održavanje aplikacija, povećava čitljivost i razumijevanje koda te smanjuje mogućnost pogrešaka.

Slika 3.2: Logo LINQ-a

4. Napredni koncepti

U ovom poglavlju detaljnije se ulazi u napredne koncepte koji su korišteni prilikom izrade rješenja. Opisan je *Model-View-Controller* obrazac dizajniranja i njegove prednosti, objašnjen je Entity Framework i zašto je bio potreban, te je opisana višedretvenost i zašto je neophodna kako bi rješenje radilo kao što je predviđeno.

4.1 Uvod u MVC

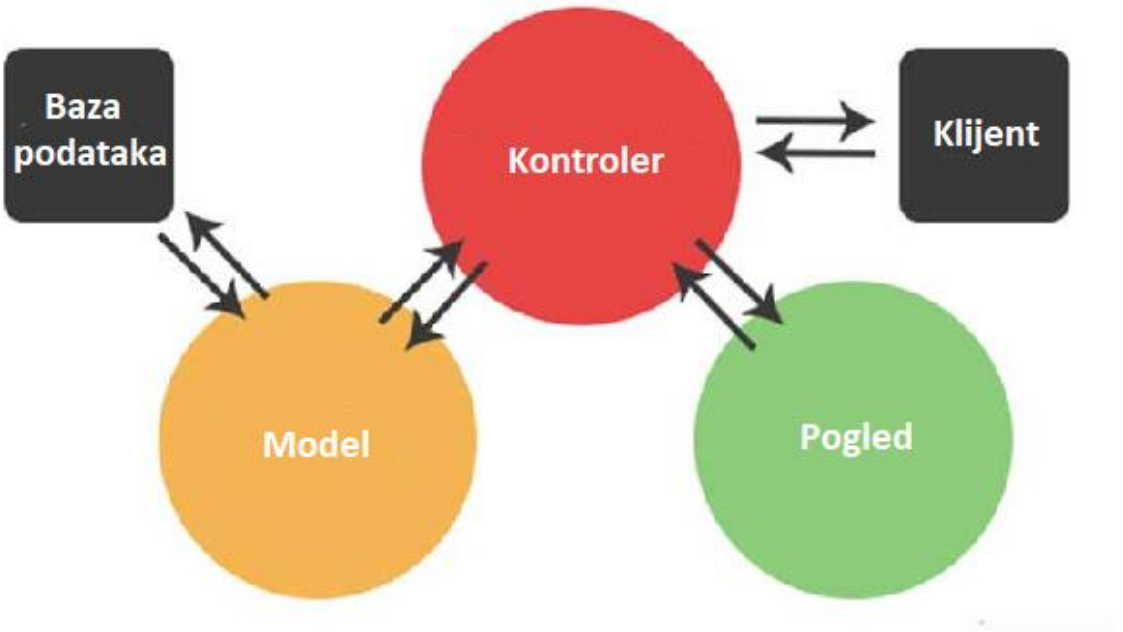
Model-View-Controller (MVC) je obrazac dizajniranja aplikacije koji je danas u širokoj upotrebi kada se radi o web aplikacijama. [8] MVC pruža strukturu i organizaciju za razdvajanje različitih aspekata aplikacije te omogućava lakši razvoj, održavanje i skaliranje aplikacije. Glavna ideja je jasno razdvajanje odgovornosti između tri glavne komponente: modela, pogleda i kontrolera. Svaka od komponenata ima specifičnu ulogu i odgovornost čime se postiže visoka modularnost i fleksibilnost aplikacije. Pojednostavljen prikaz MVC strukture predstavlja slika 4.1. [9]

- Model (engl. Model) - sloj aplikacije koji obavlja poslovnu logiku, obradu podataka te komunikaciju s bazom podataka. Model se sastoji od entiteta, klasa, sučelja i metoda koje se koriste za manipulaciju podacima.

- Pogled (engl. View) - predstavlja korisničko sučelje, odnosno ono što korisnik vidi i s čime vrši interakciju. Pogled je odgovoran za prikaz podataka iz modela na korisničkom sučelju. .NET sadrži tehnologiju *Razor* koju programeri najčešće koriste prilikom izrade web stranica. Prednost *Razor* tehnologije je što omogućuje kombiniranje HTML i C# koda za generiranje korisničkog sučelja. Pogled ne sadrži nikakvu logiku, te samo služi kao prikaz podataka koje mu pruža kontroler (*engl. controller*).

- Kontroler (engl. Controller) - vrši obradu ulaznih zahtjeva korisnika i upravljanja tokom izvršavanja aplikacije. Kontroler prima zahtjeve od korisnika , komunicira s modelom za dohvat i manipulaciju podacima te odabire odgovarajuće poglede za

prikaz rezultata koje je korisnik zatražio. Kontroler ima funkciju posrednika između modela i pogleda i tako osigurava da se podaci pravilno prikazuju i ažuriraju. .NET implementira kontrolere kao klase koje nasljeđuju gotovu klasu *ControllerBase*.



