

# Aplikacija za praćenje prihoda i troškova

---

**Bačić, Patrik**

**Undergraduate thesis / Završni rad**

**2024**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Applied Sciences in Information Technology / Veleučilište suvremenih informacijskih tehnologija**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:289:643670>

*Rights / Prava:* [In copyright](#)/[Zaštićeno autorskim pravom.](#)

*Download date / Datum preuzimanja:* **2025-02-22**

*Repository / Repozitorij:*

[VSITE Repository - Repozitorij završnih i diplomskih radova VSITE-a](#)



**VELEUČILIŠTE SUVREMENIH INFORMACIJSKIH TEHNOLOGIJA**  
**STRUČNI PRIJEDIPLOMSKI STUDIJ INFORMACIJSKIH**  
**TEHNOLOGIJA**

**Patrik Bačić**

**ZAVRŠNI RAD**

**APLIKACIJA ZA PRAĆENJE PRIHODA I TROŠKOVA**

**Zagreb, listopada 2024.**

Studij: Stručni prijediplomski studij informacijskih tehnologija  
smjer programiranje  
Student: **Patrik Bačić**  
Matični broj: 2019030

## Zadatak završnog rada

Predmet: Programiranje na Internetu  
Naslov: **Aplikacija za praćenje prihoda i troškova**  
Zadatak: Razviti i pripremiti za produkcijski rad aplikaciju za praćenje troškova. Koristiti C#, MVC, ASP.NET, JS, CSS i MS SQL tehnologije za razvoj konačnog rješenja. Opisati postupak izrade. Opisati .NET 6 tehnologiju.  
Mentor: Mariza Maini, pred.  
Zadatak uručen kandidatu: 11.10.2023.  
Rok za predaju rada: 24.10.2024.  
Rad predan: \_\_\_\_\_

### Povjerenstvo:

Dragana Čulina, pred.	član predsjednik	_____
Mariza Maini, pred.	mentor	_____
Jurica Đurić, v. pred.	član	_____

# SADRŽAJ

1. UVOD .....	7
2. KORIŠTENE TEHNOLOGIJE .....	9
2.1. ASP.NET Core .....	9
2.2. .NET 6 .....	9
2.3. HTML.....	9
2.4. CSS.....	9
2.5. JavaScript .....	10
2.6. C# .....	10
2.7. MVC.....	10
2.7.1. Model .....	10
2.7.2. Pogled .....	11
2.7.3. Kontroler .....	11
2.8. Baza podataka (SQL Server).....	11
2.9. Visual Studio NuGet paketi.....	11
3. PREGLED FUNKCIONALNOSTI APLIKACIJE .....	13
3.1. Registracija i prijava.....	13
3.2. Kreiranje kategorija.....	15
3.3. Kreiranje transakcija.....	15
3.4. Praćenje transakcija unutar statistika .....	15
4. PRAKTIČNI RAD – APLIKACIJA ZA PRAĆENJE PRIHODA I TROŠKOVA.....	17
4.1. Postavljanje aplikacije i instalacija NuGet paketa .....	17
4.2. Definiranje i konfiguracija modela.....	17
4.2.1. Model kategorija .....	17
4.2.2. Model transakcija.....	18
4.3. Postavljanje baze podataka i tablica.....	20
4.3.1. Definiranje konteksta aplikacijske baze podataka .....	20
4.3.2. Dodavanje mehanizma za ubrizgavanje ovisnosti .....	21
4.3.3. Konfiguracija Connection String-a .....	22
4.3.4. Kreiranje migracije i ažuriranje baze podataka .....	22
4.4. Implementacija kontrolera.....	23
4.4.1. Kontroler za kategorije .....	23
4.4.2. Kontroler za transakcije .....	26

4.4.3. Kontroler za statistike .....	28
4.5. Razvoj korisničkog sučelja.....	30
4.5.1. Pogledi za kategorije.....	30
4.5.2. Pogledi za transakcije .....	32
4.5.3. Pogled za statistike.....	33
5. ZAKLJUČAK .....	36
LITERATURA.....	38
SAŽETAK .....	39
SUMMARY .....	40

## POPIS SLIKA

Slika 1. Model-View-Controller princip rada (Wikipedia, 2024) .....	10
Slika 2. Korišteni NuGet paketi .....	12
Slika 3. Registracija .....	13
Slika 4. Prijava .....	14
Slika 5. Postavke računa .....	14
Slika 6. Baza podataka i tablice .....	23
Slika 7. Izgled liste kategorija.....	31
Slika 8. Izgled kreiranja nove kategorije .....	31
Slika 9. Izgled liste transakcija .....	32
Slika 10. Izgled kreiranja nove transakcije .....	33
Slika 11. Izgled filtera datuma i prikaza prihoda, troškova i stanja na računu .....	33
Slika 12. Izgled grafikona za mjesečne troškove.....	34
Slika 13. Izgled grafikona mjesečnih prihoda naspram troškova .....	34
Slika 14. Izgled grafikona za stanje računa kroz godinu .....	35

## POPIS KODOVA

Kôd 1. Svojstva i pravila za kategorije .....	18
Kôd 2. Svojstva i pravila za transakcije.....	20
Kôd 3. ApplicationDbContext klasa .....	21
Kôd 4. Ubrizgavanje ovisnosti.....	21
Kôd 5. Spajanje aplikacije s bazom podataka.....	22
Kôd 6. Konstruktor .....	24
Kôd 7. Index metoda.....	24
Kôd 8. POST metoda za kreiranje ili ažuriranje kategorije .....	25
Kôd 9. POST metoda za brisanje postojećih kategorija .....	26
Kôd 10. POST metoda za kreiranje ili ažuriranje transakcije.....	28
Kôd 11. Metoda za popunjavanje kategorija .....	28
Kôd 12. Izračun prihoda i troškova za odabrani mjesec .....	29
Kôd 13. Izračun sveukupnog stanja na računu .....	29
Kôd 14. Grafikon za praćenje svih troškova unutar odabranog mjeseca.....	30

## 1. UVOD

U današnjem svijetu praćenje vlastitih financija postaje sve važnije zbog stalnih i novih izazova vezanih uz upravljanje prihodima i troškovima. Kontrola nad financijskim transakcijama omogućava lakše planiranje budućih financijskih aktivnosti i bolji uvid u trenutno stanje. Mnogi ljudi ne koriste nikakve alate za praćenje transakcija, što može dovesti do nelagodnih financijskih situacija.

Kako bi se ovaj problem riješio, razvijena je aplikacija koja se temelji na tri glavne komponente: kategorije, transakcije i statistike. Kategorije su ključne za precizno praćenje i analizu prihoda i troškova. One omogućuju klasifikaciju financijskih aktivnosti, pri čemu svaka osoba može definirati svoje vlastite kategorije poput plaće, režija, namirnica i goriva. Transakcije omogućuju bilježenje financijskih aktivnosti unutar tih kategorija, uključujući iznos, datum izvršenja i opis transakcije. Povezanost između kategorija i transakcija omogućuje korisnicima detaljno praćenje njihovih financijskih aktivnosti.

Komponenta statistike pruža vizualni uvid u financijske aktivnosti putem grafikona. Prikazuju se troškovi za određeni mjesec i godinu, usporedba prihoda i troškova te stanje na računu kroz cijelu godinu. Osim grafikona, statistike također prikazuju ukupne iznose prihoda i troškova te stanje na računu za odabrani period. Ovi grafički prikazi pomažu korisnicima da bolje razumiju svoje financije i donesu informirane odluke.

Za razvoj ove aplikacije korišteni su programski jezik C# i ASP.NET Core MVC (.NET 6) koji nude visoke performanse, skalabilnost i jednostavno održavanje. Za pohranu podataka korišten je Microsoft SQL Server. NuGet paketi unutar Visual Studija omogućuju dodavanje raznih funkcionalnosti i ubrzavaju razvoj aplikacije. Implementirana je i funkcionalnost registracije i prijave korisnika radi osiguravanja personaliziranog iskustva i sigurnosti podataka.

Cilj ove aplikacije je olakšati korisnicima praćenje prihoda i troškova, omogućujući im bolji uvid u financijske aktivnosti i identifikaciju područja gdje mogu uštedjeti. Pružanjem vizualnog uvida putem grafikona i pomoću detaljnog praćenja kategorija i transakcija, korisnici mogu donositi informirane odluke o svojim financijama i postići bolje financijsko stanje. Registracija i prijava korisnika osiguravaju da svaki korisnik može pratiti isključivo svoje podatke, što dodatno povećava sigurnost i privatnost.

U narednim poglavljima detaljnije će se opisati korištene tehnologije, glavne komponente aplikacije, funkcionalnosti registracije i prijave te logika koda koja osigurava ispravno funkcioniranje aplikacije. Kroz detaljnu analizu i objašnjenja, cilj je pružiti sveobuhvatan uvid u razvoj i funkcioniranje ove aplikacije te demonstrirati njenu korisnost u upravljanju osobnim financijama.





