Evidencija nazočnosti

Bernatović, Matej

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:370701

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

dabar

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I

INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

EVIDENCIJA NAZOČNOSTI

Završni rad

Matej Bernatović

Osijek, 2020

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek,

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime studenta:	Matej Bernatović
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4526, 23.10.2019.
OIB studenta:	63311878367
Mentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	
Član Povjerenstva 1:	Izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva 2:	
Naslov završnog rada:	Evidencija nazočnosti
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak završnog rada	Napraviti web aplikaciju za evidenciju nazočnosti studenata na svim oblicima nastave. Korisnici aplikacije bi imali različite uloge: student i nastavnik. Student bi imao uvid u sve svoje nazočnosti, a nastavnik bi mogao unositi i mijenjati podatke o nazočnosti. Podaci o predmetima, natavnicima, studentima, grupama i svemu ostalom bi se povukli iz vanjskog izvora. Predložena platforma bi bila .NET Core.
Prijedlog ocjene pismenog dijela ispita (završnog rada):	
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: bod/boda Postignuti rezultati u odnosu na složenost zadatka: bod/boda Jasnoća pismenog izražavanja: bod/boda Razina samostalnosti: razina
Datum prijedloga ocjene mentora:	

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 28.08.2020.

Ime i prezime studenta:	Matej Bernatović
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4526, 23.10.2019.
Turnitin podudaranje [%]:	

Ovom izjavom izjavljujem da je rad pod nazivom**: Evidencija nazočnosti**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Alfonzo Baumgartner

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 studenta:

Sadržaj

1. UVOD 1
1.1. Zadatak završnog rada 1
2. RAZVOJNO OKRUŽENJE 2
2.1. Programski jezik C# 2
2.2.1 Visual Studio 2019 3
2.2.1. ASP.NET 4
2.2.1.1. MVC 6
2.2 MSSQL Baza 7
2.2.1. Osnovne naredbe 7
2.2.2 ER dijagram baze 8
2.4. Bootstrap 8
2.5. jQuery 9
3. CLEAN CODE ARHITEKTURA 11
3.1. Karakteristika „clean code“ arhitekture 11
4. PRIKAZ PROJEKTNOG KODA 12
5. PRIKAZ APLIKACIJE 22
6. ZAKLJUČAK 32
LITERATURA 33
SAŽETAK 34
ŽIVOTOPIS 36
PRILOZI 37

1. UVOD

Tema ovog završnog rada je izrada web aplikacije za evidenciju nazočnosti. Aplikacija omogućava unos evidencije nazočnosti studenata na fakultetu odnosno pojedinom kolegiju. Profesor pomoću aplikacije ima mogućnost uz samo par klikova izvršiti evidenciju nazočnosti svih studenata, a student ima mogućnost nakon prijave vidjeti postotak svoje nazočnosti.

Cilj ovoga rada je da se automatizmom dodijele profesori i studenti u pripadajuće kolegije te nakon prijave profesora u aplikaciju omogućava mu se prikaz i uređivanje nazočnosti studenata na pripadajućem kolegiju profesora, dok se studentu omogućava prikaz njegove evidencije na svim njegovim pripadajućim kolegijima.

U drugome poglavlju pod nazivom „*Razvojno okruženje*“ objasniti će se objektno-orijentirani programski jezik C#, MSSQL baza, Boostrap te jQuery. Unutar programskog jezika C# detaljnije će se obraditi pomoćni alat Visual Studio 2019 i Framework ASP.NET te unutar ASP.NET-a pojasniti će se rad MVC-a. Kod MSSQL baze navesti će se njene osnovne naredbe te će se pojasniti ER dijagram samog projekta. Unutar trećega poglavlja pod nazivom „*Clean code arhitektura*“ objasniti će se princip rada navedene arhitekture te će se također navesti njezine karakteristike uz pomoć primjera. Četvrto poglavlje pod nazivom „*Prikaz projektnog koda*“ slikovito će prikazati kod samoga projekta te objasniti njegovo značenje. U petome poglavlju pod nazivom „*Prikaz aplikacije“* slikovito će prikazati korisnikovo korištenje web aplikacije te sve njegove mogućnosti koje web aplikacija dozvoljava.

1.1. Zadatak završnog rada

Stvoriti web aplikaciju koja će omogućiti lakšu kontrolu i unos evidencije nazočnosti studenta na određenom kolegiju. Testirati aplikaciju pomoću unosa testnih uzoraka te testirati njihov prikaz na istome. Tehnologija izrade web aplikacije je C# ASP.NET Core, MSSQL baza.

2. RAZVOJNO OKRUŽENJE

2.1. Programski jezik C#

Prema podacima koji se nalaze u članku [1] C# je objektno-orijentirani programski jezik koji sadrži funkcijsku, proceduralnu, generičku, deklarativnu, komponentno-orijentiranu i objektno-orijentiranu programsku paradigmu. Objektno-orijentirano programiranje (OOP) je način pisanja računalnih programa koristeći objekte za pohranu podataka i metode. Namjena C# je jednostavnost te se u današnje vrijeme koristi češće od drugih programski jezika. Svoj razvoj stekao je 2002. godine u tvrtki Microsoft koji je vodio Anders Hejlsberg. Inspiracija za ime „C Sharp" polazi od muzičke note u C duru koja asocira na četiri spojena plusa i tako implicira da je jezik C# inkrement jezika C++. Na temelju objektnih jezika C++, Java i Visual Basic nastao je C# koji je vrlo sličan C++ i Javi. C# kreiran je za izradu stolnih (desktop) i Internet aplikacija, te nije ovisan o platformi odnosno operativnom sustavu gdje aplikacija izrađena u C# može biti servirana na Windows-u , Linux-u te ostalim operativnim sustavima. Izvršavanje i izrada distribuiranih komponenti i servisa osnovne su namjene jezične infrastrukture (CLI) te pomoću toga je postignuto omogućavanje pisanih programa da djeluju zajedno u različitim jezicima [2]. Microsoftova implementacija CLI s dodatnim paketima za podršku korisničkih sučelja, XML, podataka, Web servisa i biblioteka temeljnih klasa sadrži .NET arhitektura. Četiri programska jezika koja isporučuju .NET Framework paket za razvoj softvera su : C++, C#, Visual Basic i F#. 2002. godine izdana je prva inačica a 2012. godine izdana je posljednja inačica. U tablici 1. prikazane su inačice programskog jezika C#, te pripadne inačice .NET Frameworka i Microsoftovog IDE-a Visual Studija.

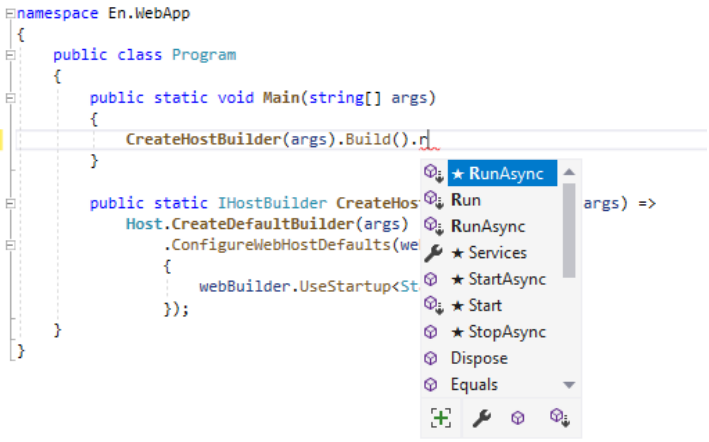
Tablica 1. Inačice C# programskog jezika i razvoj okruženja Visual Studija

C# inačica	Godina izdanja	.NET Framework	Visual Studio
C# 1.0	2002.	.NET Framework 1.0	Visual Studio .NET 2002
C# 1.2	2003.	.NET Framework 1.1	Visual Studio .NET 2003
C# 2.0	2005.	.NET Framework 2.0	Visual Studio 2005
C# 3.0	2007.	.NET Framework 3.0 .NET Framework 3.5	Visual Studio 2008 Visual Studio 2010
C# 4.0	2010.	.NET Framework 4.0	Visual Studio 2010
C# 5.0	2012.	.NET Framework 4.5	Visual Studio 2012 Visual Studio 2013

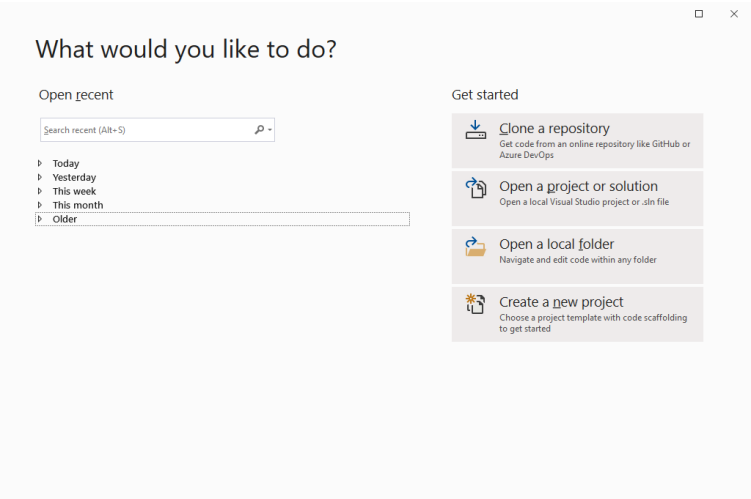
Izvor: C# in Depth, 2013.

2.2.1 Visual Studio 2019

Visual Studio 2019 (Slika 2.2.) koristi se za izradu aplikacija web stranica, web aplikacija i web servisa. Visual Studio sadrži uređivač teksta koji podržava refaktoriranje koda i *IntelliSense*. *IntelliSense* je pomoć za dovršavanje koda koja uključuje brojne značajke: članove popisa, informacije o parametru, brze informacije i cjelovitu riječ. Navedene značajke pomažu vam da saznate više o kodu koji koristite, pratite parametre koje upišete i dodate pozive u svojstva i metode uz samo nekoliko pritiska tipki. Na slici 2.1. prikazan je način rada *IntelliSense-a*.



Slika 2.1. Prikaz rada IntelliSense-a



Slika 2.2. Visual studio 2019

Programski jezici C, C++, C#, Visual Basic (VB.NET), F# neki su od jezika koje razvojno okruženje Visual Studio podržava, te ujedno omogućuje kompajliranje i debugiranje samih projekata te uređivanje samog koda.

Vrste paketa Visual Studija:

1. Visual Studio Express Edition – najosnovniji i besplatni paket
2. Ultimate
3. Premium
4. Professional
5. Test Professional
6. Team Fundation Server

2.2.1. ASP.NET

Tri različita programska okvira koja ASP.NET sadrži za razvoj web aplikacija su :

1. ASP.NET Web Pages
2. Web Forms
3. ASP.NET MVC

Centralna točka koja služi za pristup podacima putem interneta (API) dio je ASP.NET web okvira (eng. f*ramework*). Za razvoj web aplikacija koristi se HTML, CSS, i JavaScript programski jezici.