Slika 4.1: Prikaz MVC strukture

4.2 Prednosti MVC-a

MVC pruža jasnu odvojenost odgovornosti između modela, pogleda i kontrolera što omogućuje jednostavniji timski rad jer je svaka komponenta za nešto zadužena. Također, velike prednosti su i lakše testiranje pojedinačnih komponenti te olakšavanje održavanja koda. Svaka komponenta se može mijenjati neovisno o drugim komponentama što omogućuje jednostavno dodavanje novih funkcionalnosti ili promjenu postojećih. Primjer bi bio promjena ili nadogradnja pogleda bez utjecaja na model ili kontroler. [10]

Najveća korist se vidi u razvoju velikih aplikacija s više timova ili u situacijama kada je potrebno brzo reagirati na neke promjene, a zna se točno gdje se što nalazi. Svaku komponentu moguće je testirati odvojeno koristeći razne pristupe. Model se može testirati na poslovnoj logici i pravilnosti manipulacije podacima, pogled na ispravnom prikazu podataka i interakciji s korisnikom dok se kontroler testira na obradi zahtjeva i komunikaciji

s drugim komponentama. Testiranje komponenata odvojeno olakšava pronalaženje i ispravljanje grešaka u kodu te osigurava kvalitetu koda. Napravljene komponente mogu se koristiti u više različitih projekata ili više različitih dijelova istog projekta. Primjer je pogled koji se ponovno iskoristi za prikazivanje sličnih podataka na različitim stranicama, čime se smanjuje ponavljanje koda, povećava efikasnost te olakšava održavanje sustava. Dostupni su brojni gotovi alati i okviri koji olakšavaju implementaciju MVC arhitekture, pružaju napredne značajke i funkcionalnosti za brži i lakši razvoj.

4.3 Entity Framework

Entity Framework (EF) je tehnologija objektno-relacijskog mapiranja (ORM) koja omogućuje lakši rad s bazom podataka u .NET okruženju. EF omogućuje jednostavno povezivanje aplikacije s različitim vrstama baza podataka kroz konfiguracijske postavke i konekcijske zapise. Neke od vrsta baza podataka su SQL Server, MySQL i Oracle. Entity Framework koristi pristup temeljen na modelu kako bi omogućio mapiranje baze podataka na objekte. [11]

Entiteti koji se definiraju predstavljaju tablice u bazi podataka, atribute koji predstavljaju stupce i relacije između objekata. Kroz konfiguracije EF generira SQL upite za rad s bazom podataka na temelju definiranog modela. Upiti nad bazom podataka vrše se prije spomenutom LINQ tehnologijom koju EF prevodi u SQL upite te ih izvršava na bazi podataka. Mehanizam migracija olakšava upravljanje promjenama u strukturi baze podataka tijekom razvoja aplikacije. Definiraju se migracije koje opisuju promjene u modelu podataka dok Entity Framework automatski generira SQL skripte za promjene na bazi podataka. Migracije olakšavaju održavanje i ažuriranje baze podataka bez gubitaka podataka. Također posao olakšavaju i sigurnosni mehanizmi koji štite aplikaciju od uobičajenih napada, poput SQL injekcije. [12]

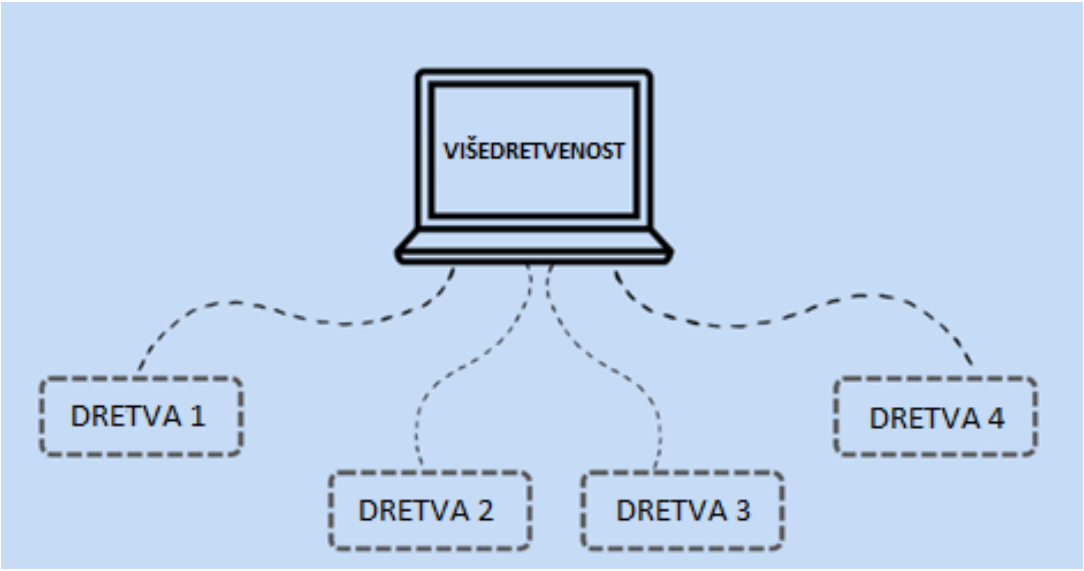
4.4 Višedretveno programiranje

Višedretvenost je tehnika u programiranju koja označava izvršavanje više dijelova koda istovremeno. Više dretvi (engl. threads) izvršava se u isto vrijeme, ali svaka s vlastitim izvršnim tokom. U .NET okruženju višedretvenost se implementira kroz korištenje klase

Thread ili preko naprednijih konstrukcija poput *Task*, *ThreadPool* ili *Parallel* biblioteke. Svaka dretva predstavlja izvršnu jedinicu koja paralelno izvršava određeni kod. Korištenje višedretvenosti zahtijeva pažljivo upravljanje resursima i sinkronizacijom kako bi se izbjegli problemi. [13] Primjer višedretvenosti prikazuje slika 4.2.