## 2. KORIŠTENE TEHNOLOGIJE

Unutar ovog poglavlja opisane su tehnologije korištene za realizaciju aplikacije, a to su ASP.NET Core, .NET 6, C#, HTML, CSS, JavaScript, MVC, baza podataka (SQL Server), NuGet paketi.

### 2.1. ASP.NET Core

ASP.NET Core je platformno neovisan okvir otvorenog koda idealan za brzu izradu aplikacija visokih performansi. U ovom projektu koristi se zbog toga što omogućava jednostavniji razvoj mrežnih aplikacija. Omogućuje integraciju modernih tehnologija za mrežni razvoj, podržava Blazor za izradu aplikacija s C# kodom koji može raditi na serveru i/ili klijentu, te nudi jednostavno testiranje i upotrebu ubrizgavanja ovisnosti (engl. *Dependency injection*). Integrira se s Kestrelom, IIS-om, Nginxom, Dockerom te ima ugrađenu podršku za učinkovit HTTP pipeline, što ga čini izuzetno moćnim alatom za moderni mrežni razvoj.

### 2.2. .NET 6

.NET 6 je verzija Microsoftovog razvojnog okvira koja unificira razvoj različitih vrsta aplikacija, od mobilnih i stolnih do cloud i IoT (engl. *Internet of Things*) aplikacija, koristeći isti SDK, osnovne biblioteke i runtime. Pruža poboljšane jezične značajke kroz C# 10, koje smanjuju količinu potrebnog koda te uvodi minimalne API-je za brže i jednostavnije pisanje mikroservisa. Ovaj okvir poznat je po visokim performansama i učinkovitosti, što pomaže smanjiti troškove resursa u cloud okruženju. Razvojni alati, kao što je Visual Studio 2022, dodatno poboljšavaju produktivnost pružajući mogućnosti poput *hot reloada* koji omogućuje programerima da odmah vide promijene u kodu tijekom izvršavanja aplikacije, poboljšanog uređivanja koda, dijagnostike i testiranja, što sve doprinosi bržem i lakšem razvoju aplikacija.

### 2.3. HTML

HTML (engl. *HyperText Markup Language*) je osnovni jezik za izradu mrežnih stranica koji se koristi za strukturiranje sadržaja. U kontekstu ove aplikacije, HTML se koristi za izgradnju strukture korisničkog sučelja, odnosno pogleda. Konkretno, koristi se za izradu formi za unos podataka, prikaz tabela s podacima o transakcijama, prikaz grafikona i izvještaja, te za navigacijske elemente kao što su izbornici i linkovi.

### 2.4. CSS

CSS (engl. *Cascading Style Sheets*) se koristi za stiliziranje HTML elemenata. Omogućava definiciju izgleda i rasporeda mrežne stranice. Unutar ove aplikacije, CSS se koristi za definiranje vizualnog izgleda sučelja, uključujući boje, fontove, margine i raspored elemenata.

Konkretno, CSS se koristi za stiliziranje formi za unos podataka, tabela, grafikona, navigacijske trake, te za osiguranje responzivnosti mrežne stranice.

## 2.5. JavaScript

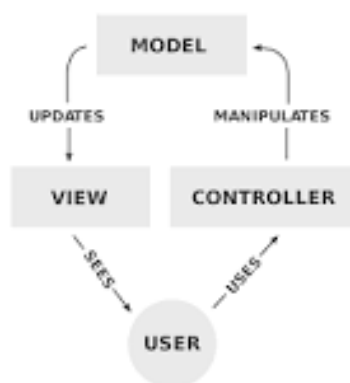
JavaScript je integralni dio mrežnih tehnologija i koristi se za stvaranje dinamičnih i interaktivnih korisničkih sučelja. U kontekstu ove aplikacije, JavaScript se koristi posredno putem Syncfusion komponenti za ASP.NET Core. Syncfusion biblioteka koristi JavaScript u pozadini za renderiranje i funkcioniranje svojih komponenti, uključujući grafikone, tablice i druge interaktivne elemente, bez potrebe da programer direktno piše JavaScript kod.

## 2.6. C#

C# (engl. *C-Sharp*) je objektno orijentirani programski jezik kojeg je razvio Microsoft. Koristi se za razvoj raznih aplikacija koje rade na .NET platformi. U kontekstu ove aplikacije, C# je korišten za implementaciju poslovne logike, modela, kontrolera i komunikaciju s bazom podataka. Njegova sintaksa je jednostavna i čitljiva, što olakšava razvoj i održavanje koda.

## 2.7. MVC

Skraćenica MVC predstavlja model, pogled i kontroler (engl. *Model-View-Controller*) obrazac koji razdvaja aplikaciju na te tri glavne komponente, čime se omogućuje bolje upravljanje složenošću koda i poboljšava se održavanje i testiranje aplikacija. Kada ne bi postojala jasna separacija logike, podataka i prezentacije, bilo bi puno teže raditi promjene u kôdu ili dodavati nove funkcionalnosti. Ova arhitektura funkcionira na način da se korisnički zahtjevi usmjeravaju na kontroler koji uz pomoć modela dohvaća ili ažurira podatke te se nakon toga ti podaci prikazuju korisniku preko pogleda (Slika 1).



Slika 1. Model-View-Controller princip rada (Wikipedia, 2024)

### 2.7.1. Model

Na model se može gledati kao na tablicu unutar baze. Unutar modela se nalazi i poslovna logika aplikacije te uz pomoć Entity Framework Corea, alata za objektno relacijsko mapiranje za .NET

koji olakšava rad s bazama podataka, model može komunicirati s bazom podataka. U kontekstu ove aplikacije, na primjer za model zvan transakcije, svojstva koja su definirana unutar tog modela će se nalaziti unutar baze podataka u obliku kolona i sadržavat će vrijednosti koje korisnik unese tijekom kreiranja jedne transakcije.

### **2.7.2. Pogled**

Pogled (engl. *View*) je zadužen za prikaz podataka i sadržaja kroz korisničko sučelje. Unutar pogleda se ne bi trebala nalaziti nikakva poslovna logika, a ako je već i ima, onda bi se ona trebala odnositi samo na prezentiranje sadržaja. Aplikacija generira HTML koristeći mehanizam za prikaz (engl. *View engine*), kao što je Razor, koji dinamički stvara HTML na osnovu podataka i predložaka.

### **2.7.3. Kontroler**

Kontroler je komponenta koja reagira na korisničke zahtjeve. On surađuje s modelom, preusmjerava upite prema modelima i po potrebi vraća poglede unutar kojih će biti prikazani obrađeni podaci. Unutar MVC-a, kontroler se smatra početnom točkom i na njemu leži odgovornost odabira tipova modela koji će se procesuirati te koji pogledi će se renderirati. U kontekstu ove aplikacije, kontroleri imaju najveću uporabu pri kreiranju kategorija i transakcija, te kod brisanja ili ažuriranja istih.