HTML (eng. *Hyper Text Markup Language)* standardni je jezik za stvaranje web stranica. Opisuje strukturu web stranice te se sastoji od niza elemenata koji govore pregledniku kako prikazati sadržaj. Struktura HTML elemenata prikazana je na slici 2.3.

```
<!DOCTYPE html>
<html>
<head>
<title>Naslov</title>
</head>
<body>

<h1>Prvi Heading</h1>
<p>Prvi paragraf.</p>

</body>
</html>
```

Slika 2.3. Prikaz strukture HTML-a

CSS (eng. *Cascading Style Sheets*) opisuje kako se HTML elementi trebaju prikazati na zaslonu, papiru ili na drugim medijima. Omogućava štednju posla te kontrolira izgled više web stranica odjednom. Vanjske tablice stilova pohranjuju se u CSS datotekama.

```
h1 {
    font-family: courier, courier-new, serif;
    font-size: 20pt;
    color: blue;
    border-bottom: 2px solid blue;
}
p {
    font-family: arial, verdana, sans-serif;
    font-size: 12pt;
    color: #6B6BD7;
}
.red_txt {
    color: red;
}
```

Slika 2.4. Prikaz CSS-a [4]

Izvor: https://agendamahala.com/introduction-to-css

JavaScript je programski jezik koji se koristi u web razvoju a izvorno ga je razvio Netscape kao sredstvo za dodavanje dinamičnih i interaktivnih elemenata na web stranice. Poput jezika skriptiranja na strani poslužitelja JavaScript kod se može umetnuti bilo gdje unutar HTML-a web stranice.

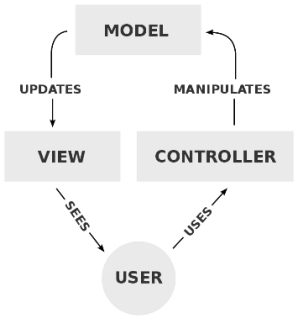
```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick="document.getElementById('demo').innerHTML = 'Hello JavaScript!'">Click Me!
</button>

</body>
</html>
```

Slika 2.5. Prikaz JavaScript koda [5]

Izvor : https://www.w3schools.com/js/js_intro.asp

2.2.1.1. MVC

Model-view-controller (MVC) je jedan od najpoznatijih programskih okvira za razvoj korisničkih sučelja koji dijeli povezanu programsku logiku na tri međusobno povezana elementa. Koristi se radi odvajanja unutarnjih prikaza informacija od načina na koji se informacije prezentiraju i prihvaćaju od korisnika. Svaki sloj aplikacije ima svoju odgovornost. Kada korisnik dođe na određenu stranicu on zapravo radi zahtjev prema serveru na način da navedeni zahtjev dolazi na kontroler te ga kontroler dalje obrađuje po logici koja je napisana unutar njega, tj. prosljeđuje podatke dalje u model i određene servise i vraća korisniku HTML odnosno prikaz stranice (view). Model je zapravo središnja komponenta uzorka, odnosno dinamička struktura podataka aplikacije koja je neovisna o korisničkom sučelju. Izravno upravlja podacima, logikom i pravilima aplikacije. Pregled je bilo koji prikaz informacija kao što su grafikon, dijagram ili tablica i ostalo. Omogućava višestruki prikaz istih podataka, kao na primjer trakasti grafikon za upravljanje i tablični prikaz za računovođe [6].



Slika 2.6. Tijek obrade MVC-a

Izvor : https://dotnet.microsoft.com/apps/aspnet/mvc

2.2 MSSQL Baza

MSSQL je paket softvera za baze podataka koji je objavio Microsoft i koji se široko koristi u poduzećima. Uključuje mehanizam relacijske baze podataka koji pohranjuje podatke u tablice, stupce i retke, Integration Services (SSIS) alat je za kretanje podataka za uvoz, izvoz i transformiranje podataka, Reporting Services (SSRS) se koristi za stvaranje izvješća i posluživanje izvješća krajnjim korisnicima, a Services Services (SSAS) je višedimenzionalna baza podataka koja se koristi za pretraživanje podataka iz glavnog pokretača baze podataka. Omogućuje korisnicima analizu podataka, predviđanje prodaje, pa čak i predviđanje ponašanja kupaca pomoću analitike poslovne inteligencije [7].

2.2.1. Osnovne naredbe

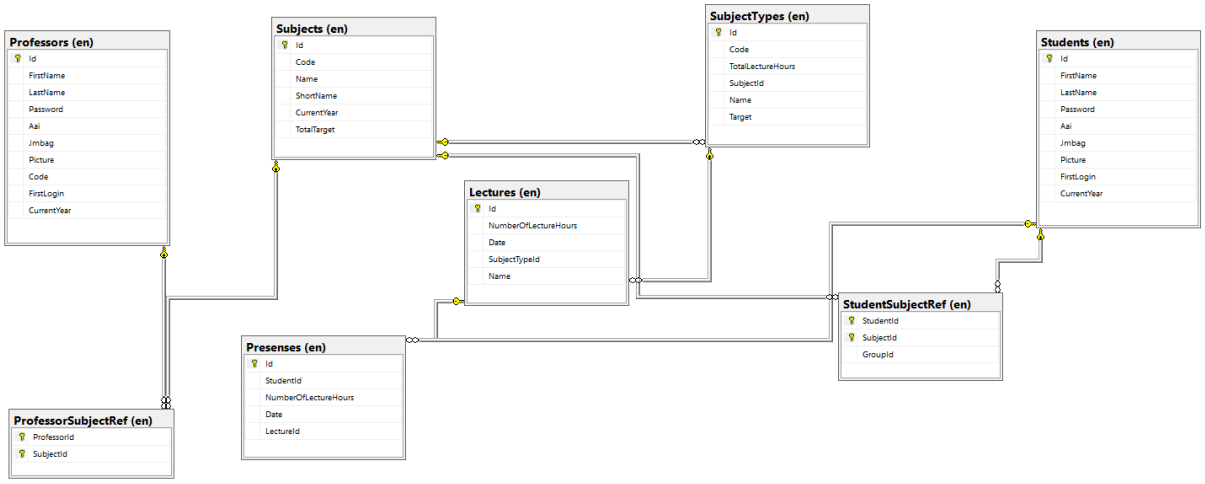
Microsoft SQL Management Studio (SSMS) je alat koji se može koristiti za izradu Microsoft baza pomoću naredbi. Osnovne naredbe SSMS-a su [8] :

1. SELECT – povlači podatke iz baze podataka
2. UPDATE – ažurira podatke u bazi podataka
3. DELETE – briše podatke iz baze podataka
4. INSERT INTO – ubacuje nove podatke u bazu podataka
5. CREATE DATABASE – stvara novu bazu podataka
6. ALTER DATABASE – mijenja bazu podataka
7. CREATE TABLE – stvara novu tablicu

8. ALTER TABLE – mijenja postojeću tablicu
9. DROP TABLE – briše tablicu
10. CREATE INDEX - stvara indeks (ključ za pretraživanje)
11. DROP INDEX - briše indeks

2.2.2 ER dijagram baze

ER dijagram (eng. *Entity Relationship diagram*) prikazuje veze skupova entiteta pohranjenih u bazi podataka. Relacije među entitetima mogu biti unarne, binarne ili n-arne. Unarna relacija predstavlja jedan na prema jedan relaciju između entiteta. Relacija je binarna ako sudjeluju dva entiteta, a n-arna ako postoji više od dva [9]. Entitet u ovom kontekstu je objekt, komponenta podataka. Skup entiteta je zbirka sličnih entiteta koji mogu imati atribute koji definiraju njegova svojstva. Definiranjem entiteta, njihovih atributa i prikazom odnosa između njih, ER dijagram ilustrira logičku strukturu baza podataka [10]. Na slici 7. prikazan je ER dijagram izrađene aplikacije.



Slika 2.7. ER dijagram projekta

2.4. Bootstrap

Bootstrap je jedan od najvažnijih alata za front-end programere. Stvorili su ga bivši zaposlenici Twittera Mark Otto i Jacob Thornton. Bootstrap je ogromna kolekcija praktičnih, višekratnih dijelova koda napisanih u HTML-u, CSS-u i JavaScript-u. To je ujedno i napredni

razvojni okvir koji programerima i dizajnerima omogućuje brzu izradu web-lokacija koje u potpunosti reagiraju [11].

Prednosti bootstrapa su :

- Nema više trošenja sati na kodiranje vlastite mreže - dolazi sa svojim unaprijed definiranim mrežnim sustavom
- dolazi s vlastitim kodom za automatski mijenjanje veličine slika na temelju trenutne veličine zaslona
- dolazi s cijelim nizom komponenata koje možete lako pričvrstiti na svoju web stranicu
- također omogućuje programerima da iskoriste preko desetak prilagođenih JQuery dodataka.
- Svaki komad koda detaljno je opisan i objašnjen na njihovoj web stranici.

2.5. jQuery

jQuery je brza, mala i JavaScript značajka bogata značajkama. Čini stvari poput HTML obilaženja i manipulacije dokumentima, rukovanja događajima, animacije i Ajaxa mnogo jednostavnijim s API-jem jednostavnim za upotrebu koji radi u mnoštvu preglednika. Kombinacijom svestranosti i proširivosti, jQuery je promijenio način na koji milijuni ljudi pišu JavaScript [12].

Kratki pregled:

1. DOM preokret i manipulacija

```
1  $( "button.continue" ).html( "Next Step..." )
```

Slika 2.8. Prikaz jQuery DOM manipulacije

Na slici 2.8. prikazano je kako se uzima element <button> koji ima klasu „continue“ te postavlja vrijednost HTML-a na „Next Step…“

2. Rukovanje događajima

```
1  var hiddenBox = $( "#banner-message" );
2  $( "#button-container button" ).on( "click", function( event ) {
3    hiddenBox.show();
4  });
```

Slika 2.9. Prikaz jQuery rukovanje događajima

Na slici 2.9. prikazano je kako klikom na gumb se prikazuje „#banner-message“ element

3. Ajax

```
$.ajax({
  url: "/api/getWeather",
  data: {
    zipcode: 97201
  },
  success: function( result ) {
    $( "#weather-temp" ).html( "<strong>" + result + "</strong> degrees" );
  }
});
```

Slika 2.10. Prikaz jQuery Ajax poziva

Na slici 2.10. prikazano je kako se šalje Ajax poziv prema API-ju koja sadrži određene podatke (zipcode : 97201) te nakon toga dobiva rezultat i prikazuje određeni HTML.

3. CLEAN CODE ARHITEKTURA

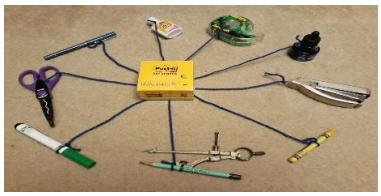
Arhitektura označava cjelokupni dizajn projekta. To je organizacija koda u klase, datoteke, komponente ili module gdje su skupine koda međusobno povezane. Arhitektura definira gdje aplikacija izvodi svoju osnovnu funkcionalnost i kako ta funkcionalnost utječe na stvari poput baze podataka i korisničkog sučelja [13].