S druge strane, asinkrono programiranje omogućuje izvršavanje zadataka bez blokiranja glavne dretve izvođenja, odnosno aplikacija može nastaviti izvršavati zadatke dok čeka rezultat asinkronog zadatka. Asinkrono programiranje je korisno u situacijama kada aplikacija mora obavljati dugotrajne operacije poput čitanja ili pisanja u bazu podataka, poziva web usluga ili izvršavanja operacija s mrežnim resursima. Asinkrono programiranje u .NET-u postiže se korištenjem ključne riječi *async i await.* Asinkronim programiranjem glavna dretva oslobađa se za izvršavanje drugih zadataka dok se asinkroni zadaci izvršavaju paralelno što poboljšava odziv aplikacije, omogućuje bolje korištenje resursa i smanjuje potrebu za stvaranjem i upravljanjem vlastitim dretvama. [14]

Kombinacija višedretvenosti i asinkronog programiranje u .NET-u omogućuje izradu skalabilnih, odzivnih i efikasnih aplikacija. Koristeći više dretvi i asinkrone zadatke aplikacija može u isto vrijeme izvršavati različite zadatke te time poboljšati performanse, smanjiti vrijeme čekanja i bolje iskoristiti resurse sustava. Kombinacija je posebno korisna u situacijama kada aplikacija mora odgovoriti na veliki broj zahtjeva istovremeno ili kada se obrađuju dugotrajne operacije, a ne želi se blokirati glavna dretva izvođenja. [15]



Slika 4.2: Primjer višedretvenosti

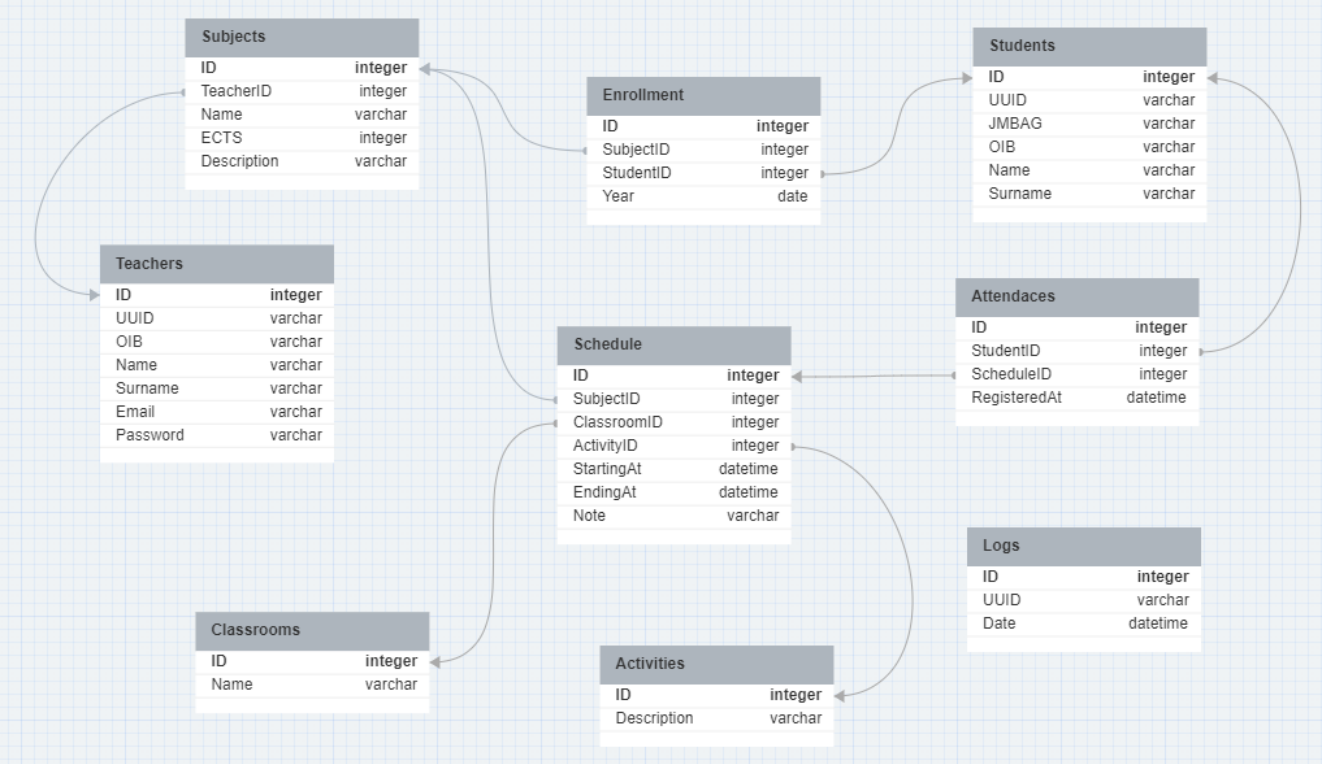
5. Implementacija programskog rješenja

U ovom poglavlju opisan je proces izrade aplikacije te dijelova aplikacije koji su baza podataka, izrada višedretvenosti za komunikaciju s IoT uređajem te korisničko sučelje i funkcionalnosti. Također, objašnjen je način na koji su svi dijelovi aplikacije povezani te kako zajedno rade kao jedan sustav.

5.1 Baza podataka

Za izradu ovog rada korišten je *Entity Framework* opisan u poglavlju 4.3 u kombinaciji s ekstenzijom SQL Servera za upravljanje bazama podataka. Za vizualizaciju baze podataka te lakše dizajniranje i modeliranje korišten je DB Designer.

Baza podataka je stvorena tako da su prvo izrađeni modeli za podatke koje je trebalo spremiti, a zatim su oni međusobno povezani preko stranih ključeva. Kada su odnosi između entiteta i povezanost podataka bili zadovoljavajući napravljen je *scaffolding* kako bi se generirali kontroleri. Nakon toga izvršena je migracija kako bi se sačuvalo tadašnje stanje aplikacije. Nakon migracije ažurirana je baza podataka. Prikaz povezanosti podataka te dijagram baze podataka može se vidjeti na slici 5.1, dok primjer modela na programskom kodu 5.1, a primjer koda za *scaffolding* na programskom kodu 5.2.