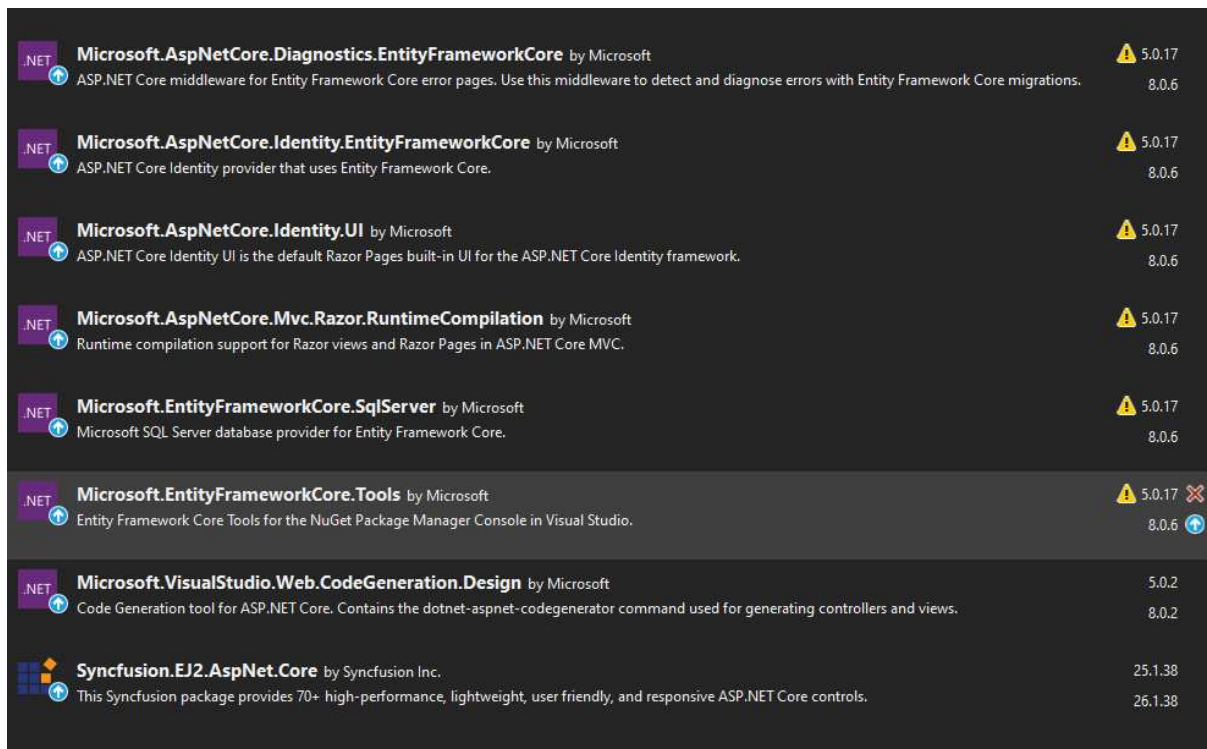
## **2.8. Baza podataka (SQL Server)**

Baza podataka je organizirani skup povezanih podataka koji zadovoljava informacijske potrebe korisnika u organizaciji, pohranjen u vanjskoj memoriji računala bez neželjene redundancije (Bužić, Dalibor, 2016). SQL Server je relacijski sustav za upravljanje bazama podataka koji je razvijen od strane Microsofta. Koristi se za pohranu i upravljanje velikom količinom podataka na siguran i efikasan način. Primjena baze podataka, točnije SQL Servera, u ovoj aplikaciji jest ta da prilikom izvršavanja raznih operacija unutar kôda, podaci o transakcijama i kategorijama se ovisno o operacijama ažuriraju, kreiraju, brišu ili dohvaćaju. Svaki prijavljeni korisnik će striktno vidjeti vlastite podatke vezane uz transakcije i kategorije unutar aplikacije. Kako bi baza podataka bila povezana s aplikacijom, unutar kôda se koristi *connection string* u kojem je definirana adresa servera na kojem se nalazi baza podataka, naziv baze na koju se treba povezati te neki način autorizacije za pristup bazi.

## **2.9. Visual Studio NuGet paketi**

NuGet je upravljač paketa za .NET ekosustav koji omogućuje jednostavno preuzimanje, instalaciju i upravljanje vanjskim bibliotekama i alatima unutar projekta. NuGet paketi se koriste za brži i jednostavniji razvoj projekta te za integraciju već gotovih rješenja koja rješavaju

specifične probleme vezane uz odgovarajući projekt. U nastavku se nalaze NuGet paketi koji su korišteni pri izradi ove aplikacije (Slika 2). Što se tiče verzija ovih paketa, one se postavljaju tako da se prvo unutar Visual Studija provjeri pod *Dependencies* -> *Frameworks* -> *Microsoft.AspNetCore.App* -> *Properties* koja je verzija aplikacije te se za projekt odabere ta ista verzija koja je definirana unutar *Properties*.



Slika 2. Korišteni NuGet paketi

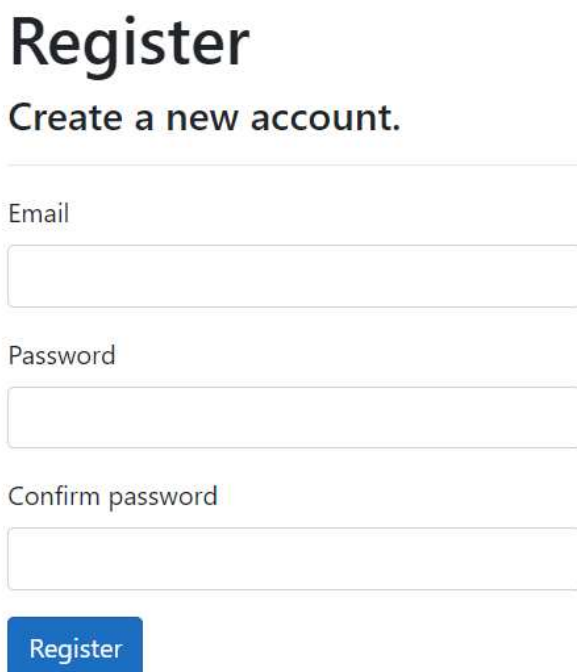
- Syncfusion.EJ2.AspNetCore - Koristi se za prikazivanje grafova i poboljšanje osnovnih komponenti aplikacije poput gumba, tekstualnih okvira, oznaka i padajućih izbornika.
- Microsoft.EntityFrameworkCore.Tools - Omogućava upravljanje migracijama baze podataka i ažuriranje baze putem Package Manager Console naredbi.
- Microsoft.EntityFrameworkCore.SqlServer - Integrira Entity Framework Core sa SQL Serverom za rad s podacima korisnika, kategorija i transakcija.
- Microsoft.AspNetCore.Identity.UI - Pruža funkcionalnosti za registraciju, prijavu i upravljanje korisničkim računima.
- Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation - Omogućava izvršavanje promjena u aplikaciji bez potrebe za ponovnim pokretanjem.
- Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore - Služi za praćenje i ispravljanje grešaka vezanih uz Entity Framework Core i bazu podataka.

### 3. PREGLED FUNKCIONALNOSTI APLIKACIJE

U ovom dijelu se detaljnije opisuju osnovne funkcionalnosti aplikacije, ispravan način i postupak registracije i prijave, postupak kreiranja kategorija i transakcija te korištenje i pregled statistika.

#### 3.1. Registracija i prijava

Kako bi se glavne komponente aplikacije mogle koristiti korisnik je dužan registrirati se i prijaviti u aplikaciju da bi dobio pristup vlastitim kategorijama i transakcijama te da bi ih mogao ažurirati ili kreirati nove. Tijekom registracije korisnik treba unijeti email adresu i lozinku koja se sastoji od barem jednog velikog tiskanog slova, barem jednog znaka i barem jednog broja te veličina lozinke mora biti preko 8 znamenki (Slika 3).



The image shows a registration form with the following elements:

- Register**: Large heading text.
- Create a new account.**: Sub-heading text.
- Email**: Label above a text input field.
- Password**: Label above a text input field.
- Confirm password**: Label above a text input field.
- Register**: A blue button with white text at the bottom.

Slika 3. Registracija

Nakon što se korisnik registrirao, zadnji korak kojeg mora odraditi, prije korištenja aplikacije, je prijava. Na tom koraku korisnik mora samo unijeti istu email adresu i lozinku koju je postavio tijekom registracije i stisnuti na gumb za prijavu (Slika 4).

# Log in

Use a local account to log in.

Email

nekimail@gmail.com

Password

.....

Remember me?

Log in

[Forgot your password?](#)

[Register as a new user](#)

[Resend email confirmation](#)

Slika 4. Prijava

Ako korisnik želi mijenjati bilo kakve podatke vezane uz svoj račun, to može napraviti klikom na svoj račun koji se nalazi na navigacijskoj traci (Slika 5).

## Profile

### Manage your account

#### Change your account settings

<b>Profile</b>	<b>Profile</b>
Email	Username
Password	nekimail123@gmail.com
Two-factor authentication	Phone number
Personal data	
	Save

Slika 5. Postavke računa

### **3.2. Kreiranje kategorija**

Prvi korak koji je nužan za odraditi je kreiranje kategorija. Kategorije će uglavnom predstavljati stavke na koje neki korisnik najčešće troši svoj novac, no to ne moraju nužno biti samo troškovi, to može biti i nekakav prihod poput mjesečne plaće ili nekog drugog oblika zarade. Prilikom pozicioniranja u kategorije, korisniku se prikaže lista kategorija koja će u početku biti prazna. Da bi se kategorija kreirala potrebno je stisnuti gumb za stvaranje kategorije (engl. *New Category*), nakon toga korisnik mora unijeti naziv kategorije, označiti je li ta kategorija trošak ili prihod te pri završetku stisnuti na gumb za spremanje, da bi se ta kategorija stvorila. Nakon što se neka kategorija unese, unutar liste kategorija se prikazu naziv i tip kategorije koji se mogu filtrirati. Za svaku kategoriju na raspolaganju su dva gumba koja služe za izmjenu i brisanje kategorije.

### **3.3. Kreiranje transakcija**

Nakon kreiranja kategorije, sljedeći korak je kreirati transakciju za tu kategoriju. Prilikom pozicioniranja na komponentu transakcije, korisniku se prikaže prazna lista transakcija koja će se puniti unosom transakcija. Unutar te liste postoje stupci za naziv kategorije, iznos i datum izvršenja transakcije. Ti stupci kao i stupci unutar liste kategorija se također mogu filtrirati, te se kao i kod kategorija svaka transakcija može izmijeniti i obrisati. Da bi se kreirala nova transakcija, treba se pritisnuti gumb za stvaranje nove transakcije (engl. *New Transaction*) iznad liste transakcija. Nakon toga se mora iz padajuće liste odabrati jednu od kreiranih kategorija, unijeti iznos koji je veći od nula, po želji staviti neki detaljan opis vezan uz tu transakciju i na kraju odabrati datum izvršenja transakcije. Jedino polje koje nije nužno popuniti je polje za opis transakcije. Na kraju kada su se sva potrebna polja popunila transakcija se kreira pritiskom na gumb za stvaranje te se novu transakciju može odmah vidjeti unutar liste transakcija.