3.1. Karakteristika „clean code“ arhitekture

Na slici 3.1. prikazana je ilustracija „spaghetti code-a“ gdje možemo uočiti kako razni predmeti predstavljaju komponente, odnosno klase, koje su međusobno ovisne. Slika prikazuje nepravilan raspored koji ne sadrži redoslijed postavljenih predmeta te bilo kakve izmjene zahtijevaju veći napor pri izvršavanju željenog zadatka.

Slika 3.1. Ilustracija „*spaghetti code*-a“

Na slijedećoj slici prikazana je ilustracija „*clean code-a*“ gdje možemo uočiti kako predmeti koji predstavljaju komponente, odnosno klase, koje nisu međusobno ovisne nego ovise o jednoj centralno komponenti koja spaja ostale jedinice.

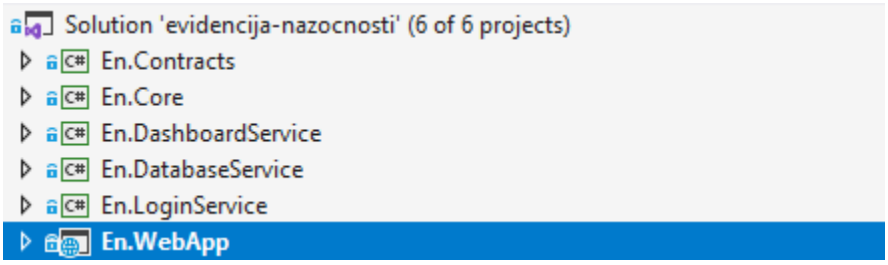


Slika 3.2. Ilustracija „clean code-a“

Iz gore navedenih slika može se uočiti kako kod „clean code“ arhitekture zamjena bilo kakve komponente je vrlo jednostavna. Uzmimo za primjer škare gdje na slici 3.1. ovise o olovci, boci s tintom, vrpci i kompasu te da bi zamijenili škare s nečim drugim trebamo maknuti i ostale predmete te zamijeniti pa vratiti predmete. Dok na slici 3.2. možemo vidjeti kako škare ovise jedino o bloku i kako možemo škare jednostavno izvaditi i zamijeniti s željenim predmetom.

4. PRIKAZ PROJEKTNOG KODA

Za izradu web aplikacije korišten je program „Visaul Studio“. Izrađeni projekt sadrži 6 komponenti gdje svaka od njih ima svoj zadatak što odrađuje. Na slici 4.1. možemo vidjeti sve komponente projekta.



Slika 4.1. Prikaz komponenti projekta

1. En.Webapp – glavna komponenta, aplikacija koja izrađuje prikaz aplikacije
2. En.LoginService – komponenta koja sadrži provjeru prilikom prijave korisnika
3. En.DatabaseService – komponenta koja sadrži sve detalja o bazi
4. En.DashboardService – komponenta koja sadrži svu logiku aplikacije
5. En.Core –zajednička komponenta
6. En.Contracts – komponenta koja sadrži modele koji se koriste kroz aplikaciju

Na slici 4.2. prikazuje se metoda za prijavljivanje profesora u aplikaciju, gdje se provjerava postoje li podaci u bazi s određenim AAI i lozinkom te dali se profesor prvi put prijavljuje.

```
public async Task<bool> ProfessorLogin(string aai, string password)
{

    var professor = await this.context.Professors.FirstOrDefaultAsync(p => p.Aai == aai && p.Password == password);

    if (professor == null)
        return false;

    this.memoryCache.Set("aai", aai);
    return true;
}
public async Task<bool> CheckIsFristProfessorLogin(string aai)
{
    var professor = await this.context.Professors.FirstOrDefaultAsync(p => p.Aai == aai && p.FirstLogin == true);

    if (professor != null)
    {
        return true;
    }

    return false;
}
```

Slika 4.2. Prikaz metode prijave profesora

Vidljivo je kako se u prvoj metodi radi LINQ upit na bazu koji dohvaća prvi rekord u bazu s traženim podacima te ako ništa ne pronađe vraća vrijednost „null“. Druga metoda prikazuje provjeru u bazi gdje se traži rekord s atributom „FirstLogin“ gdje vrijednost mora biti istinita. U slučaju ne pronađenog rekorda metoda vraća vrijednost laž.

Na slici 4.3. prikazane su metode za dohvaćanje podataka koje se koristite na profesorovoj i studentovoj naslovnici.

```
public async Task<Dashboard> GetStudentDashboardDataAsync(string aai)
{
    var response = new Dashboard();

    var student = await this.context.Students.FirstOrDefaultAsync(x => x.Aai == aai);
    if (student != null)
    {
        response.Student = mapper.Map<Contracts.Student>(student);
    }

    response.Subjects = mapper.Map<List<Subject>>(
        await this.context.Subjects
        .Include(x => x.SubjectTypes)
        .ThenInclude(s => s.Lectures)
        .ThenInclude(p => p.Presenses)
        .Where(x => x.Students.FirstOrDefault(t => t.Student == student) != null)
        .ToListAsync());

    return response;
}

public async Task<Dashboard> GetProfessorDashboardDataAsync(string aai)
{
    var response = new Dashboard();
    var professor = await this.context.Professors.FirstOrDefaultAsync(x => x.Aai == aai);
    if(professor != null)
    {
        var subjects = mapper.Map<List<Subject>>(
            await this.context.Subjects
            .Where(x => x.Professors.FirstOrDefault(t => t.Professor == this.context.Professors.FirstOrDefault(s => s.Aai == aai)) != null)
            .ToListAsync());
        var students = new List<DashboardStudent>();
        foreach(var subject in subjects)
        {
            students.AddRange(await this.subjectService.GetDashboardStudentsByIdAsync(subject.Id));
        }

        response.Subjects = subjects;
        response.DashboardStudents = students;
    }

    return response;
}
```

Slika 4.3. Prikaz metode dohvaćanja podataka za naslovnicu

Prva metoda „GetStudentDashboardDataAsync“ prima parametar AAI što se odnosi na studenta i prvo radi LINQ upit na bazu da pronađe studenta koji ima taj AAI, dok se u drugom dijelu radi LINQ upit na tablicu „Subjects“ i traži se podaci svih vezanih tablica gdje je pronađeni student aktivan. Isto tako se radi i kod druge metode „GetProfessorDashboardDataAsync“ samo što se odnosi na profesora.

Slika 4.4. predstavlja dohvaćanje studenata pomoću jedinstvenog broja kolegija.

```
public async Task<List<Student>> GetStudentsBySubjectIdAsync(int subjectId)
{
    var response = new List<Student>();
    var subject = await this.context.Subjects.Include(x => x.Students).ThenInclude(p => p.Student).FirstOrDefaultAsync(x => x.Id == subjectId);
    return mapper.Map<List<Student>>(subject.Students.Select(x => x.Student));
}
```

Slika 4.4. Prikaz dohvaćanja studenta

Metoda „GetStudentsBySubjectIdAsync“ radi upit na bazu te dohvaća prvi kolegij iz tablice „Subjects“ pomoću jedinstvenog broja kolegija, te se uzimaju svi studenti koji su upisani na odabrani kolegij.

Slika 4.5. predstavlja kreiranje novog predavanja te dodavanje svih studenata koji su bili na tom predavanju.

```
public async Task<bool> SaveLecture(LectureModel model)
{
    var presenses = new List<DatabaseService.Models.Presense>();
    var lecture = new DatabaseService.Models.Lecture();
    lecture.Date = model.Date;
    lecture.Name = model.Name;
    lecture.NumberOfLectureHours = model.NumberOfHours;
    lecture.SubjectType = await this.context.SubjectTypes.FirstOrDefaultAsync(x => x.Id == model.SubjectTypeId);
    if (model.Students != null)
    {
        foreach (var student in model.Students)
        {
            presenses.Add(new DatabaseService.Models.Presense()
            {
                Date = model.Date,
                NumberOfLectureHours = model.NumberOfHours,
                StudentId = student.Id,
                Lecture = lecture
            });
        }
    }
    lecture.Presenses = presenses;

    await this.context.Lectures.AddAsync(lecture);

    return await this.context.SaveChangesAsync() == 1;
}
```

Slika 4.5. Prikaz rada s bazom za kreiranje predavanja

Metoda „SaveLecture“ kreira novi rekord u tablici „Lectures“ što predstavlja novo predavanje te se dodaju svi studenti koji su sudjelovali na tom predavanju ovisno o proslijeđenom parametru.

Slika 4.6. prikazuje rad s bazom, odnosno kako se ažurira te briše rekord iz baze.

```
public async Task<bool> UpdateLecture(LectureModel model)
{
    var presenses = new List<DatabaseService.Models.Presense>();
    var lecture = await this.context.Lectures.FirstOrDefaultAsync(x => x.Id == model.Id);
    lecture.Date = model.Date;
    lecture.Name = model.Name;
    lecture.NumberOfLectureHours = model.NumberOfHours;
    this.context.Lectures.Update(lecture);

    var pres = await this.context.Presenses.Where(x => x.LectureId == model.Id).ToListAsync();
    this.context.Presenses.RemoveRange(pres);
    foreach (var student in model.Students)
    {
        presenses.Add(new DatabaseService.Models.Presense()
        {
            Date = model.Date,
            NumberOfLectureHours = model.NumberOfHours,
            StudentId = student.Id,
            Lecture = lecture
        });
    }
    await this.context.Presenses.AddRangeAsync(presenses);
    return await this.context.SaveChangesAsync() == 1;
}

public async Task<bool> DeleteLecture(int lectureId)
{
    var lecture = await this.context.Lectures.FirstOrDefaultAsync(x => x.Id == lectureId);

    if(lecture != null)
    {
        var pres = await this.context.Presenses.Where(x => x.LectureId == lecture.Id).ToListAsync();
        this.context.Presenses.RemoveRange(pres);
        this.context.Lectures.Remove(lecture);
        var save = await this.context.SaveChangesAsync();
        return true;
    }

    return false;
}
```

Slika 4.6. Prikaz rada s bazom za ažuriranje i brisanje

Postoje dvije metode, prva „UpdateLecture“ metoda predstavlja ažuriranje predavanja, samim time što se predavanje ažurira, ažurira se i tablica „student“. Druga „DeleteLecture“ metoda predstavlja brisanje predavanja odnosno uklanjanje iz baze, isto tako se brišu i određeni studenti koji su dodani na određeno predavanje.