Slika 5.1: Dijagram baze podataka

- *Users* – glavna tablica na koju se sve veže te se spremaju korisnici koji će koristiti aplikaciju odnosno u ovom slučaju profesori
- *Subjects* – tablica koja sadrži informacije za sve predmete koji se mogu izvoditi odnosno njihovo ime, broj ECTS bodova koji nose te opis
- *Students* – tablica koja pohranjuje sve osobne informacije o svim studentima koji pohađaju faks te najbitnije njihov UUID studentske isprave
- *Enrollment* – tablica koja veže podatke iz tablice *Students* i iz tablice *Subjects*. Tablica služi kako bi povezala studente te predmete koje oni pohađaju i godinu pohađanja tog predmeta
- *Attendances* – tablica koja povezuje studenta i raspored, u toj tablici se zapisuje podatak kada je došao na koje predavanje i iz te tablice se povlače podaci za prikaz dolaznosti studenata
- *Activites* – tablica u koju pohranjujem podatke kakve sve aktivnosti se mogu provoditi, npr. laboratorijske vježbe.

- *Classrooms* – tablica koja sadrži sve moguće dvorane za provoditi nastavu, bitna tablica koja se veže na tablicu *Schedule*
- *Schedule* – tablica rasporeda koja povezuje tablicu *Classrooms* u kojoj su spremljene učionice u kojima se mogu provoditi nastavne aktivnosti, povezana je sa tablicom *Activities* iz koje doznajemo koje sve aktivnosti mogu biti te tablicu *Subjects* koja sadrži predmete koji se mogu izvoditi. Iz te tablice povlače se informacije u tablicu *Attendances* kako bi se dobile informacije kada predmet počinje i do kada traje kako bi utvrdili njegovu prisutnost na predmetu.

Programski kod 5.1. prikazuje model *Attendances* koji je korišten za generiranje tablice u bazi podataka i kontrolera.

Programski kod 5.1: Model Attendances

```
public class Attendance
{
public int Id { get; set; }
public int StudentId { get; set; }
public int ScheduleId { get; set; }
public DateTime RegisteredAt { get; set; }
public Student? student { get; set; }
public Schedule? schedule { get; set; }
}
```

Programski kod 5.2. demonstrira kod za *scaffolding* modela s kojim se generira kontroler za određeni model.

Programski kod 5.2: Kod za scaffolding modela

```
dotnet aspnet-codegenerator controller -name SchedulesController -m Schedule -dc AttendanceProjectbm --relativeFolderPath Controllers --useDefaultLayout --referenceScriptLibraries
```

Entity Framework generira migraciju odnosno C# datoteku prepoznatljivu po .cs ekstenziji. U migraciji su definirane sve upute i pravila pojedinog modela za izradu tablica. Pravilima se određuje tip podatka, definiraju odnosi između podataka, određuju primarni ključeve i sl. Važno je napomenuti da shema baze podataka igra važnu ulogu u definiranju strukture i odnosa između različitih relacijskih tablica.

Kako aplikacija raste i razvija se tijekom vremena neophodno je mijenjati strukturu podataka kako bi odgovarala zahtjevima. U takvim situacijama migracija postaje neizostavan alat za održavanje baze podataka ažurnom. Na primjer, migracija omogućuje dodavanje novih svojstava u modele bez gubitka podataka. Proces migracije radi tako da se trenutno stanje modela uspoređuje s prethodnim. Rezultat je generiranje datoteka koje sadrže klase s *Up* i *Down* metodama koje omogućuju primjenu i reverziju promjena u bazi podataka čime se omogućuje jednostavno upravljanje promjenama te održavanje dosljednosti baze podataka tijekom cijelog životnog ciklusa aplikacije.

5.2 IoT komunikacija

Višedretvenost se koristi za neprestanu komunikaciju između IoT uređaja koji očitava RFID i .NET aplikacije koja služi kao prikaz dolaznosti studenata. Neprekidno u drugoj dretvi .NET asinkroni sustav čeka poruku IoT uređaja preko MQTT-a. IoT uređaj će poslati RFID studentske isprave .NET asinkronom sustavu koji će zatim zapisati dolazak studenta na nastavnu aktivnost. U programskom kodu 5.2. vidljiv je Client.cs odnosno bitne linije koda za izvesti ovu funkcionalnost iz Client.cs klase koja vrši višedretvenost te omogućuje neprekidnu komunikaciju između IoT uređaja i moje aplikacije. Shema IoT komunikacije je prikazana na slici 5.3.

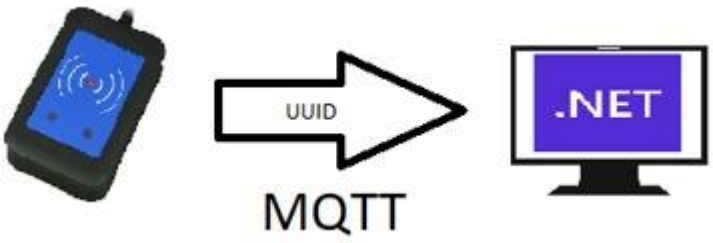
Programski kod 5.3: Višedretvenost

```
public class Client
    {
        public Client()
        {

            Task t = Task.Run(() =>
        {
            //Ostatak koda za MQTT i slanje podataka u bazu
        });
        }
    }
```

Kako bi funkcionalnost iz programskog koda 5.2 bila omogućena mora biti pozvana u Program.cs prije linije za pokretanje aplikacije odnosno prije app.run() kao u primjeru programskog koda 5.4.