### **3.4. Praćenje transakcija unutar statistika**

Posljednja, no i ne manje bitna komponenta statistike, služi za detaljno prikazivanje i praćenje postojećih transakcija. Prva stvar koju korisnik može uočiti prilikom pozicioniranja na komponentu statistike je iznos prihoda i troškova za trenutni mjesec te sveukupno stanje na računu. Ispod ove tri stavke nalazi se filter za datum i mjesec te gumb za ažuriranje svih statistika na željeni datum. Statistike se još sastoje od 3 ključna grafikona za bolje praćenje vlastitih transakcija. Prvi grafikon prikazuje sve troškove koji su izvršeni u odabranom mjesecu i godini (Slika 12). Na dnu grafikona su prikazani nazivi svih troškova koji se nalaze u grafikonu te ovisno o iznosu troška, što je veći iznos to će i trošak biti prikazan kao veći i obrnuto u slučaju da je iznos troška nizak. Drugi grafikon uspoređuje prihode naspram troškova



za odabrani mjesec i godinu (Slika 13). Ovaj grafikon služi za bolje dočaravanje korisniku troši li više nego li zarađuje, te treba li smanjiti potrošnju ili ne. Unutar grafikona, prihod je prikazan zelenom bojom, a trošak crvenom. Posljednji grafikon se odnosi na prikaz stanja na računu kroz odabranu godinu (Slika 14). Svrha ovog grafikona je da korisnik može vidjeti raste li njegovo stanje na računu ili pada kroz čitavu godinu. Također može uočiti postoje li periodi unutar godine gdje korisnik troši više ili manje.

## 4. PRAKTIČNI RAD – APLIKACIJA ZA PRAĆENJE PRIHODA I TROŠKOVA

Unutar ovog poglavlja se detaljnije prolazi kroz postavljanje razvojnog okruženja, razvoj kategorija, transakcija i statistika, izradu baze podataka te upotrebu autorizacije za tri glavne komponente.

### 4.1. Postavljanje aplikacije i instalacija NuGet paketa

Za početak razvoja aplikacije odabran je predložak ASP.NET Core Web App (Model-View-Controller) unutar razvojnog okruženja Visual Studio. Ovaj predložak omogućuje brzu izradu aplikacija koristeći MVC obrazac, što olakšava razvoj i održavanje kôda.

Prilikom konfiguracije predloška, odabrana je opcija za autentifikaciju Individualni računi (engl. *Individual Accounts*). Ova opcija omogućuje jednostavnu implementaciju registracije i prijave korisnika te pruža osnovne funkcionalnosti za upravljanje korisničkim računima, uključujući sigurnosne aspekte poput autentifikacije i autorizacije.

Također je označena opcija konfiguriranja za HTTPS (engl. *Configure for HTTPS*) kako bi se osigurala komunikacija putem HTTPS protokola, čime se povećava sigurnost aplikacije. Korištenje HTTPS-a pomaže u zaštiti osjetljivih podataka korisnika prilikom prijenosa između klijenta i servera.

Nakon inicijalne konfiguracije projekta, instalirani su potrebni NuGet paketi koji su navedeni i prikazani u prethodnom poglavlju, kako bi se osigurala funkcionalnost aplikacije.

### 4.2. Definiranje i konfiguracija modela

Sljedeći korak u razvoju aplikacije je definiranje i konfiguracija modela. Modeli predstavljaju strukturu podataka koja se koristi u aplikaciji i definiraju kako će se ti podaci pohranjivati u bazi podataka. U ovom slučaju, ključni modeli su Kategorija i Transakcija. Model Kategorija koristi se za pohranjivanje informacija o različitim kategorijama prihoda i troškova, dok model Transakcija prati pojedinačne financijske transakcije unutar tih kategorija.

Modeli se definiraju kao klase unutar aplikacije, a njihova konfiguracija uključuje postavljanje potrebnih svojstava i ograničenja za svaki model. Također je potrebno definirati relacije između modela kako bi se osiguralo ispravno funkcioniranje aplikacije i konzistentnost podataka.

#### 4.2.1. Model kategorija

Unutar modela za kategorije nalaze sva bitna svojstva i pravila koja su nužna za stvaranje kategorija (Kôd *I*).

- ID kategorije (CategoryId) - predstavlja jedinstveni identifikator za svaki zapis unutar tablice. Za ovo svojstvo postavljen je atribut koji ga definira kao primarni ključ što će svaku kategoriju identificirati i razlikovati jednu od druge unutar baze podataka.
- Naslov (Title) - predstavlja naziv kategorije. Atributi potrebno (engl. *Required*) i maksimalna dužina (engl. *MaxLength*) označavaju da naslov kategorije mora biti definiran, odnosno ne smije biti izostavljen, te isto tako ne smije sadržati više od 80 znakova.
- Tip (Type) - služi kako bi se odredilo hoće li kategorija predstavljati trošak ili prihod. Zadana vrijednost je trošak jer će za većinu korisnika biti više kategorija koje će predstavljati trošak nego prihod.
- ID korisnika (UserId) - predstavlja jedinstveni identifikator korisnika kojem kategorija pripada.
- Korisnik (IdentityUser User) - predstavlja navigacijsko svojstvo koje omogućava dohvat povezanog korisnika.

```
public class Category
{
    [Key]
    public int CategoryId { get; set; }
    [MaxLength(80)]
    [Required(ErrorMessage = "Title is required!")]
    public string Title { get; set; }
    [MaxLength(10)]
    public string Type { get; set; } = "Expense"; //default type

    public string UserId { get; set; }
    public IdentityUser User { get; set; }
}
```

Kôd 1. Svojstva i pravila za kategorije

#### 4.2.2. Model transakcija

Model za transakcije definira strukturu podataka za pojedinačne financijske transakcije unutar aplikacije. Unutar klase za transakcije nalaze se sva svojstva koje jedna transakcija mora posjedovati (Kôd 2).

- ID transakcije (TransactionId) - označava jedinstveni identifikator za svaku transakciju. Za ovo svojstvo definiran je atribut koji ga deklarira kao primarni ključ kako bi se svaka transakcija unutar baze podataka razlikovala jedna od druge.

- ID kategorije (CategoryId) - igra ulogu stranog ključa koji povezuje transakciju s određenom kategorijom. Ovo svojstvo označava kojoj kategoriji transakcija pripada, čime se osigurava veza između transakcija i njihovih odgovarajućih kategorija.
- Kategorija (Category) - služi kao navigacijsko svojstvo koje omogućuje pristup podacima kategorije povezane s transakcijom. Ovo svojstvo koristi se za definiranje relacije između transakcije i kategorije.
- Iznos (Amount) - predstavlja iznos pojedine transakcije.
- Opis (Description) - koristi se za dodavanje opcionalnog opisa transakcije. Ovo svojstvo omogućuje korisnicima dodavanje dodatnih informacija o transakciji, kao što su detalji ili bilješke.
- Datum i vrijeme (DateTime) - koristi se za prikaz datuma izvršenja transakcije. Zadana vrijednost je trenutni datum i vrijeme. Ovo svojstvo omogućuje praćenje kada je transakcija obavljena, što je korisno za analizu i izvještavanje.
- ID korisnika (UserId) - Predstavlja jedinstveni identifikator korisnika kojem transakcija pripada.
- Korisnik (IdentityUser User) - predstavlja navigacijsko svojstvo koje omogućava dohvat povezanog korisnika.
- Naslov kategorije (CategoryTitle) - služi kako bi unutar određenih pogleda vraćalo naslov kategorije, atribut definiran za ovo svojstvo označava kako ono nije mapirano na kolonu u bazi podataka.
- Formatirani iznos (FormattedAmount) - provjerava da li je tip kategorije trošak te, u slučaju da je, iznos će imati negativan predznak, inače će biti pozitivan: formatiranje uključuje valutu euro i svojstvo nije mapirano kako bi se izbjeglo mapiranje na kolonu u bazi podataka.

```
public class Transaction
{
    [Key]
    public int TransactionId { get; set; }
    [Range(1, int.MaxValue, ErrorMessage = "Select a category.")]
    public int CategoryId { get; set; }
    public Category? Category { get; set; }
    [Range(1, int.MaxValue, ErrorMessage = "Amount should be greater than 0.")]
    public int Amount { get; set; }
    [Column(TypeName = "nvarchar(75)")]
    public string? Description { get; set; }
}
```

```

public DateTime DateTime { get; set; } = DateTime.Now;

public string UserId { get; set; }
public IdentityUser User { get; set; }

[NotMapped]
public string? CategoryTitle
{
    get
    {
        return Category == null ? "" : Category.Title;
    }
}

[NotMapped]
public string? FormattedAmount
{
    get
    {
        return ((Category == null || Category.Type ==
"Expense")? "- " : "+ ") + Amount.ToString("'€'0");
    }
}
}

```

Kôd 2. Svojstva i pravila za transakcije

### 4.3. Postavljanje baze podataka i tablica

Za interakciju s bazom podataka i za upravljanje podacima koristi se Entity Framework Core. Proces postavljanja baze podataka uključuje definiranje konteksta aplikacijske baze podataka, dodavanje ovisnosti kroz mehanizam ubrizgavanja ovisnosti (engl. *Dependency Injection*), konfiguraciju stringa za povezivanje (engl. *Connection String*) te kreiranje migracije i ažuriranje baze podataka.