Slika 4.7. predstavlja metodu koja promjenjuje lozinku korisnika.

```
public async Task<bool> ChangeProfilePassword(string password, string user, bool isStudent)
{
    if (isStudent)
    {
        var student = await this.context.Students.FirstOrDefaultAsync(x => x.Aai == user);
        if (student != null)
        {
            student.Password = HashPassword.Hash(password);
            this.context.Students.Update(student);
            return await this.context.SaveChangesAsync() == 1;
        }
        else
        {
            return false;
        }
    }
    else
    {
        var professor = await this.context.Professors.FirstOrDefaultAsync(x => x.Aai == user);
        if(professor != null)
        {
            professor.Password = HashPassword.Hash(password);
            this.context.Professors.Update(professor);

            return await this.context.SaveChangesAsync() == 1;
        }
        else
        {
            return false;
        }
    }
}
```

Slika 4.7. Prikaz metode za promjenu lozinke

Možemo uočiti kako se u metodi za promjenu lozinke koristi metoda Hash (slika 4.8.) koja specijalno kodira lozinku te se tako kodirana lozinka sprema u bazu.

```
public class HashPassword
{
    public static string Hash(string value)
    {
        if (value == null)
        {
            return null;
        }
        else
        {
            return Convert.ToBase64String(
                System.Security.Cryptography.SHA256.Create()
                    .ComputeHash(Encoding.UTF8.GetBytes(value))
            );
        }
    }
}
```

Slika 4.8. Prikaz metode za kriptiranje lozinke

Metoda „Hash“ prima parametar koji kriptira pomoću SHA256 algoritma za kriptiranje.

Slika 4.9. prikazuje HTML kod za listu svih kreiranih predavanja.

```
1   @model En.WebApp.Models.LectureListViewModel
2   @{
3       Layout = "_DashboardLayout";
4       ViewData["Title"] = "LectureList";
5   }
6
7   <div class="m-grid__item m-grid__item--fluid m-wrapper">
8       <div class="m-subheader ">
9           <div class="d-flex align-items-center">
10              <div class="mr-auto">
11                  <h3 class="m-subheader__title m-subheader__title--separator">@Model.Name</h3>
12                  <ul class="m-subheader__breadcrumbs m-nav m-nav--inline">
13                      <li class="m-nav__item m-nav__item--home">
14                          <a href="@Url.RouteUrl("professor-dashboard-get")" class="m-nav__link m-nav__link--icon">
15                              <i class="m-nav__link-icon la la-home"></i>
16                          </a>
17                      </li>
18                      <li class="m-nav__separator">-</li>
19                      <li class="m-nav__item">
20                          <a href="@Url.RouteUrl("get-professor-subject",new { subjectId=Model.SubjectId})" class="m-nav__link">
21                              <span class="m-nav__link-text">@Model.Name</span>
22                          </a>
23                      </li>
24                      <li class="m-nav__separator">-</li>
25                      <li class="m-nav__item">
26                          <a href="" class="m-nav__link">
27                              <span class="m-nav__link-text">@Model.SubjectType.Name</span>
28                          </a>
29                      </li>
30                  </ul>
31              </div>
32              <div>
33                  <div class="m-dropdown m-dropdown--inline m-dropdown--arrow m-dropdown--align-right m-dropdown--align-push">
34                      <a href="@Url.RouteUrl("new-lecture",new { subjectId = Model.SubjectId , subjectTypeId = Model.SubjectType.Id})"
35                         class="m-portlet__nav-link btn btn-lg btn-secondary  m-btn m-btn--outline-2x m-btn--air m-btn--icon m-btn--icon-only m-btn--pill  m-dropdown__toggle">
36                          <i class="la la-plus"></i>
37                      </a>
38                  </div>
39              </div>
40          </div>
41
42      </div>
43      <div class="m-content">
44          <div class="m-portlet">
45              <div class="m-portlet__body">
46                  @await Component.InvokeAsync("LectureList", new { subjectTypeId = Model.SubjectType.Id })
47              </div>
48          </div>
49      </div>
50  </div>
```

Slika 4.9. Prikaz HTML koda

Možemo primijetiti kako linija 46 koristi asinkronu metodu što je zapravo „View component“ u koju se prosljeđuju određeni parametri te ona ima svoju logiku (slika 4.10.) i prikazuje svoj HTML kod (slika 4.11).

```
using En.WebService;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace En.WebApp.ViewComponents
{
    public class LectureListViewComponent : ViewComponent
    {
        private readonly ILectureService lectureService;

        public LectureListViewComponent(ILectureService lectureService)
        {
            this.lectureService = lectureService;
        }

        public async Task<IViewComponentResult> InvokeAsync(int subjectTypeId)
        {
            var lecture = await this.lectureService.GetLectures(subjectTypeId);
            return View(lecture);
        }
    }
}
```

Slika 4.10. Logika „View Component“

```
@model IEnumerable<En.Contracts.Lecture>
@{
    ViewData["Title"] = "Default";
}


<table class="table m-table m-table--head-separator-primary">
    <thead>
        <tr>
            <th>#</th>
            <th>Datum</th>
            <th>Naziv Predavanja</th>
            <th>Broj Sati</th>
            <th>Akcije</th>
        </tr>
    </thead>
    <tbody>
        @if (Model.Any())
        {
            @foreach (var item in Model.OrderBy(x => x.Date))
            {
                <tr>
                    <th scope="row">@(Model.ToList().IndexOf(item)+1)</th>
                    <td>@item.Date.ToString("dd/MM/yyyy")</td>
                    <td>@item.Name</td>
                    <td>@item.NumberOfLectureHours</td>
                    <td>
                        <span class="dropdown">
                            <a href="#" class="btn m-btn m-btn--hover-brand m-btn--icon m-btn--icon-only m-btn--pill" data-toggle="dropdown" aria-expanded="false">
                                <i class="la la-ellipsis-h"></i>
                            </a>
                            <div class="dropdown-menu dropdown-menu-right" x-placement="bottom-end" style="position: absolute; transform: translate3d(-196px, 27px, 0px); top: 0px
                                <a class="dropdown-item" href="@Url.RouteUrl("view-lecture",new { lectureId=item.Id})"><i class="la la-info-circle"></i> Detalji</a>
                                <a class="dropdown-item" href="@Url.RouteUrl("edit-lecture",new { lectureId=item.Id})"><i class="la la-edit"></i> Uredi</a>
                                <a class="dropdown-item" onclick="return confirm(this,@item.Id)"><i class="la la-times-circle"></i> Obriši</a>
                            </div>
                        </span>
                    </td>
                </tr>
            }
        }

    </tbody>
</table>
```

Slika 4.11. HTML kod „View Component“

Na slici 4.12. prikazan je rad kontrolera.

```
42          [HttpGet("dashboard", Name = "student-dashboard-get")]
43          public async Task<IActionResult> Index()
44          {
45              return View(await this.dashboardService.GetStudentDashboardDataAsync(AAI));
46          }
47
48          [HttpGet("subject", Name = "get-student-subject")]
49          public async Task<IActionResult> Subject([FromQuery]int subjectId) {
50              if (AAI != null)
51              {
52                  return View(await this.presenseService.PresenceStatisticByAai(AAI, subjectId));
53              }
54              else
55              {
56                  return View(new Contracts.PresenseStatistic() { Success = false });
57              }
58          }
59
60          [HttpGet("profile",Name ="student-profile")]
61          public async Task<IActionResult> Profile()
62          {
63              return View(await profileService.GetStudentProfile(AAI));
64          }
65
66          [HttpPost("profile/password",Name ="student-change-password")]
67          public async Task<JsonResult> ProfilePassword([FromForm]string password, [FromForm]string oldPassword)
68          {
69              if(await profileService.CheckCurrentPassword(oldPassword, AAI, true))
70              {
71                  if(await profileService.ChangeProfilePassword(password, AAI, true))
72                  {
73                      return Json(new { Success = true, Message = "Lozinka je uspješno promjenjena." });
74                  }
75                  else
76                  {
77                      return Json(new { Success = false, Message = "Dogodila se pogreška, pokušajte kasnije." });
78                  }
79              }
80              else
81              {
82                  return Json(new { Success = false, Message = "Stara lozinka nije ispravna !" });
83              }
84
85          }
```

Slika 4.12. Primjer metoda na kontroleru

Postoji više metoda u kontroleru. Možemo uočiti kako svaka metoda ima oznaku „HttpGet“ ili „HttpPost“ što označava dali server dohvaća podatke ili prima podatke te na temelju toga vraća određenu akciju. Prema slici 4.12. vidimo da metoda „Index“ ima oznaku „HttpGet“ te ona vraća „View“ odnosno HTML kod koji se kod klijenta servira, dok metoda „ProfilePicture“ ima oznaku „HttpPost“ i prima određene parametre iz forme i vraća „JsonResult“.

Na slici 4.13. prikazana je autorizacija korisnika.

```
50          services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
51                  .AddCookie(CookieAuthenticationDefaults.AuthenticationScheme, options =>
52                  {
53                      options.Events.OnRedirectToLogin = (context) =>
54                      {
55
56
57                          var redirectUri = new Uri(context.RedirectUri);
58
59                          context.RedirectUri = $"{redirectUri.Scheme}://{redirectUri.Host}:{redirectUri.Port}{redirectUri.Query}";
60                          context.Response.StatusCode = StatusCodes.Status302Found;
61                          context.Response.Redirect($"{redirectUri.Scheme}://{redirectUri.Host}:{redirectUri.Port}{redirectUri.Query}");
62
63                           return Task.CompletedTask;
64
65                      };
66
67                      options.Events.OnRedirectToAccessDenied = (context) => {
68                          var redirectUri = new Uri(context.RedirectUri);
69                          context.RedirectUri = $"{redirectUri.Scheme}://{redirectUri.Host}:{redirectUri.Port}{redirectUri.Query}";
70                          context.Response.StatusCode = StatusCodes.Status401Unauthorized;
71                          context.Response.Redirect($"{redirectUri.Scheme}://{redirectUri.Host}:{redirectUri.Port}{redirectUri.Query}");
72                          return Task.CompletedTask;
73
74                      };
75
76                      options.AccessDeniedPath = new PathString("/access-denied");
77                  });
78
79          services.AddAuthorization(o =>
80          {
81              o.AddPolicy("ProfessorPolicy", configure => configure.RequireRole(Constants.Professor));
82              o.AddPolicy("StudentPolicy", configure => configure.RequireRole(Constants.Student));
83          });
```

Slika 4.13. Logika „cookie“ autorizacije

Svaki zahtjev na sustav prolazi kroz autorizaciju koja koristi kolačiće (eng. Cookie) te se određuje čemu logirani korisnik može pristupati. Iz slike 4.14. možemo vidjeti kako prilikom prijave korisnika u sustav, svakom prijavom dodjeljuju se „Claimovi“ te se koristi pomoćna metoda „SignInAsync“ koja koristi shemu za autoriziranje korisnika pomoću kolačića, dok iz slike 4.15. možemo uočiti kako se kontroleru dodjeljuje atribut „Authorize“ koji označava koji se „Policy“ treba koristiti i time se određuje čemu korisnik može pristupiti. Na slici 4.13. liniji 81 i 82 vidimo kako se „Policy“ kreira i možemo uočiti da na primjer „ProfessorPolicy“ zahtjeva da prijavljeni korisnik ima na sebi Rolu „PROFESSOR“.

```
private async Task Loggin(string aai, bool isStudent)
{
    var authProperties = new AuthenticationProperties() { IsPersistent = true };
    var claims = new List<Claim>();

    if (isStudent)
    {
        claims.Add(new Claim(Constants.StudentAai, aai));
        claims.Add(new Claim(Constants.Role, Constants.Student));
    }
    else
    {
        claims.Add(new Claim(Constants.ProfessorAai, aai));
        claims.Add(new Claim(Constants.Role, Constants.Professor));
    }

    var userIdentity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);
    ClaimsPrincipal principal = new ClaimsPrincipal(userIdentity);
    await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, principal, authProperties);
}
```