Programski kod 5.4: Pozivanje Client.cs

```
AttendanceProject.Client client = new AttendanceProject.Client();

app.Run();
```



Slika 5.3: Shema IoT komunikacije

5.3 MQTT

MQTT (engl. Message Queuing Telemetry Transport) je komunikacijski protokol koji je dizajniran za efikasnu i pouzdanu razmjenu informacija tj. poruka između uređaja u IoT okruženju. Protokol koristi model izdavač-pretplatnik za razmjenu poruka između uređaja. [16]

U ovom sustavu IoT uređaj je izdavač (engl. publisher) koji šalje poruku kada skenira UUID sa studentske isprave. Pretplatnik (engl. subscriber) je .NET aplikacija koja je registrirana na određenu temu gdje očekuje da će doći poruka. MQTT broker je posrednik između izdavača i pretplatnika koji upravlja prometom poruka te bilježi koje teme pretplatnici prate i osigurava ispravno usmjeravanje poruka prema pretplatnicima. Kada izdavač objavi poruku na neku temu broker će je proslijediti svim pretplatnicima te teme. [17]

Slika 5.4: Logo MQTT-a

.NET aplikacija ima posebnu dretvu na kojoj se pokreće dio programa koji se spaja na MQTT server te pretplaćuje na odabranu temu u ovisnosti o profesoru koji izvodi nastavnu aktivnost. Prema profesoru, vremenu izvođenja i rasporedu zna se koja je nastavna aktivnost trenutno aktivna te za koju je potrebno bilježiti dolaske. Time se omogućuje da se podaci ažuriraju u stvarnom vremenu bez kašnjenja. Kada .NET dretva za komunikaciju s IoT uređajem dobije ispravnu poruku tj. ispravan studentski UUID sa studentske isprave spaja se na bazu podataka i zapisuje dobiveni UUID i vrijeme dolaska u tablicu *Logs* koja se može vidjeti na slici 5.5. Podaci za prikaz izvlače se iz tablice *Logs* te se prikazuju u tablici dolaznosti odnosno u tablici *Attendance* koja je prikazana na slici 5.6.

Id	UUID	Date
Filter...	Filter...	Filter...
1	81818181	2023-07-05T21:53:00.000Z
2	12345678	2023-06-07T19:53:00.000Z
3	87654321123	2023-08-03T19:54:00.000Z
4	87633321123	2023-06-06T19:56:00.000Z
5	12345678	2023-08-10T19:54:00.000Z
6	12345678	2023-08-26T19:59:00.000Z
7	12345678	2023-05-10T19:55:00.000Z

Slika 5.5: Prikaz Logs tablice

Id	StudentId	ScheduleId	RegisteredAt
Filter...	Filter...	Filter...	Filter...
1	1	1	2023-07-22T10:13:00.000Z
9	5	1	2023-07-15T14:05:00.000Z
10	4	4	2023-07-15T14:11:00.000Z
13	1	4	2023-07-15T14:29:00.000Z
15	2	4	2023-07-16T19:36:00.000Z

Slika 5.6: Prikaz Attendance tablice

5.4 Korisnička strana aplikacije

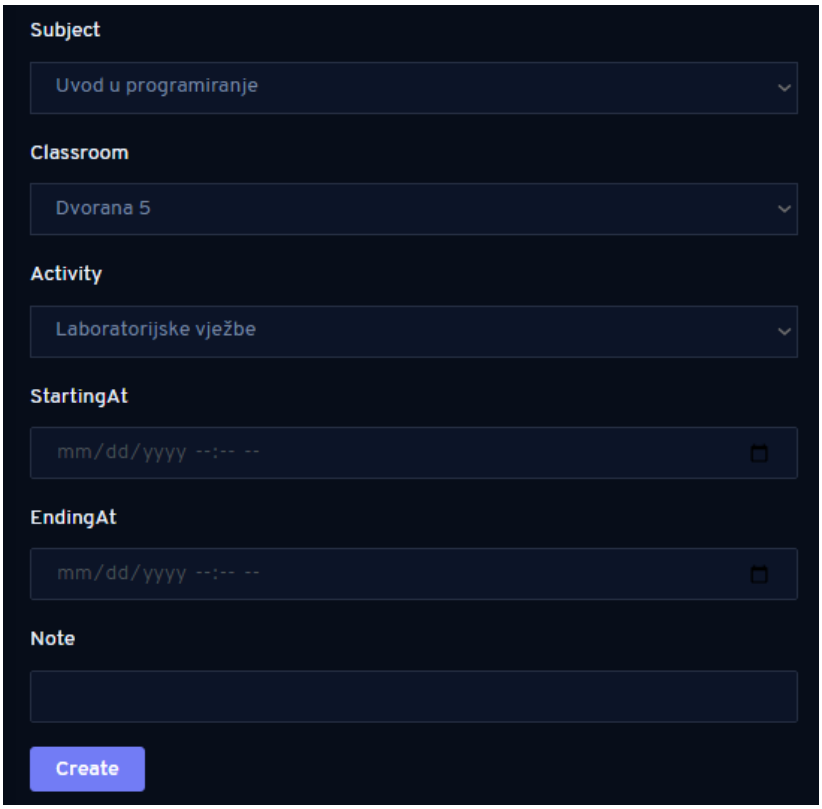
Aplikacija na strani korisnika namijenjena je profesorima kako bi mogli pratiti nazočnost učenika na svojim nastavnim aktivnostima. Aplikacija omogućuje jednostavnu i brzu analizu dolaznosti studenata na nastavne aktivnosti. Na slici 5.7. vidi se prikaz tablice dolaznosti na nastavnu aktivnost.