#### 4.3.1. Definiranje konteksta aplikacijske baze podataka

Definiranje konteksta aplikacijske baze podataka (engl. *Application DbContext*) ključno je za interakciju s bazom podataka kada se koristi Entity Framework Core. Kontekst predstavlja poveznicu između aplikacije i baze podataka, omogućujući aplikaciji pristup i upravljanje podacima (Kôd 3).

```

public class ApplicationDbContext : IdentityDbContext
{
    public
ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)

```

```

    {
    }

    public DbSet<Transaction> Transactions { get; set; }
    public DbSet<Category> Categories { get; set; }
}

```

#### Kôd 3. ApplicationDbContext klasa

U ovom kôdu, klasa `ApplicationDbContext` nasljeđuje klasu `IdentityDbContext`, koja omogućuje integraciju funkcionalnosti identiteta i autentifikacije unutar aplikacije. Konstruktor klase prima parametar opcije (engl. *Options*) koji sadrži postavke za kontekst baze podataka i prosljeđuje ga baznoj klasi `IdentityDbContext`. Svojstvo transakcije (engl. *Transactions*) predstavlja kolekciju transakcija unutar baze podataka. Svaka transakcija je entitet koji se pohranjuje u tablici transakcija. Ista stvar vrijedi i za svojstvo kategorije (engl. *Categories*) koja predstavlja kolekciju kategorija unutar baze podataka te također, svaka kategorija je entitet koji se pohranjuje u bazi podataka.

#### 4.3.2. Dodavanje mehanizma za ubrizgavanje ovisnosti

Mehanizam za ubrizgavanje ovisnosti (engl. *Dependency Injection*) je obrazac koji omogućuje da se objekti (servisi) injektiraju u druge objekte, npr. kontrolere i druge servise, tijekom izvršavanja (Kôd 4), čime se olakšava upravljanje ovisnostima i povećava modularnost i olakšava testiranje aplikacije.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDatabaseDeveloperPageExceptionFilter();

    services.AddDefaultIdentity<IdentityUser>(options =>
        options.SignIn.RequireConfirmedAccount = false)
        .AddEntityFrameworkStores<ApplicationDbContext>();
    services.AddControllersWithViews();
}

```

#### Kôd 4. Ubrizgavanje ovisnosti

Prvi servis ove metode registrira kontekst baze podataka s opcijama za korištenje SQL Servera te preuzima *ConnectionString* iz konfiguracijske datoteke. Drugi servis dodaje filter koji hvata iznimke vezane uz bazu podataka i prikazuje korisniku informativne stranice o greškama tijekom razvoja aplikacije. Treći servis postavlja uslugu identiteta, koristeći `ApplicationDbContext` za pohranu korisničkih podataka.

Opcija `SignIn.RequireConfirmedAccount` postavljena na netočno znači da korisnici ne trebaju potvrditi svoje račune putem e-pošte kako bi se prijavili. Posljednji servis omogućava podršku za MVC arhitekturu u ASP.NET Core aplikaciji, uključujući kontrolere i poglede.

### 4.3.3. Konfiguracija Connection Stringa

Konfiguracija *Connection Stringa* ključna je za uspostavu veze između ASP.NET Core aplikacije i baze podataka. U `appsettings.json` datoteci definira se *ConnectionStrings* sekcija koja sadrži detalje potrebne za povezivanje s određenom bazom podataka (Kôd 5). *Connection String* je primijenjen unutar `Startup.cs` klase u dijelu kôda koji je prikazan u prethodnom dijelu (Kôd 4).

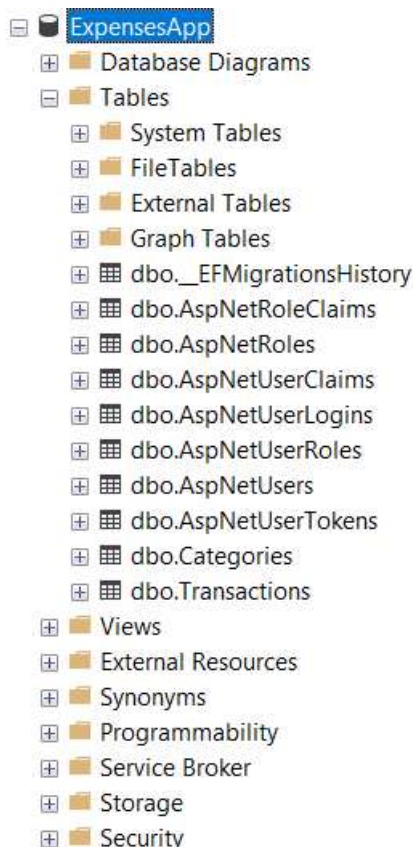
```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=DESKTOP-
UBN63ES;Database=ExpensesApp;Trusted_Connection=True;MultipleActiveR
esultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Kôd 5. Spajanje aplikacije s bazom podataka

### 4.3.4. Kreiranje migracije i ažuriranje baze podataka

Kreiranje i ažuriranje baze podataka u ASP.NET Core aplikaciji obavlja se kroz *Entity Framework Core* migracije. Migracije su način da se struktura baze podataka uskladi s promjenama u modelima podataka aplikacije, omogućujući sigurno i kontrolirano dodavanje ili promjenu tablica i shema. Da bi se migracije kreirale i primijenile na bazu podataka koristi se konzola upravitelja paketima (engl. *Package Manager Console*) unutar Visual Studio okruženja. Naredba koja je korištena za kreiranje migracije imena `CreateCategoryAndTransaction` je `Add-Migration CreateCategoryAndTransaction`, ovom naredbom se generiraju skupovi migracijskih skripti na temelju promjena u modelima podataka. Druga naredba koja se koristi za primjenu stvorene migracije na bazu podataka jest `Update-Database`. Ova naredba automatski izvršava SQL skripte generirane tijekom prethodnog koraka kako bi se struktura baze podataka ažurirala u skladu s definicijama modela. Nakon izvršavanja

prethodno navedenih naredbi, unutar SQL Server Management konzole bit će vidljiva baza imena ExpensesApp i njene tablice (Slika 6).



Slika 6. Baza podataka i tablice

#### 4.4. Implementacija kontrolera

U ovom dijelu objašnjava se implementacija kontrolera koji upravljaju različitim aspektima aplikacije, poput kategorija, transakcija i statistika. Kontroleri služe kao posrednici između modela i pogleda, omogućujući korisnicima interakciju s podacima na intuitivan i organiziran način. Detaljno je prikazano kako su kreirani kontroleri za pojedine funkcionalnosti aplikacije.

##### 4.4.1. Kontroler za kategorije

Kontroler za kategorije omogućuje korisnicima pregled, kreiranje, uređivanje i brisanje kategorija unutar aplikacije. Ovaj kontroler koristi Entity Framework Core za interakciju s bazom podataka i osigurava da samo autorizirani korisnici mogu pristupiti njegovim funkcionalnostima tako što je definiran atribut Autoriziraj (engl. *Authorize*) za kontroler.

Konstruktor prihvaća instancu ApplicationDbContext koja se koristi za pristup podacima iz baze podataka (Kôd 6). UserManager objekt se ovdje koristi kako bi svaki korisnik dobio prikaz samo svojih kategorija.

```
private readonly ApplicationDbContext _context;  
private readonly UserManager<IdentityUser> _userManager;
```



```

public CategoryController(ApplicationDbContext context,
    UserManager<IdentityUser> userManager)
{
    _context = context;

    _userManager = userManager;
}

```

#### Kôd 6. Konstruktor

Index metoda dohvaća trenutnog korisnika i prikazuje sve kategorije koje pripadaju tom korisniku (Kôd 7).

```

public async Task<IActionResult> Index()
{
    var user = await _userManager.GetUserAsync(User);
    if (user == null)
    {
        return NotFound($"Unable to load user with ID
'_{userManager.GetUserId(User)}'.");
    }

    var categories = await _context.Categories
        .Where(c => c.UserId == user.Id)
        .ToListAsync();

    return View(categories);
}

```

#### Kôd 7. Index metoda

GET metoda za kreiranje ili ažuriranje kategorija prikazuje pogled za kreiranje nove kategorije ili uređivanje postojeće, ovisno o tome je li prosljeđen ID.