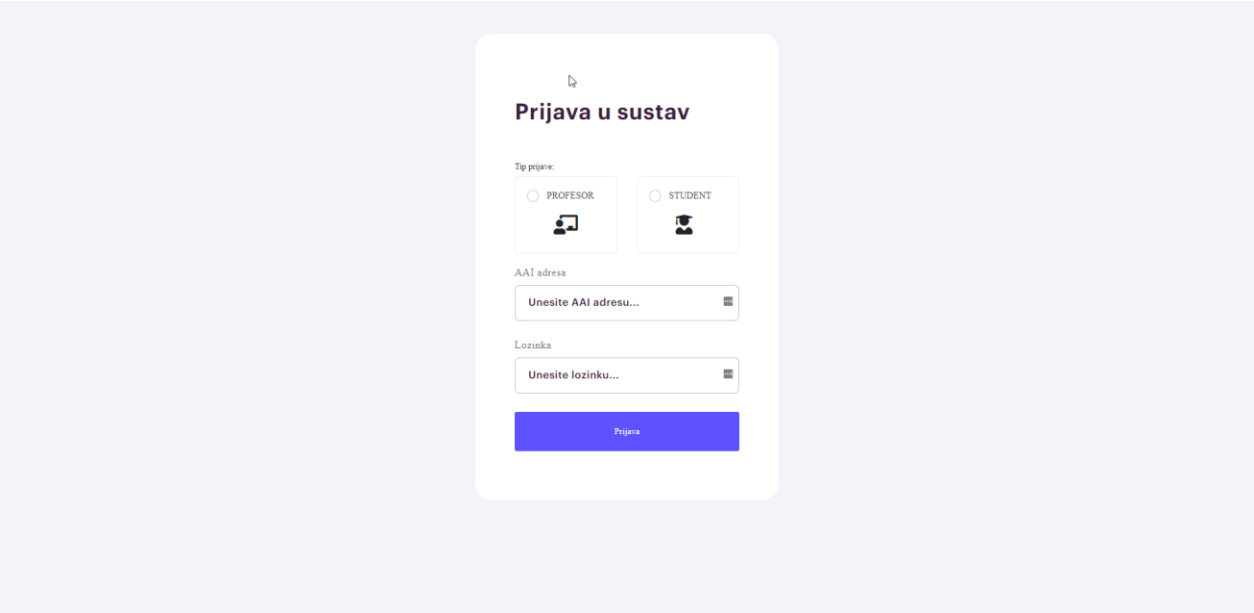
Slika 4.14. Metoda za dodjeljivanje „Claimova“ i prijavljivanje korisnika u sustav

```
namespace En.Web.Controllers
{
    public class HomeController : Controller
    {

namespace En.Web.Controllers
{
    [Authorize(Policy = "ProfessorPolicy")]
    [Route("professor")]
    public class ProfessorDashboardController : Controller

namespace En.Web.Controllers
{
    [Authorize(Policy = "StudentPolicy")]
    [Route("student")]
    public class StudentDashboardController : Controller
```

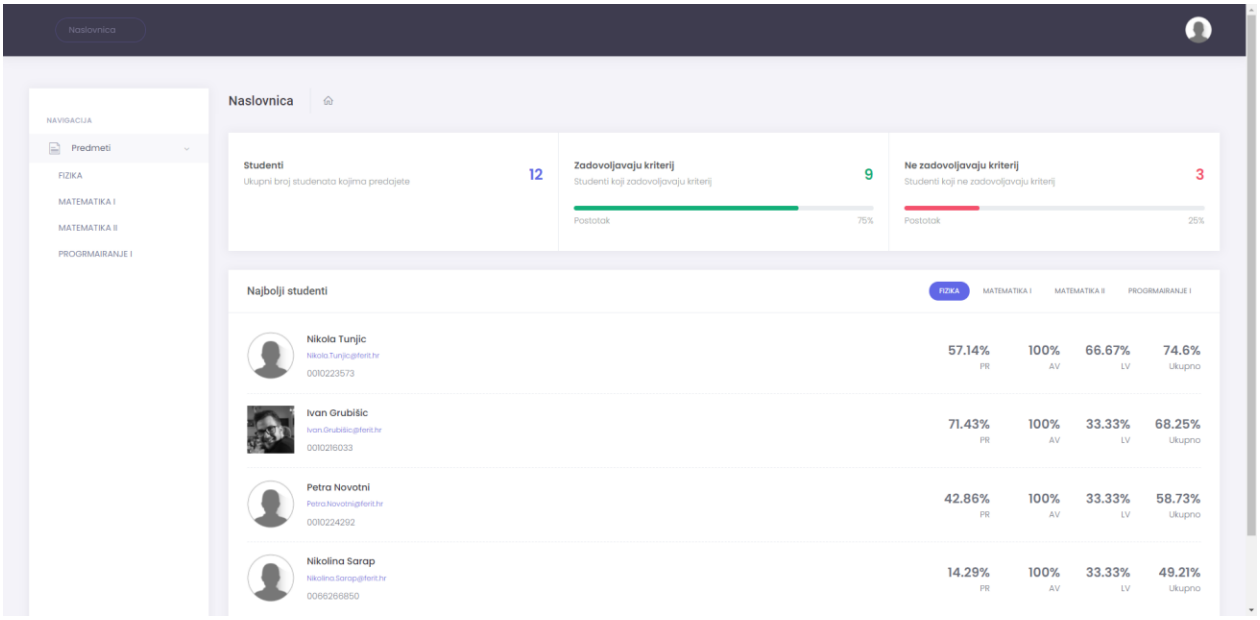
Slika 4.15. Prikaz „Authorize“ atributa na kontroleru

5. PRIKAZ APLIKACIJE

Aplikacija za evidenciju nazočnosti pri pokretanju automatski otvara početnu stranicu (Slika 5.1.) za prijavu korisnika u sustav. Postoji mogućnost prijave na 2 načina ,odnosno kao student ili kao profesor. Prilikom prijave mora se odabrati tip prijave , unijeti AAI adresa te lozinka. Sustav izbacuje greške ukoliko nešto od navedenog nije uneseno ili ako je lozinka pogrešna. Svi podaci prikazani na slikama su testni podaci s par uzoraka.



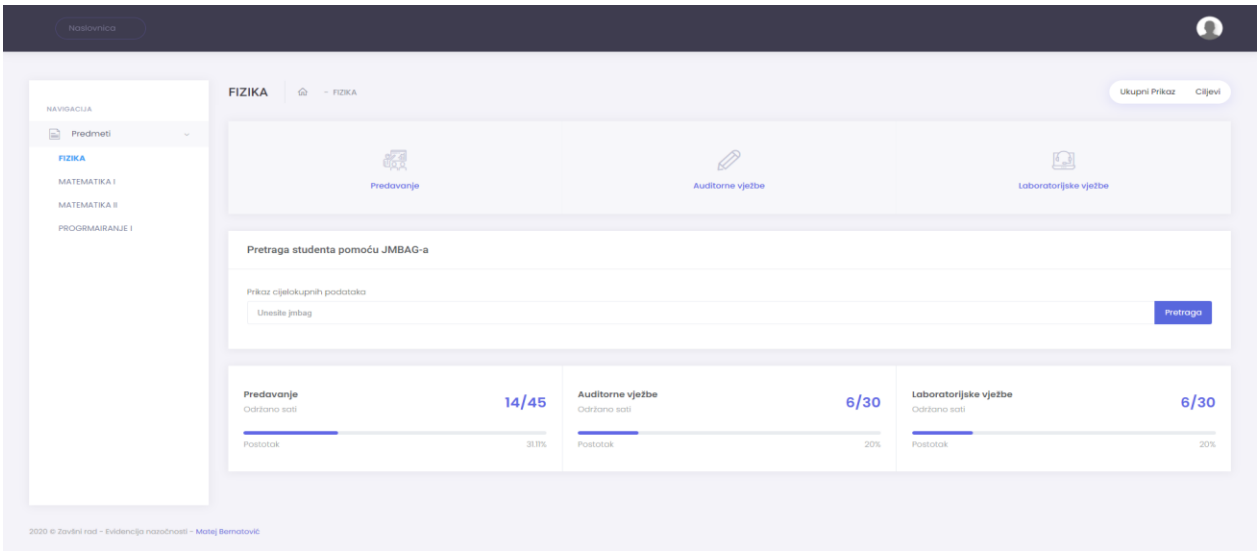
Slika 5.1. Početna stranica



Slika 5.2. Naslovnica profesora

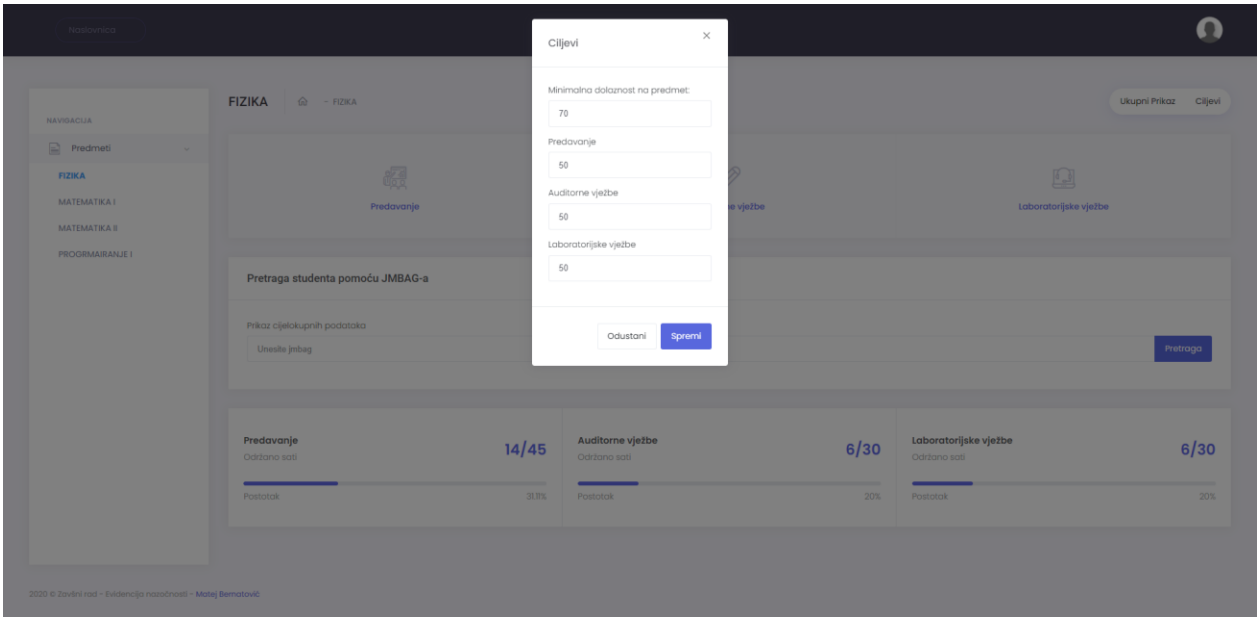
Naslovnica prikazuje statistiku ukupnog broja studenata, broj studenta koji zadovoljavaju kriterij, broj studenata koji ne zadovoljavaju kriterij te je vidljiva tablica koja prikazuje prvih deset studenata po ukupno postotku nazočnosti za svaki predmet u koji je profesor upisan. Sa lijeve strane navigacije možemo uočiti predmete koji pripadaju prijavljenom profesoru. Svaki prijavljeni profesor vidi samo svoje kolegije i studente koji pripadaju određenom kolegiju.