NAME	SURNAME	SUBJECT	ARRIVED	ATTENDED	STARTED AT	ENDED AT
Matko	Matković	Uvod u programiranje	7/22/2023 10:13:00 AM	✗	6/9/2023 9:00:00 PM	6/9/2023 10:30:00 PM
Petra	Petric	Uvod u programiranje	7/15/2023 2:05:00 PM	✗	6/9/2023 9:00:00 PM	6/9/2023 10:30:00 PM
Mare	Mareic	Uvod u programiranje	7/15/2023 2:11:00 PM	✓	6/14/2023 2:11:00 PM	7/16/2023 2:11:00 PM
Matko	Matković	Uvod u programiranje	7/15/2023 2:29:00 PM	✓	6/14/2023 2:11:00 PM	7/16/2023 2:11:00 PM
Marija	Marijic	Uvod u programiranje	7/16/2023 7:36:00 PM	✗	6/14/2023 2:11:00 PM	7/16/2023 2:11:00 PM

Slika 5.7: Tablica dolaznosti na nastavnu aktivnost

Aplikacija omogućuje profesoru jednostavniju provjeru aktivnosti, a studentima lakšu prijavu jer samo trebaju imati svoju studentsku identifikacijsku karticu te ju prisloniti na IoT uređaj. .NET aplikacija obavlja ostatak posla za njih dok prikazuje profesoru potrebne informacije. *Attendance* stranica prikazuje ime i prezime studenta, predmet koji treba pohađati, kada je došao te je li došao unutar vremena predviđenog za prijavu prisutnosti.

Također, postoje neke dodatne funkcionalnosti kao što je na primjer kreiranje rasporeda. Raspored je vezan na tablice *Subject, Classroom* te *Activity. Subject* tablica sadrži profesora koji izvodi nastavnu aktivnost te neke osnovne informacije o nastavnim aktivnostima kao što su ime, broj ECTS bodova ili neki kratki opis. *Classroom* tablica sadrži sve prostorije u kojima je izvediva nastava. *Activity* tablica u bazi podataka je jednostavna tablica koja samo daje podatke koje sve nastavne aktivnosti postoje. Na primjer, aktivnost može biti laboratorijska vježba, auditorna vježba, predavanje ili se mogu dodati još neke aktivnosti kao što su izvannastavne aktivnosti, sport ili vanjske aktivnosti. Prikaz kreiranja rasporeda je prikazan na slici 5.8.



Slika 5.8.: Kreiranje rasporeda

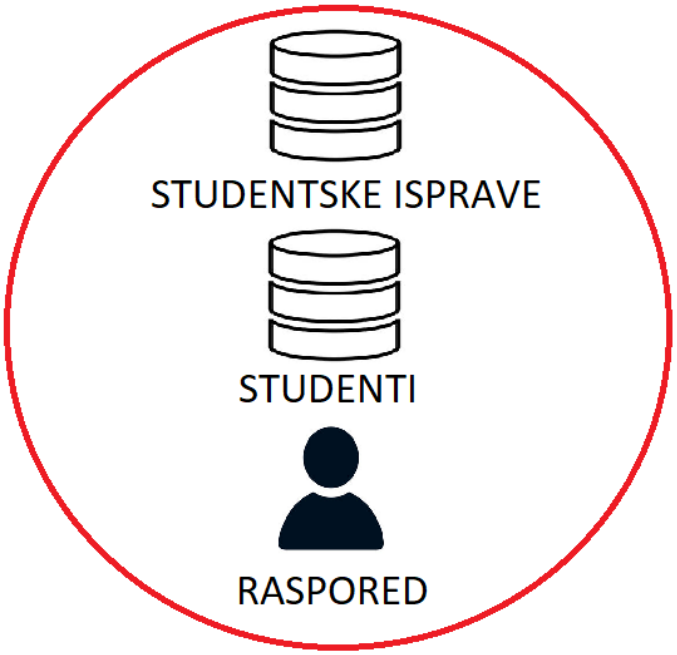
6. Preporuke za unaprjeđenje sustava

U ovom poglavlju objašnjeno je koji su sve načini i razlozi za unaprjeđenje ovog sustava te kako najbolje to ostvariti. Neke nadogradnje su vrlo bitne, dok su druge takozvane *quality of life.*

6.1 Povezivanje s postojećim bazama podataka

Najveće unaprjeđenje aplikacije bi bilo povezivanje s već postojećim bazama podataka iz razloga što je trenutno prazna baza podataka u koju bi se ručno morale unositi sve informacije o studentu od njegovog imena i prezimena do predmeta koje pohađa. Dijagram baze podataka prikazan je na slici 5.1., a s dijagrama se može vidjeti koliko je zapravo informacija potrebno kako bi aplikacija imala sve potrebne podatke i radila kako je zamišljeno.

Baza podataka koja je neophodna sustavu je ISVU baza podataka koja sadrži sve informacije o studentima, predmetima te o studentskim ispravama i njihovim UUID-ovima. Povlačenjem tih podataka i spremanjem u bazu podataka dobiva se većina informacija za automatiziran sustav prijave prisutnosti. Tada nedostaje samo raspored nastavnih aktivnosti koji se lako može implementirati iz razloga što već postoji tablica u bazi podataka i CRUD operacije za njega. Raspored nastavnih aktivnosti bi i dalje morao biti ručno unošen i kontroliran jer je promjenjiv i ovisan o puno faktora stoga bi taj dio sustava morao biti pod nadzorom, dok bi ostatak sustava bio automatiziran. Potpun DB sustav nakon nadogradnje možemo vidjeti na slici 6.1. koja prikazuje shemu potpunog DB sustava i informacije koje su nam potrebne.



Slika 6.1: Shema potpune baze podataka

6.2 Ostala poboljšanja

Jedno od poboljšanja bilo bi složiti neki način registracije na IoT uređaj kako bi svaki profesor mogao koristiti bilo koji IoT uređaj. Trenutnom konfiguracijom na IoT uređaju svaki profesor morao bi imati svoj uređaj koji bi bio vezan uz njegov identitet. Kada bi drugi profesor došao na zamjenu ili bi iz nekog drugog razloga bilo potrebe za promjenom profesora, moralo bi se ulaziti u kod izmijeniti ga. Iz navedenog proizlazi da sustav u tom pogledu nije takozvano *user-friendly*. Također, moguće je da bi netko htio prevariti sustav te koristiti tuđu studentsku iskaznicu tako da bi bilo korisno implementirati neki sustav u kojem bi se uz studentsku iskaznicu na još neki način utvrdio identitet osobe koja dolazi na nastavnu aktivnost. Time poboljšavamo sigurnost i robusnost našeg sustava.