POST metoda za kreiranje ili ažuriranje kategorija ima nekoliko funkcionalnosti (Kôd 8). Prva je provjeravanje je li model validan. Druga funkcionalnost je provjera postoji li kategorija s istim naslovom za trenutnog korisnika kako ne bi došlo do multipliciranja istih kategorija. Treća funkcionalnost po provjeri ID-a kategorije stvara novu kategoriju u slučaju da je ID jednak nuli, inače ažurira postojeću. Posljednja funkcionalnost je spremanje promjena u bazu podataka i preusmjeravanje nazad na pogled s listom kategorija.

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult>
CreateOrEdit([Bind("CategoryId,Title,Type")] Category category)
{
    if (ModelState.IsValid)
    {

```

```

        // Check if category with the same title already exists for
the current user
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        var existingCategory = await _context.Categories
            .FirstOrDefaultAsync(c => c.Title == category.Title &&
c.UserId == userId);

        if (existingCategory != null && existingCategory.CategoryId
!= category.CategoryId)
        {
            // Category with the same title exists, add a model
state error
            ModelState.AddModelError("Title", "A category with this
title already exists.");
            return View(category);
        }

        if (category.CategoryId == 0)
        {
            // New category creation
            category.UserId = userId;
            _context.Add(category);
        }
        else
        {
            // Update existing category
            var existingCategoryToUpdate = await
_context.Categories.FindAsync(category.CategoryId);
            if (existingCategoryToUpdate != null)
            {
                existingCategoryToUpdate.Title = category.Title;
                existingCategoryToUpdate.Type = category.Type;
                _context.Update(existingCategoryToUpdate);
            }
            else
            {
                return NotFound();
            }
        }

        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(category);
}

```

Kôd 8. POST metoda za kreiranje ili ažuriranje kategorije

Posljednja POST metoda unutar kontrolera služi za brisanje kategorije iz baze podataka (Kôd 9).

```

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    if (_context.Categories == null)
    {
        return Problem("Entity set 'ApplicationDbContext.Categories'
is null.");
    }
    var category = await _context.Categories.FindAsync(id);
    if (category != null)
    {
        _context.Categories.Remove(category);
    }

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

```

Kôd 9. POST metoda za brisanje postojećih kategorija

#### 4.4.2. Kontroler za transakcije

Kontroler za transakcije omogućuje korisnicima pregled, kreiranje, uređivanje i brisanje transakcija unutar aplikacije. Ovaj kontroler koristi Entity Framework Core za interakciju s bazom podataka, te kao i kod kontrolera za kategorije, osigurava da samo autorizirani korisnici mogu pristupiti njegovim funkcionalnostima.

Index metoda ima istu svrhu kao i kod kontrolera za kategorije samo što se umjesto kategorija dohvaćaju transakcije koje pripadaju trenutnom korisniku te se potom prikazuju u listi transakcija.

GET metoda za kreiranje ili ažuriranje transakcija prikazuje pogled za kreiranje nove transakcije ili uređivanje postojeće, ovisno o tome je li prosljeđen ID. Također poziva metodu za popunjavanje kategorija (engl. *PopulateCategories*).

POST metoda za kreiranje ili ažuriranje transakcija kao i kod kategorija, ima nekoliko funkcionalnosti (Kôd 10). Prva funkcionalnost je provjeravanje validnosti modela. Druga funkcionalnost je povezivanje transakcije s trenutnim korisnikom, ako je ID transakcije jednak nuli, dodaje novu transakciju u bazu podataka, a ako nije nula onda ažurira postojeću transakciju. Treća funkcionalnost je spremanje ili ažuriranje transakcije u bazi podataka. U slučaju greške bilježi iznimke i dodaje greške u ModelState koji bilježi greške validacije podataka prilikom obrade zahtjeva i omogućava njihovo prikazivanje korisniku.

```

[HttpPost]
[ValidateAntiForgeryToken]

```

```

public async Task<IActionResult>
CreateOrEdit([Bind("TransactionId,CategoryId,Amount,Description,Date
Time")] Transaction transaction)
{
    _logger.LogInformation("CreateOrEdit POST accessed with
transaction: {@Transaction}", transaction);

    if (ModelState.IsValid)
    {
        try
        {
            // Associate the transaction with the current user
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);
            transaction.UserId = userId;

            // Save the transaction
            if (transaction.TransactionId == 0)
            {
                _context.Add(transaction);
            }
            else
            {
                _context.Update(transaction);
            }
            await _context.SaveChangesAsync();

            return RedirectToAction(nameof(Index));
        }
        catch (Exception ex)
        {
            // Handle exception
            _logger.LogError(ex, "Error occurred while saving
transaction.");
            ModelState.AddModelError("", "An error occurred while
saving the transaction.");
        }
    }

    // Log ModelState errors
    foreach (var state in ModelState.Values)
    {
        foreach (var error in state.Errors)
        {
            _logger.LogError("ModelState error: {ErrorMessage}",
error.ErrorMessage);
        }
    }
}

```

```

    PopulateCategories();
    return View(transaction);
}

```

Kôd 10. POST metoda za kreiranje ili ažuriranje transakcije

POST metoda za brisanje postojećih transakcija (engl. *DeleteConfirmed*) dohvaća transakciju iz baze podataka prema proslijeđenom ID-u i, ako transakcija postoji, uklanja ju iz baze podataka. Promjene se spremaju i korisnik se vraća na prikaz popisa transakcija. Kôd za brisanje transakcija je isti kao i onaj za brisanje kategorija samo što se brišu transakcije.

Metoda za popunjavanje kategorija popunjava padajući izbornik kategorijama koje su prethodno kreirane od strane trenutno prijavljenog korisnika (Kôd 11).

```

[NonAction]
public void PopulateCategories()
{
    // Retrieve the current user's ID
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    // Retrieve categories associated with the current user
    var categoryCollection = _context.Categories
        .Where(c => c.UserId == userId || c.UserId == null) //
    Include categories with no specified user
        .ToList();

    // Add a default category
    var defaultCategory = new Category() { CategoryId = 0, Title =
    "Choose a Category" };
    categoryCollection.Insert(0, defaultCategory);

    // Pass the filtered categories to the view
    ViewBag.Categories = categoryCollection;
}

```

Kôd 11. Metoda za popunjavanje kategorija

#### 4.4.3. Kontroler za statistike

Kontroler za statistike koristi se za dohvaćanje i prikazivanje financijskih podataka korisnika u obliku različitih statističkih prikaza. Ovaj kontroler omogućava pregled prihoda, troškova i ukupnog stanja na računu korisnika za određeni mjesec i godinu, te tri grafikona od kojih jedan prikazuje ukupne troškove kroz odabrani mjesec, drugi prikazuje prihod naspram troška za odabrani mjesec i treći prikazuje stanje na računu kroz cijelu godinu. Kao i kod kategorija i transakcija, statistikama se ne može pristupiti bez autorizacije. Jedna od prvih funkcionalnosti unutar metode Index je filter datuma, ako korisnik nije odabrao godinu i mjesec, postavljaju se trenutna godina i mjesec. StartDate i EndDate predstavljaju prvi i posljednji dan odabranog

mjeseca kako bi cijeli mjesec mogao biti obuhvaćen. Metoda dohvaća sve transakcije za trenutnog korisnika koje spadaju unutar odabranog razdoblja.

Sljedeći dio kôda (Kôd 12) izračunava ukupne mjesečne prihode i troškove za odabrani mjesec.

Podaci se spremaju u ViewBag kako bi bili dostupni u pogledu.

```
// Total Income for the selected month
int TotalMonthlyIncome = SelectedTransactions
    .Where(i => i.Category.Type == "Income")
    .Sum(j => j.Amount);
ViewBag.TotalIncome = TotalMonthlyIncome.ToString("'€'0");

// Total Expense for the selected month
int TotalMonthlyExpense = SelectedTransactions
    .Where(i => i.Category.Type == "Expense")
    .Sum(j => j.Amount);
ViewBag.TotalExpense = TotalMonthlyExpense.ToString("'€'0");
```

Kôd 12. Izračun prihoda i troškova za odabrani mjesec

Sljedeća funkcionalnost na kontroleru za statistike služi kako bi se dobio potpuni iznos stanja na računu oduzimanjem sveukupnih troškova od sveukupnih prihoda (Kôd 13).