Na sljedećoj slici (Slika 5.3.) prikazan je određeni kolegij te statistički podaci.



Slika 5.3. Prikaz određenog kolegija

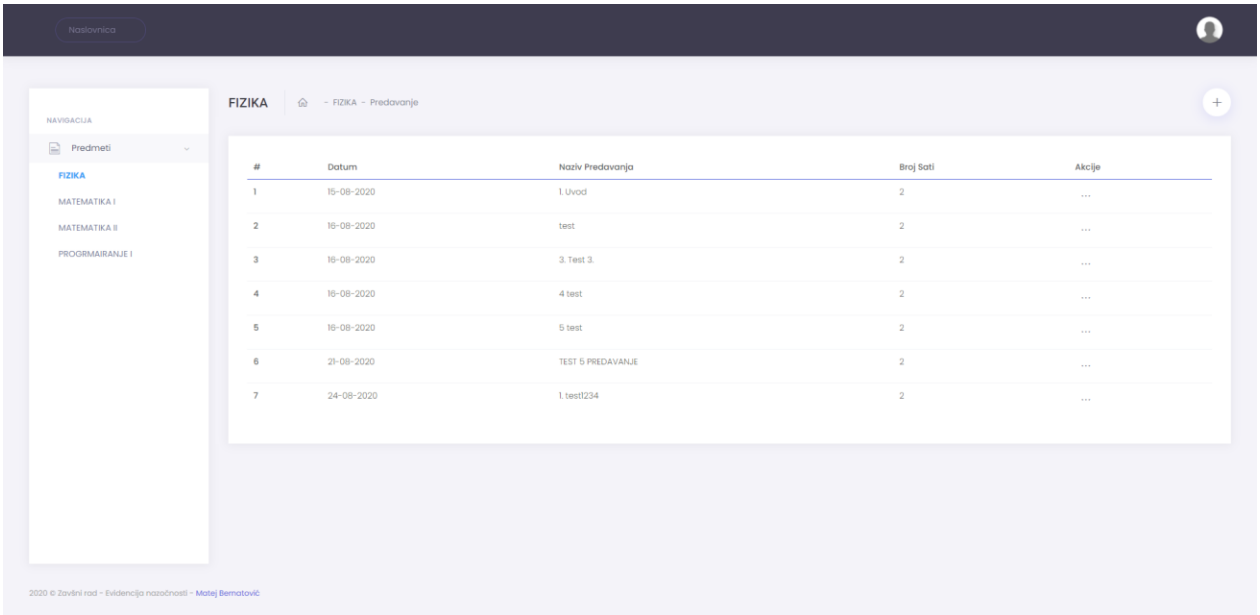
Klikom na naziv kolegija otvara se ekran (Slika 5.3.) te ovisno o sadržaju određenog kolegija ispisuje se navigacija. U ovom primjeru možemo vidjeti da ima predavanje, auditorne vježbe, laboratorijske vježbe te postotak održanosti istih. Omogućena je pretraga studenta pomoću JMBAG-a. U gore desnom kutu nalaze se dva gumba „Ukupni prikaz“ (Slika 5.7.) i „Ciljevi“ (Slika 5.4.).



Slika 5.4. Ciljevi kolegija

Klikom na gumb „Ciljevi“ otvara se prozor koji omogućava uređivanje minimalnog postotka nazočnosti studenta. Postoji dva tipa : ukupna minimalna nazočnost te tipovi kolegija koji postoje na određenom kolegiju.

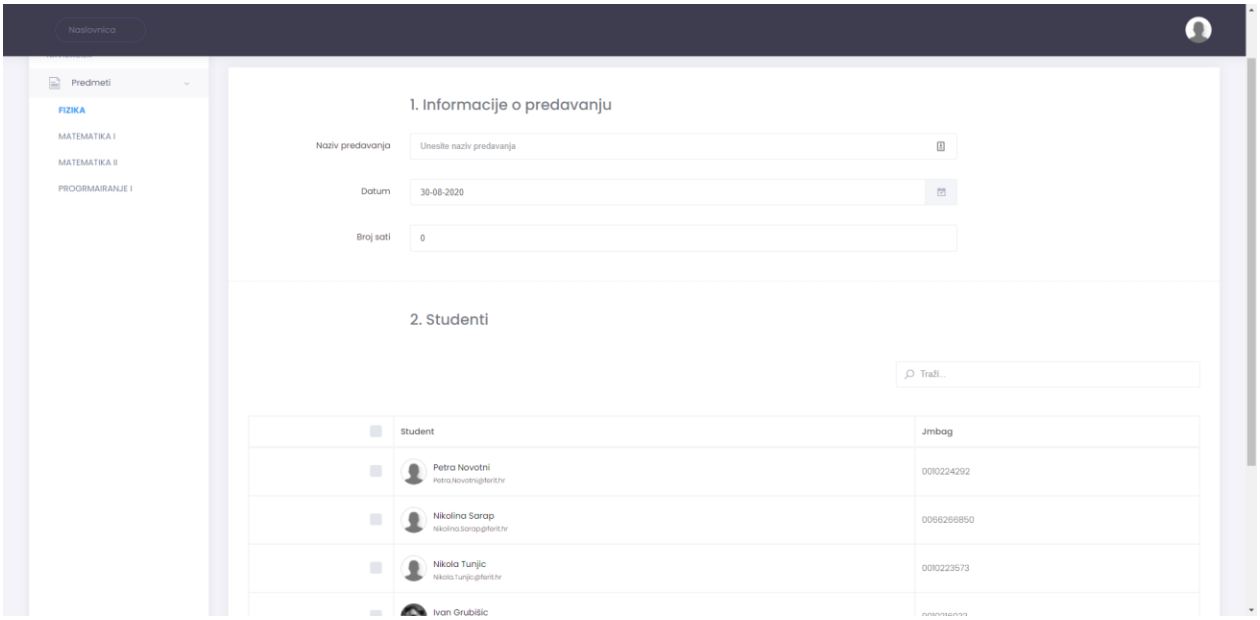
Klikom na „Predavanje“ (Slika 5.3.) prikazuje se tablica svih unesenih predavanja, te se isti takav prikaz koristi za ostale tipove (Auditorne vježbe, Laboratorijske vježbe).



Slika 5.5. Prikaz svih predavanja

Tablica sadrži redni broj predavanja, datum predavanja, naziv predavanja, broj odrađenih sati predavanja te akcije koje su označene gumbom sa tri točkice. Prilikom klika na gumb otvara se mali prozor gdje se može odabrati tri akcije : Detalji, Uredi, Obriši. Akcija „Detalji“ otvara novi prozor gdje se nalazi tablica sa popisom svih studenata te se može vidjeti tko je prisustvovao na predavanju a tko nije. Akcija „Uredi“ otvara novi prozor gdje se može urediti informacije o predavanju te nazočnosti studenta. Akcija „Obriši“ uklanja predavanje i sve studente koji su povezani s navedenim predavanjem.

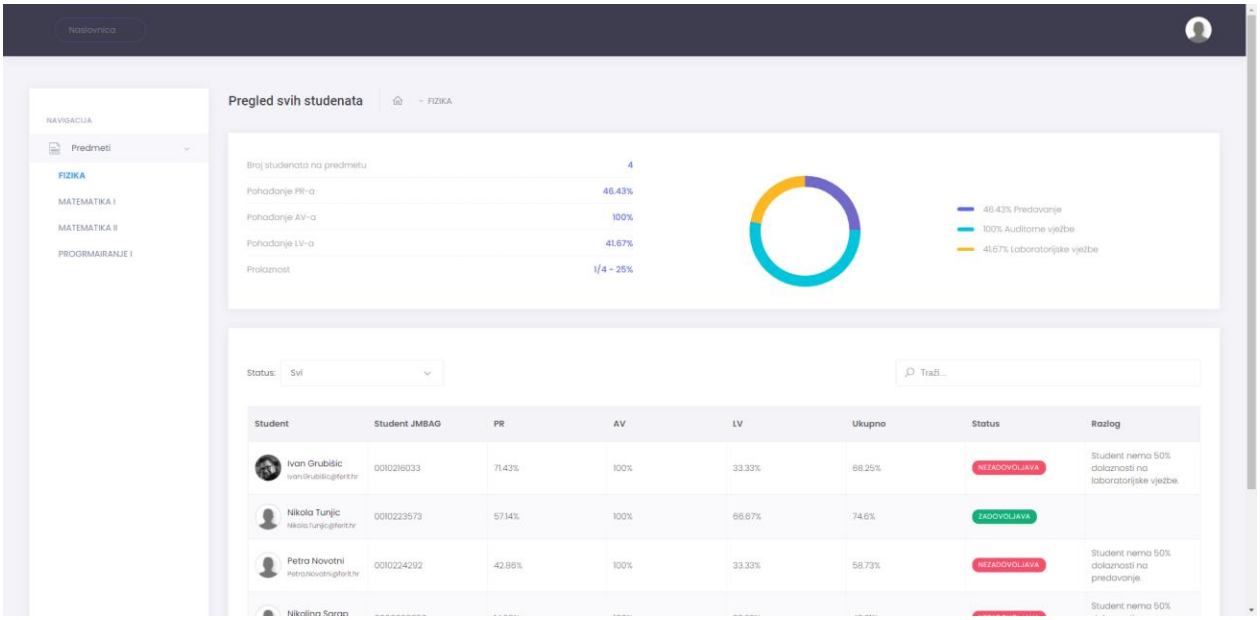
Klikom na gore gumb (+) u gornjem desnom kutu (Slika 5.5.) otvara se novi prozor (Slika 5.6.) gdje se dodaje novo predavanje.



Slika 5.6. Dodavanje novog predavanja

Postoje dvije sekcije : informacije o predavanju i studenti. Kako bi se dodalo novo predavanje mora se popuniti prva sekcija informacije o predavanju gdje pripadaju: naziv predavanja, datum predavanja, te broj sati koji će se održati. Druga sekcija studenti sadrži tablicu svih studenata koji su upisani na određeni kolegij te klikom na okvir koji stoji određenog studenta određujemo nazočnost studenta.