Studentima bi ovaj sustav također bio koristan kada bi imali pristup njemu. Korisno poboljšanje bilo bi kada bi omogućili i studentima prijavu na sustav uz ograničena dopuštenja. Studenti bi mogli pratiti samo svoje nastavne aktivnosti i evidenciju dolazaka na njih kako ne bi bili iznenađeni ako imaju previše izostanaka iz neke nastavne aktivnosti te im je time zabranjen izlazak na ispit tog kolegija.

Kada bi se iskoristila baza podataka koja je objašnjena u poglavlju 6.1 tada bi se mogao složiti sustav prijave na aplikaciju pomoću studentske iskaznice za studente te možda osobne iskaznice ili neke druge s UUID-om za nastavnike. Nitko ne voli pamtiti mail-ove i zaporke za ulaske na razne web stranice stoga bi prijava skeniranjem bila vrlo korisna i brza. Također, još neka poboljšanja koja bi se mogla implementirati su analiza prisutnosti, generiranje izvještaja prisutnosti svih studenata na nekoj nastavnoj aktivnosti kao i izvještaja za svakog studenta ponaosob.

7. ZAKLJUČAK

U ovom završnom radu implementirano je .NET rješenje za evidenciju prisutnosti na predmetima uz pomoć IoT tehnologije. Kroz korištenje informacije dobivene IoT uređajem za skeniranje UUID-a i razvoj aplikacije temeljene na .NET platformi, uspješno je automatiziran sustav praćenja prisutnosti učenika na predmetima na Veleučilištu u Bjelovaru. Primjena IoT tehnologije u obrazovanju donosi mnoge prednosti. Automatizirana evidencija prisutnosti na predmetima povećava učinkovitost te smanjuje administrativne poslove. Ručno bilježenje prisutnosti postaje stvar prošlosti, čime štedimo na vremenu i resursima obrazovnih institucija. Također, povećava se preciznost evidencije jer se koriste pametne identifikacijske oznake i tehnologija skeniranja koju je teško prevariti.

Tijekom implementacije sustava, pojavili su se izazovi. Integracija informacija dobivenih putem IoT uređaja u posebnoj dretvi, razvoj aplikacije te upravljanje bazom podataka zahtijevali su pažljivo planiranje te obuhvatno istraživanje. Zahvaljujući detaljnom istraživanju prevladani su izazovi te postignuti značajni rezultati, kao što je brzo i pouzdano bilježenje prisutnosti.

Ovaj rad daje uvid u korisnost kombinacije .NET platforme i IoT tehnologije u području evidencije prisutnosti na predmetima. Međutim, važno je istaknuti da postoji prostor za daljnji napredak. Na primjer, moguće je sustav učiniti puno sigurnijim i robusnijim što bi bilo neophodno prije komercijalne uporabe. Također, implementacija analize prisutnosti i generiranje izvještaja bili bi odlični dodaci.

Ovaj rad je odlična osnova za automatizirani sustav evidencije prisutnosti na predmetima koji bi značajno olakšao administrativne poslove te poboljšao učinkovitost nastave i praćenje prisutnosti učenika na predmetima. Nadam se da će ovaj rad potaknuti daljnji razvoj sličnih rješenja za automatizaciju u obrazovnom sektoru. Ova inovacija ima potencijal za unapređenje sustava evidencije prisutnosti i poboljšanje učinkovitosti u različitim sektorima.

8. LITERATURA

[1] Microsoft. What is .NET? Introduction and overview [Online]. 24.03.2023. Dostupno na:
https://learn.microsoft.com/en-us/dotnet/core/introduction

[2] GeeksForGeeks. Introduction to .NET Framework [Online]. 20.02.2023.
Dostupno na:
https://www.geeksforgeeks.org/introduction-to-net-framework/

[3] Amazon Web Service. What is .NET? [Online]. 2023.
Dostupno na:
https://aws.amazon.com/what-is/net/

[4] GraffersID. Advantages and Disadvantages of Using .NET in 2023 [Online]. 2023.
Dostupno na:
https://graffersid.com/advantages-and-disadvantages-of-using-net/

[5] NCube. Pros and Cons of .NET Framework [Online]. 24.07.2020.
Dostupno na:
https://ncube.com/blog/pros-and-cons-of-net-framework

[6] Microsoft. Common Language Runtime (CLR) overview [Online]. 25.04.2023.
Dostupno na:
https://learn.microsoft.com/en-us/dotnet/standard/clr

[7] Microsoft. Language Intergrated Query (LINQ) [Online]. 18.07.2023.
Dostupno na:
https://learn.microsoft.com/en-us/dotnet/csharp/linq/

[8] C# Corner. Introduction To ASP.NET MVC [Online]. 11.12.2018.
Dostupno na:
https://www.c-sharpcorner.com/article/introduction-to-Asp-Net-mvc/

[9] Microsoft. ASP.NET MVC Overview [Online]. 16.06.2023.
Dostupno na:
https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview

[10] GeeksForGeeks. Benefit of using MVC [Online]. 15.04.2023.
Dostupno na:
https://www.geeksforgeeks.org/benefit-of-using-mvc/

[11] Entity Framework Tutorial. What is Entity Framework? [Online]. 15.04.2023.
Dostupno na:
https://www.entityframeworktutorial.net/what-is-entityframework.aspx

[12] Simplilearn. What Is C# Entity Framework? A Comprehensive Guide [Online]. 20.02.2023. Dostupno na:
https://www.simplilearn.com/tutorials/asp-dot-net-tutorial/entity-framework-in-c-sharp

[13] GeeksForGeeks. C# Multithreading [Online]. 22.02.2023.
Dostupno na:
https://www.geeksforgeeks.org/c-sharp-multithreading/