```
// Cumulative Transactions for all time for the current user
List<Transaction> CumulativeTransactions = await
_context.Transactions
    .Include(x => x.Category)
    .Where(t => t.UserId == userId)
    .ToListAsync();

// Total Cumulative Income
int TotalCumulativeIncome = CumulativeTransactions
    .Where(i => i.Category.Type == "Income")
    .Sum(j => j.Amount);

// Total Cumulative Expense
int TotalCumulativeExpense = CumulativeTransactions
    .Where(i => i.Category.Type == "Expense")
    .Sum(j => j.Amount);

// Balance (all-time)
int Balance = TotalCumulativeIncome - TotalCumulativeExpense;
ViewBag.Balance = Balance.ToString("'€'0");
```

Kôd 13. Izračun sveukupnog stanja na računu

Prvi grafikon se odnosi na prikaz svih troškova unutar odabranog mjeseca tako što se grupiraju transakcije prema kategoriji troškova i sumiraju iznosi za svaku kategoriju (Kôd 14).

```
ViewBag.CircularChartData = SelectedTransactions
    .Where(i => i.Category.Type == "Expense")
```

```

        .GroupBy(j => j.Category.CategoryId)
        .Select(k => new
        {
            categoryTitle = k.First().Category.Title,
            amount = k.Sum(j => j.Amount),
            formattedAmount = k.Sum(j => j.Amount).ToString("'€'0"),
        })
        .OrderByDescending(l => l.amount)
        .ToList();

```

Kôd 14. Grafikon za praćenje svih troškova unutar odabranog mjeseca

Podaci za grafikon koji prikazuje prihode naspram troškova za odabrani mjesec se generiraju grupiranjem transakcija prema datumu. Prihodi i troškovi se spajaju kako bi se dobio cjelokupni pregled za svaki dan u mjesecu. Unutar kôda, prvo se prihodi (transakcije s kategorijom *Income*) grupiraju po datumu, a zatim se za svaki dan izračunava ukupni prihod, dok se troškovi inicijaliziraju na 0. Slično se troškovi (transakcije s kategorijom *Expense*) grupiraju po datumu, izračunava se ukupni trošak za svaki dan, a prihodi se inicijaliziraju na 0. Na kraju se podaci o prihodima i troškovima spajaju, grupiraju po danu i sumiraju kako bi se dobio cjelokupni pregled prihoda i troškova za svaki dan u mjesecu. Rezultati se pohranjuju u `ViewBag.Chart3dData` za prikaz u pogledu.

Posljednji grafikon prikazuje stanje na računu kroz godinu, odnosno od početka godine do trenutno odabranog mjeseca. Kôd postavlja popise godina i mjeseci za padajuće izbornike, omogućujući korisniku odabir željene godine i mjeseca, pri čemu je trenutni mjesec postavljen kao zadani odabir. Zatim se generira grafikon koji prikazuje kumulativno stanje na računu od početka godine do odabranog mjeseca. Kumulativni saldo se računa za svaki mjesec tako da se za svaki mjesec zbrajaju prihodi i oduzimaju troškovi, a rezultati se pohranjuju unutar `ViewBag.CumulativeBalanceData` za prikaz u pogledu.

`SplineChartData` i `Chart3dData` su pomoćne klase koje se koriste za strukturiranje podataka za prikazivanje u grafikonima. `SplineChartData` sadrži podatke o mjesecima i stanju računa, dok `Chart3dData` sadrži podatke o danima, prihodima i troškovima.

#### 4.5. Razvoj korisničkog sučelja

U ovom dijelu opisano je konfiguriranje pogleda za kategorije, transakcije i statistike. Kako bi se ostvario ljepši prikaz i dizajn ovih pogleda koristili su se Syncfusion elementi.

##### 4.5.1. Pogledi za kategorije

Pogled `Index.cshtml` prikazuje popis svih kategorija u obliku tablice s podrškom za sortiranje i paginaciju (Slika 7). Ovaj pogled koristi komponentu za prikazivanje podataka u tablici, gdje su podaci o kategorijama dohvaćeni iz modela. Tablica omogućuje sortiranje i prikaz podataka

po stranicama, s maksimalno pet kategorija po stranici. Gumbi za uređivanje vode do pogleda za stvaranje ili uređivanje kategorije, dok gumbi za brisanje šalju zahtjev za brisanje odabrane kategorije.

## Categories

[+ New Category](#)

Category	Type	Actions
Salary	Income	<a href="#">✎</a> <a href="#">🗑️</a>
Taxi	Expense	<a href="#">✎</a> <a href="#">🗑️</a>
Shopping	Expense	<a href="#">✎</a> <a href="#">🗑️</a>
Party	Expense	<a href="#">✎</a> <a href="#">🗑️</a>
ATM Payout	Expense	<a href="#">✎</a> <a href="#">🗑️</a>

< 1 2 3 > 1 of 3 pages (12 items)

Slika 7. Izgled liste kategorija


Pogled CreateOrEdit.cshtml omogućava korisnicima stvaranje nove kategorije ili uređivanje postojeće (Slika 8). Forma je povezana s akcijama stvaranja ili uređivanja na kontroleru. Korisnici mogu unijeti naziv kategorije, odabrati vrstu kategorije, prihod ili trošak, i spremiti podatke putem gumba za slanje forme. Također, forma sadrži link za povratak na popis kategorija. Skripte za validaciju osiguravaju ispravno ispunjavanje svih polja.

## Create a new category

Title

EXPENSE  INCOME

[Back to Category list](#)



Slika 8. Izgled kreiranja nove kategorije



#### 4.5.2. Pogledi za transakcije

Pogled Index.cshtml prikazuje popis svih transakcija u tabličnom obliku s mogućnošću sortiranja i straničenja (Slika 9). Tablica omogućuje prikaz maksimalno pet transakcija po stranici. Gumbi za uređivanje vode do pogleda za stvaranje ili uređivanje transakcije, dok gumbi za brisanje šalju zahtjev za brisanje odabrane transakcije.

## Transactions

[+ New Transaction](#)

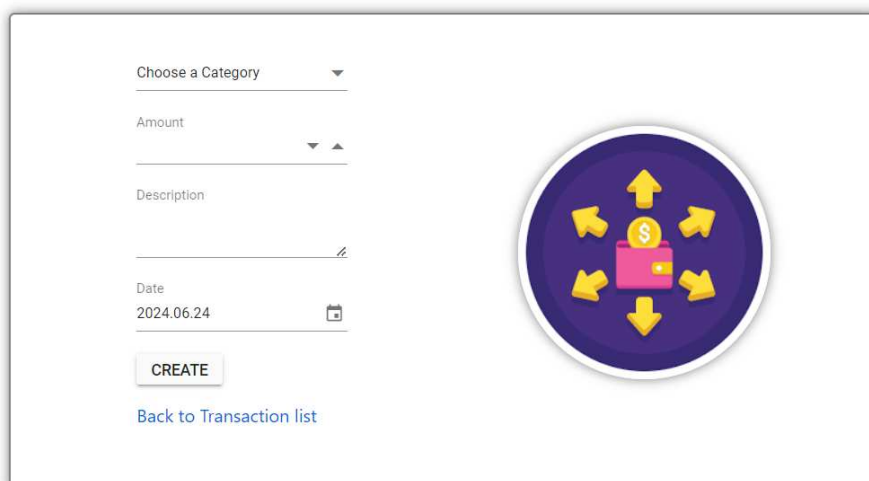
Transaction	Amount	Date	Actions
Taxi	- €8	01.01.2024	<a href="#">✎</a> <a href="#">🗑</a>
ZET Card	- €13	02.01.2024	<a href="#">✎</a> <a href="#">🗑</a>