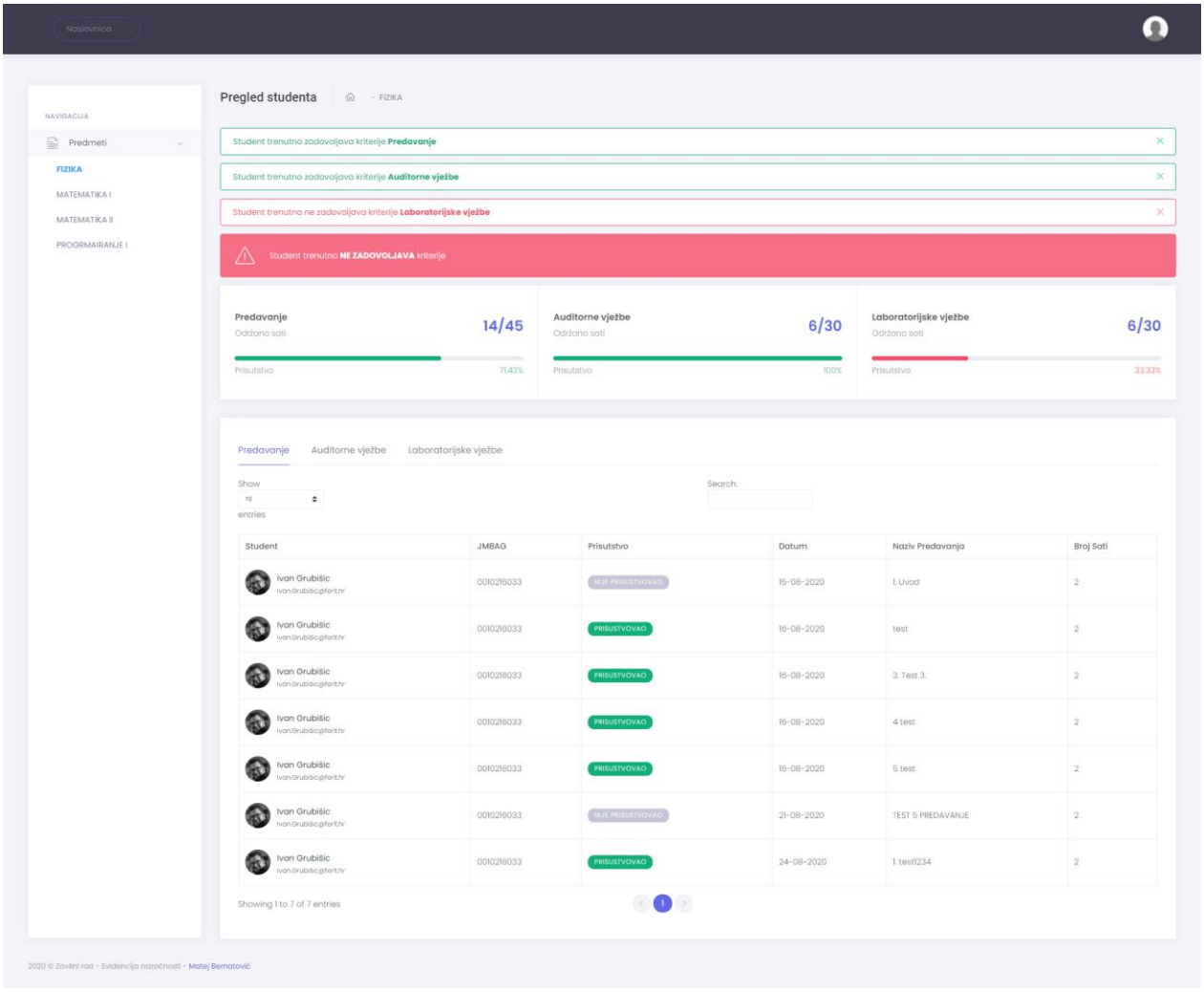
Klikom na gumb „Ukupni prikaz“ (Slika 5.3.) otvara se ekran pregled svih studenata upisanih na određeni kolegij (Slika 5.7.).



Slika 5.7. Pregled svih studenata određenog kolegija

Prikazan je broj studenata na predmetu, postotak pohađanja tipova kolegija te postotak prolaznosti. Isto tako prikazana je tablica u kojoj se nalaze svi studenti kolegija, njihovi podaci, JMBAG, pojedinačni postotak nazočnosti na tipove kolegija, ukupan postotak nazočnosti, status koji označava zadovoljavanje odnosno nezadovoljavanje, te razlog ukoliko student ne zadovoljava kriterij. Tablica u gore lijevom kutu ima padajući izbornik statusa gdje se klikom na određeni tablica filtrira, te u gore desnom kutu postoji tražilica koja pretražuje tablicu prema upisanoj riječi.

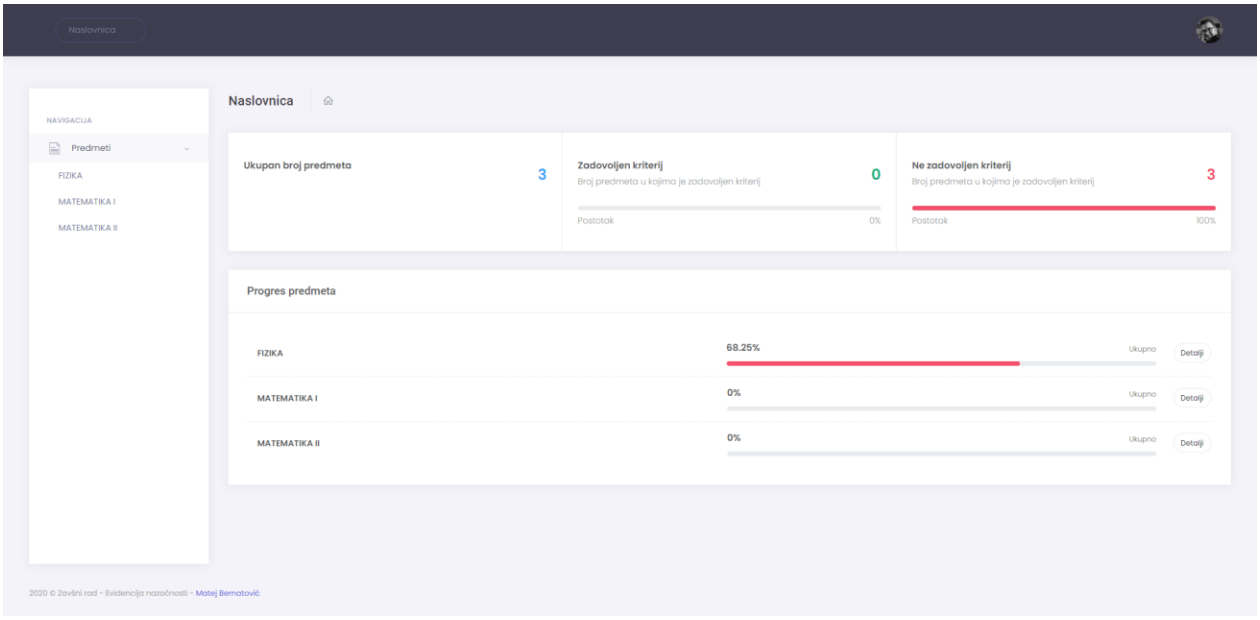
Ukoliko se upiše studentov JMBAG te stisne gumb „Pretraga“ (Slika 5.3.) otvara se ekran koji prikazuje pregled studenta s upisanim JMBAG-om (Slika 5.8).



Slika 5.8. Pregled studenta pomoću JMBAG-a

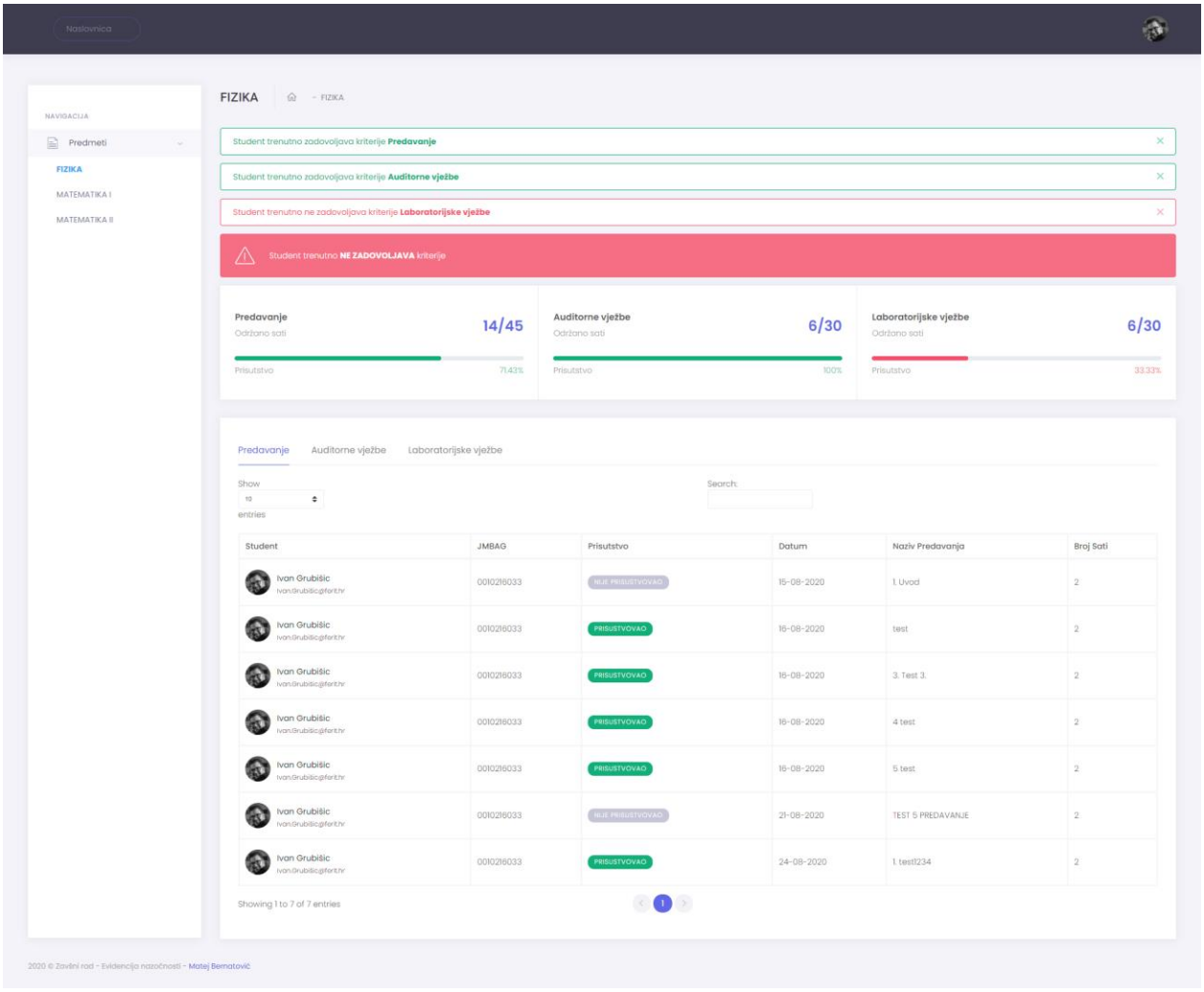
Prikazane su poruke koje označavaju kriterije pojedinog tipa u dva slučaja : zelena poruke (ukoliko student zadovoljava kriterij) i crvena poruka (ukoliko student ne zadovoljava kriterij). Prikazana je statistika tipova kolegija gdje je naveden broj održanih sati te prisustvo studenta na temelju istih. Također nalazi se tablica svih tipova s popisom svih predavanja gdje se može vidjeti je li student bio nazočan ili ne.