[14] Microsoft. Task-based asynchronous programming [Online]. 13.01.2023.
Dostupno na:
https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/task-based-asynchronous-programming

[15] Microsoft. Using threads and threading [Online]. 10.04.2022.
Dostupno na:
https://learn.microsoft.com/en-us/dotnet/standard/threading/using-threads-and-threading

[16] TechTarget. MQTT (MQ Telemetry Transport) [Online]. siječanj, 2021.
Dostupno na:
https://learn.microsoft.com/en-us/dotnet/standard/threading/using-threads-and-threading

[17] Amazon Web Service. What is MQTT? [Online]. 2023.
Dostupno na:
https://aws.amazon.com/what-is/mqtt/

9. OZNAKE I KRATICE

ASP.NET (engl. Active Server Pages Network Enabled Techonologies)

ASYNC (engl. Asynchronous)

BCL (engl. Base Class Library)

CIL (engl. Common Intermediate Language)

CLR (engl. Common Language Runtime)

CPU (engl. Central Processing Unit)

CRUD (engl. Create, Read, Update, Delete)

CSS (engl. Cascading Style Sheets)

DB (engl. Database)

ECTS (engl. European Credit Transfer and Accumulation System)

EF (engl. Entity Framework)

GC (engl. Garbage Collector)

GUI (engl. Graphical User Interface)

HTML (engl. Hypertext Markup Language)

IDE (engl. Integrated Development Environment)

IoT (engl. Internet of Things)

JIT (engl. Just-In-Time Compilation)

LINQ (engl. Language Integrated Query)

MVC (engl. Model-View-Controller)

MSIL (engl. Microsoft Intermediate Language)

MQTT (engl. Message Queuing Telemetry Transport)

NFC (engl. Near Field Communication)

OOP (engl. Object-Oriented Programming)

ORM (engl. Object-Relational Mapping)

RAM (engl. Random-Access Memory)

RFID (engl. Radio-Frequency Identification)

SQL (engl. Structured Query Language)

UI (engl. User Interface)

UUID (engl. Universally Unique Identifier)

VB.NET (engl. Visual Basic .NET)

VUB - Veleučilište u Bjelovaru

WEB (engl. World Wide Web)

XML (engl. Extensible Markup Language)

QR (engl. Quick Response)

10. SAŽETAK

Naslov: Primjena .NET tehnologije za sustavnu evidenciju nazočnosti studenata na nastavnim aktivnostima

Predmet ovog rada je kombinacija primjene .NET platforme i Internet of Things (IoT) tehnologije u automatiziranoj evidenciji prisutnosti na predmetima u obrazovanju. Implementirano je .NET rješenje koje omogućuje precizno praćenje prisutnosti učenika putem skeniranja UUID-ova pomoću IoT uređaja. Aplikacija, temeljena na .NET platformi, zamjenjuje tradicionalne metode evidencije prisutnosti poput ručnog bilježenja.

Uz primjenu IoT tehnologije, postiže se brza, pouzdana i precizna evidencija prisutnosti, smanjujući administrativne zadatke obrazovnih institucija. Integracija informacija dobivenih putem IoT uređaja, razvoj aplikacije i upravljanje bazom podataka bili su izazovi koji su uspješno prevladani kroz pažljivo planiranje i istraživanje.

U radu su analizirani dobiveni rezultati i razmatrana potencijalna unaprjeđenja sustava, poput sigurnosti, robusnosti i generiranja izvještaja o prisutnosti. Ova inovacija ima potencijal za unaprjeđenje sustava evidencije prisutnosti i poboljšanje učinkovitosti ne samo u obrazovnom sektoru, već i u drugim područjima.

Ključne riječi: .NET, IoT, UUID, bilježenje prisutnosti.

11. ABSTRACT

Title: Utilization of .NET technology for Systematic Attendance Recording of Students in Educational Activities

The subject of this paper is the combination of the .NET platform and Internet of Things (IoT) technology in automated attendance tracking in education. A .NET solution has been implemented to enable precise monitoring of student attendance through scanning UUIDs using IoT devices. The application, based on the .NET platform, replaces traditional attendance tracking methods such as manual recording.

By applying IoT technology, fast, reliable, and accurate attendance tracking is achieved, reducing administrative tasks for educational institutions. Integrating information obtained from IoT devices, developing the application, and managing the database posed challenges that were successfully overcome through careful planning and research.

The paper analyzes the obtained results and discusses potential system improvements, such as security, robustness, and generating attendance reports. This innovation has the potential to enhance attendance tracking systems and improve efficiency not only in the education sector but also in other fields.

Keywords: .NET, IoT, UUID, attendance monitoring.

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	Potpis studenta
U Bjelovaru, 5. RUJNA, 2023.	BRANIMIR FRŠKO	Branimir Frško

U skladu s čl. 58, st. 5 Zakona o visokom obrazovanju i znanstvenoj djelatnosti, Veleučilište u Bjelovaru dužno je u roku od 30 dana od dana obrane završnog rada objaviti elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru u nacionalnom repozitoriju.

Suglasnost za pravo pristupa elektroničkoj inačici završnog rada u nacionalnom repozitoriju

BRANIMIR FRČKO
ime i prezime studenta

Dajem suglasnost da tekst mojeg završnog rada u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu bude pohranjen s pravom pristupa (zaokružiti jedno od ponuđenog):

a) Rad javno dostupan
b) Rad javno dostupan nakon _____ (upisati datum)
c) Rad dostupan svim korisnicima iz sustava znanosti i visokog obrazovanja RH
d) Rad dostupan samo korisnicima matične ustanove (Veleučilište u Bjelovaru)
e) Rad nije dostupan

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 5. RUJNA, 2023.

Branimir Frčko
potpis studenta