Nakon ispune podataka s početne stranice (Slika 5.1.) te ukoliko je tip prijave „student“ prilikom klika na gumb „prijava“ sustav autorizira korisnika te se prikazuje naslovnica studenta (Slika 5.9.).



Slika 5.9. Naslovnica studenta

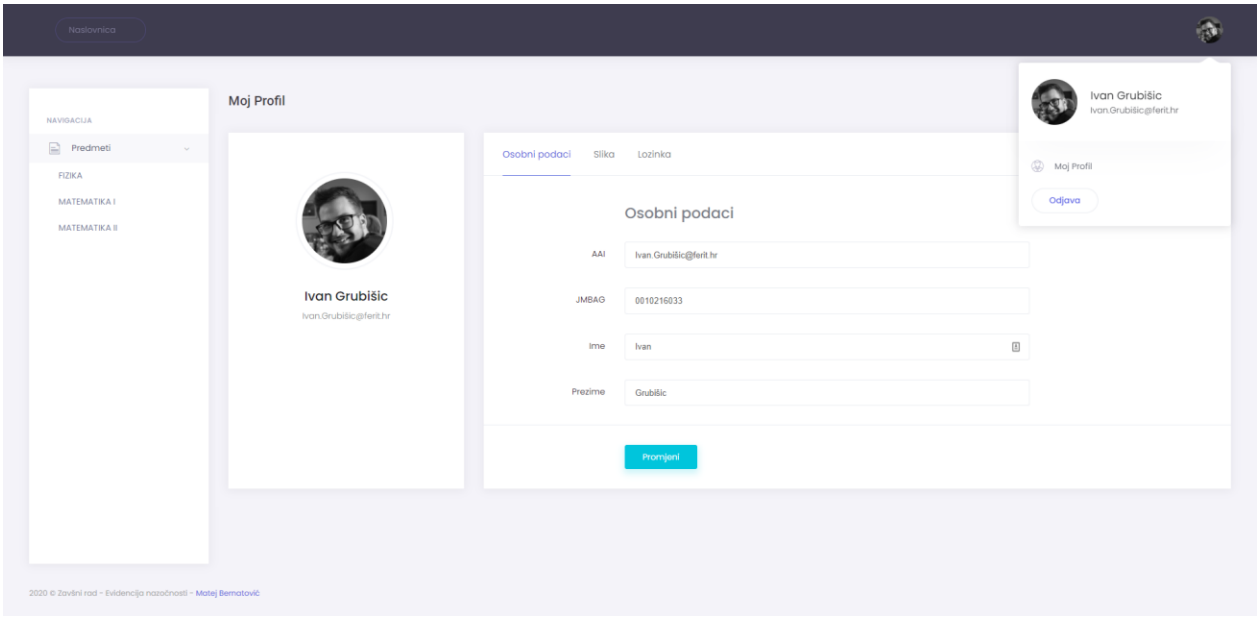
Naslovnica studenta prikazuje ukupan broj kolegija u koje je student upisan te broj i postotak zadovoljenog, odnosno ne zadovoljenog kriterija te tablicu koja prikazuje progres svih kolegija. Klikom na gumb „Detalji“ pokraj postotka nazočnosti otvara se novi ekran sa statistikom odabranog kolegija (Slika 5.10.).



Slika 5.10. Pregled studentova kolegija

Na slici su prikazane su poruke koje označavaju kriterije pojedinog tipa u dva slučaja : zelena poruke (ukoliko student zadovoljava kriterij) i crvena poruka (ukoliko student ne zadovoljava kriterij). Prikazana je statistika tipova kolegija gdje je naveden broj održanih sati te prisustvo studenta na temelju istih. Također nalazi se tablica svih tipova s popisom svih predavanja gdje se može vidjeti je li student bio nazočan ili ne.

U navigacijskoj traci, prilikom klika na sliku (gore desni kut) otvara se prozor sa detaljima prijavljenog korisnika (studenta ili profesora) te postoje dva gumba „Moj Profil“ (Slika 5.11.) i „Odjava“. Prilikom klika na gumb „Odjava“ korisnik izlazi iz aplikacije i vraća se na početni ekran.



Slika 5.11. Moj profil

Postoje tri kartice : Osobni podaci, Slika, Lozinka. Početna kartica su “Osobni podaci” gdje su prikazani korisnikovi podaci, a korisnik ima mogućnost uređivanja isključivo imena i prezimena. Prilikom odabira kartice “Slika” korisnik ima mogućnost postavljanja svoje fotografije profila , dok prilikom odabira na karticu “Lozinka” korisnik ima mogućnost promjene svoje lozinke. Ukoliko korisnik uspješno promijeni lozinku aplikacija ga vraća na početni ekran te prijavljuje sa svojom novom lozinkom.

6. ZAKLJUČAK

Cilj ovog rada je olakšavanje profesorima evidenciju nazočnosti studenata na fakultetima kao i lakši pristup studentima provjere pojedinačne osobne nazočnosti te je povodom toga izrađena web aplikacija evidencija nazočnosti. Aplikacija omogućava unos predavanja u sustav, dodavanje nazočnosti studenta navedenog predavanja, pregled održanih predavanja i nazočnost studenta te postotak istih. Prilikom izrade aplikacije koristio se alat za logiku Visual Studio framework .Net Core 3.1 te MSSQL baza za spremanje podataka. Komunikacija između Visual Studia i MSSQL baze provodi se sa Entity Frameworkom. Prikaz sučelja napravljen je pomoću HTML-a i CSS-a (korišteno je pomagalo Boostrap). Izrađena web aplikacija mogla bi svoje unaprjeđenje steći povezivanjem sustava sa ISVU te povlačenja podataka s istog. Prilikom kreiranja novog predavanja mogao bi se kreirati QR kod te prikazati na projektoru svim studentima. Studenti bi pomoću aplikacije skenirali QR kod te bi se na taj način prijavili na predavanje, a to bi omogućilo prijavu studenata automatizmom. Takav način bi profesorima skratilo vrijeme unosa podataka evidencije nazočnosti.

LITERATURA

[1] Yahl, M. (2012). C#, CSC 415 : *Programming Languages.* Dostupno 28.08.2020 na http://campus.murraystate.edu/academic/faculty/wlyle/415/2012/Yahl.docx

[2] Mayo J. (2002). C# *Biblioteka Ekspert*. Zagreb: Miš

[3] Millet, S. *Professional ASP.NET Design Patterns.* Indianapolis : Wiley Publishing, Inc., 2010.

[4] Agenda Mahala, *Introduction to CSS*. Dostupno 28.08.2020. na https://agendamahala.com/introduction-to-css

[5] W3Schools, *JavaScript Introduction.* Dostupno 30.08.2020 na https://www.w3schools.com/js/js_intro.asp

[6] Đambić. G., Zdešić, V. *Razvoj web aplikacija: priručnik.* Zagreb: Algebra, 2014.

[7] Atlantic, *What is MSSQL(Microsoft SQL)?.* Dostupno 30.08.2020. na https://www.atlantic.net/what-is-mssql/

[8] W3Schools, *SQL Syntax.* Dostupno 30.08.2020. na https://www.w3schools.com/sql/sql_syntax.asp

[9] Smartdraw, *Entity Relationship Diagram.* Dostupno 31.08.2020. na https://www.smartdraw.com/entity-relationship-diagram/

[10] Kaštelan, T. *Uvod u baze podataka: priručnik.* Zagreb: Algebra, 2010.

[11] CareerFoundy, *What is Boostrap: A Beginners Guide.* Dostupno 31.08.2020 na https://careerfoundry.com/en/blog/web-development/what-is-bootstrap-a-beginners-guide/

[12] jQuery. Dostupno 31.08.2020. na https://jquery.com/

[13] Pusher, *CLEAN ARCHITECTURE FOR THE REST OF US.* Dostupno 31.08.2020 na https://pusher.com/tutorials/clean-architecture-introduction

SAŽETAK

Cilj završnog rada bio je izraditi web aplikaciju za evidenciju nazočnosti u programskom jeziku C# koja će biti u svrhu uporabe profesorima i studentima. Bit je skratiti vrijeme unosa podataka ručnim putem te omogućiti lakši pregled i veću dostupnost podataka u bilo koje vrijeme. Aplikaciju je moguće koristiti i na mobilnim uređajima pomoću web preglednika jer responzibilna i za mobilne uređaje zahvaljujući Bootstrapu. U teorijskom dijelu rada opisane su korištene tehnologije poput programskog jezika C# unutar kojega su frameworka ASP.NET Core, tehnike MVC. Za bazu podataka korištena je Microsoftova MSSQL baza. Komunikacija između C#-a i baze se vodi pomoću Entity Framework platforma. Kod prikaza aplikacije tehnologije koje su korištene i opisane su : HTML , CSS, Javascript (Bootstrap).

Ključne riječi: C#, ASP.NET Core, MVC, MSSQL, Bootstrap.

ABSTRACT

The aim of the final work was to create a web application for attendance records in the C # programming language, which will be for use by professors and students. The essence is to shorten the time of data entry manually and to enable easier review and greater availability of data at any time. The application can also be used on mobile devices using a web browser because it is also responsive for mobile devices thanks to Bootstrap. The theoretical part of the paper describes the technologies used, such as the C # programming language, which includes the ASP.NET Core framework, MVC techniques. Microsoft's MSSQL database was used for the database. Communication between C # and the database is managed using the Entity Framework platform. When displaying application technologies used and described are: HTML, CSS, Javascript (Bootstrap).

Keywords: C #, ASP.NET Core, MVC, MSSQL, Bootstrap.

ŽIVOTOPIS

Matej Bernatović rođen 20.lipnja 1994. godine u Našicama. Pohađao je osnovnu školu Kralja Tomislava u Našicama. Nakon završenog osnovnoškolskog obrazovanja upisuje Matematičku gimnaziju u srednjoj školi Izidora Kršnjavog . Maturirao je u Našicama 2013.godine te je iste te godine upisao Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Hrvatski jezik mu je materini jezik, dok se također aktivno služi engleskim jezikom. Matej poznaje i ima iskustva u korištenju proizvoda Microsoft Office-a, programskog alata Visual Studio te programskih jezika C# tehnologije .Net Core, MSSQL baze, HTML-a, CSS-a, JavaScripte.

Vlastoručni potpis:

_____

Matej Bernatović

PRILOZI

CD:

- Elektronička inačica rada (.docx i .pdf format)
- Izrađeni projekt u Visual Studiu