ATM Payment	+ €255	02.01.2024	<a href="#">✎</a> <a href="#">🗑</a>
Balance from previous year	+ €690	01.01.2024	<a href="#">✎</a> <a href="#">🗑</a>
Shopping	- €302	03.01.2024	<a href="#">✎</a> <a href="#">🗑</a>

1 of 11 pages (55 items)

Slika 9. Izgled liste transakcija

Pogled CreateOrEdit.cshtml omogućava korisnicima stvaranje nove transakcije ili uređivanje postojeće (Slika 10). Forma je povezana s akcijom CreateOrEdit na kontroleru i koristi nekoliko komponenata za unos podataka: padajući izbornik za odabir kategorije transakcije, numerički okvir za unos iznosa s oznakom valute, opcionalni tekstualni okvir za opis transakcije i birač datuma za odabir datuma izvršenja. Korisnici mogu spremiti podatke klikom na gumb za spremanje ili se vratiti na popis transakcija putem linka za povratak. Skripte za validaciju forme osiguravaju ispravno ispunjavanje svih polja.

## Create a new transaction

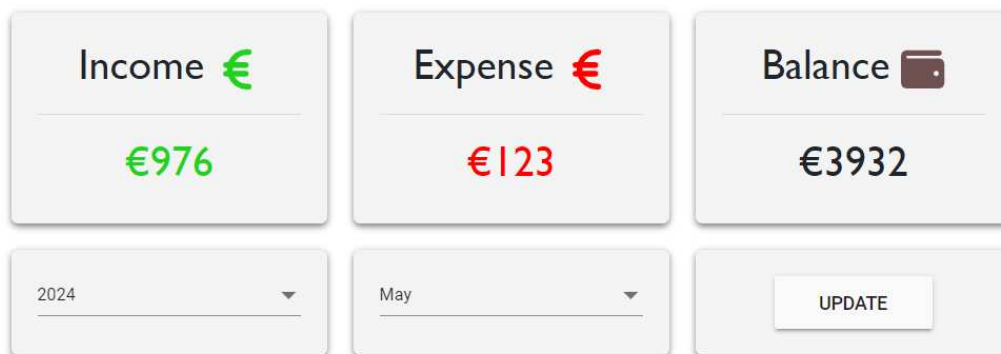


Slika 10. Izgled kreiranja nove transakcije

### 4.5.3. Pogled za statistike

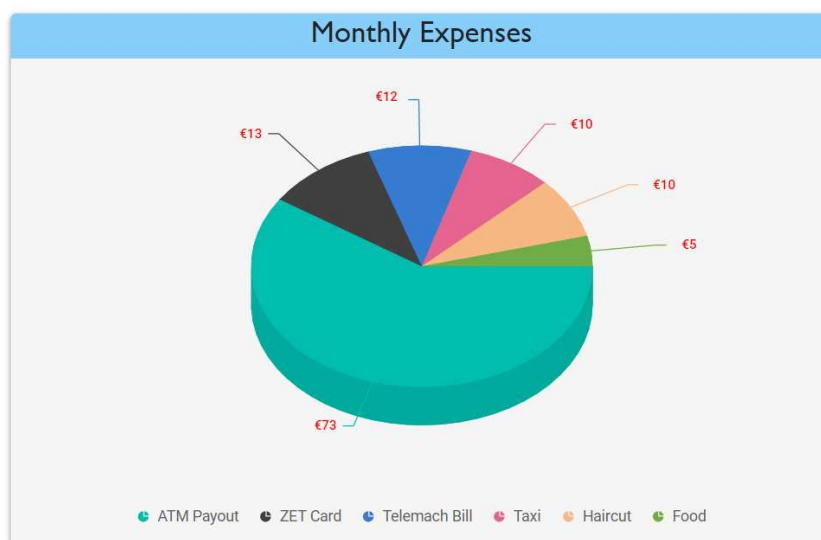
Pogled za statistike je samo jedan jer je njegova svrha samo prikaz prethodno kreiranih kategorija i transakcija. Pri kreiranju filtera datuma koristili su se padajući izbornici te gumb koji će napraviti nužnu izmjenu podataka i grafikona na odabrani datum, a okviri unutar kojih se nalaze prihod, trošak i stanje na računu postignuti su korištenjem CSS-a i div elemenata (Slika 11).

## Statistics



Slika 11. Izgled filtera datuma i prikaza prihoda, troškova i stanja na računu

Kako bi se ostvario izgled prvog grafikona koristio se kružni Syncfusion grafikon kojeg se samo trebalo konfigurirati po vlastitom ukusu. Neke od opcija koje su postavljene za ovaj grafikon su detaljni prikaz podataka o transakciji prilikom prelaska mišem preko nekog polja u grafikonu, te su sva polja imenovana ispod samog grafikona i uz njih stoji boja kojom su prikazane u grafikonu (Slika 12).



Slika 12. Izgled grafikona za mjesečne troškove

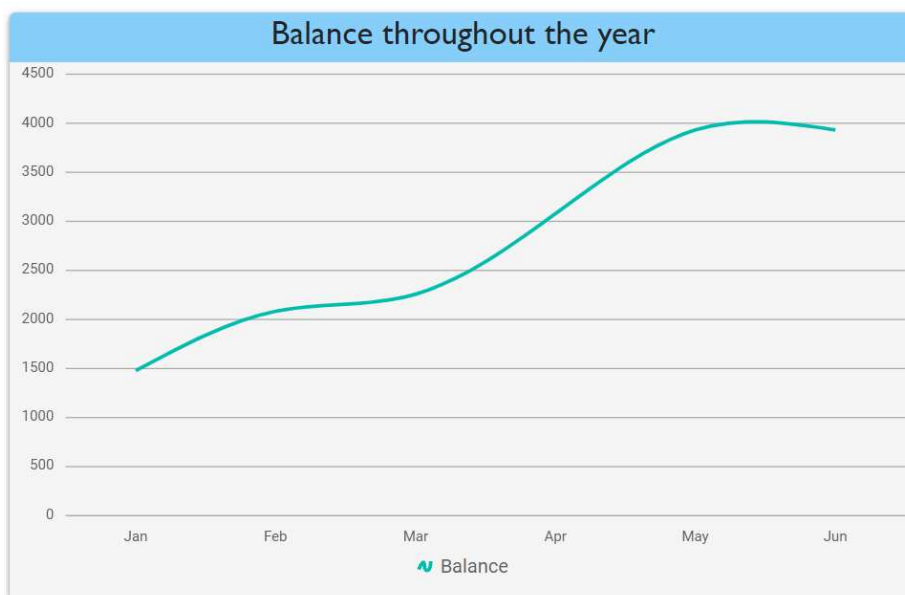
Za stvaranje drugog grafikona koristio se 3D stupčasti Syncfusion grafikoni koji ima tablični prikaz. Za svaki datum kad je izvršena transakcija prikazuje se usporedba prihoda koji je u zelenoj boji, naspram troška koji je u crvenoj boji. Ispod grafikona je za dodatnu jasnoću dodana i opcija za oznaku prihoda i troška skupa s bojom (Slika 13). Prelaskom miša preko stupaca mogu se vidjeti iznosi i datumi transakcija.



Slika 13. Izgled grafikona mjesečnih prihoda naspram troškova

Za stvaranje posljednjeg grafikona koristio se spline Syncfusion grafikoni kako bi se ostvario prikaz stanja na računu kroz godinu. S lijeve strane je prikazan iznos, a pri dnu su prikazani mjeseci. Linija koja predstavlja rast ili pad stanja računa je prikazana zelenom bojom i

naznačena pri dnu grafikona za bolju razumljivost istog (Slika 14). Prelaskom miša preko linije mogu se vidjeti iznosi stanja računa za određene mjesece.



Slika 14. Izgled grafikona za stanje računa kroz godinu

## 5. ZAKLJUČAK

Svrha ovog završnog rada je bila kreirati aplikaciju koja će korisniku omogućiti da na jednostavan način prati svoje financije i upravlja istima. Tri ključne komponente aplikacije su kategorije, transakcije i statistike koje skupa pružaju pregled financijskih aktivnosti korisnika. Pri razvijanju ovih komponenti ciljalo se ponajviše na jednostavnost uporabe s obzirom na to da praćenje transakcija prosječnoj osobi zna biti naporno te ponekad zbunjujuće. Kako bi se spriječilo takvo negativno iskustvo fokus je bio na osnovnim funkcionalnostima kao što su kreiranje, brisanje i ažuriranje kategorija i transakcija te unos vrijednosti za iznos transakcije, datum izvršenja i opcionalni opis za selektiranu kategoriju tijekom kreiranja transakcije, kao i definiranje imena kategorije i odabir tipa kategorije (trošak ili prihod). Kako bi se korisniku bolje dočaralo i prikazalo vlastite financijske aktivnosti ukomponirani su i grafikoni za lakše i razumljivije praćenje svih izvršenih transakcija te naravno kako korisnik ne bi bio ograničen ili zakinut za transakcije iz prethodnog mjeseca ili godine, omogućeno je i filtriranje transakcija za pojedini mjesec te pojedinu godinu.

Implementacija ASP.NET Core MVC arhitekture doprinijela je olakšanom razvoju, održavanju i testiranju aplikacije zbog jasne separacije odgovornosti između korištenih komponenti unutar aplikacije. SQL Server je osigurao efikasno i sigurno pohranjivanje i dohvaćanje podataka unutar baze, dok su NuGet paketi ubrzali razvoj aplikacije integracijom dodatnih funkcionalnosti i biblioteka.

Kako bi se korisnik osjećao sigurnije koristeći aplikaciju, implementirana je i funkcionalnost registracije i prijave te svaki prijavljeni korisnik vidi samo vlastite kategorije i transakcije kao i statistike vezane uz njih, također tri glavne komponente aplikacije se mogu koristiti tek nakon što je korisnik kreirao korisnički račun i prijavio se tim istim računom.

Zaključno, aplikacija za praćenje prihoda i troškova se pokazala kao koristan i vrijedan alat za osobno upravljanje financijama. S obzirom na to da je glavni fokus bio na jednostavnim dijelovima svih korištenih komponenti unutar aplikacije, ova aplikacija bi se još mogla proširiti implementacijom nešto kompleksnijih funkcionalnosti kao što su budžetiranje i planiranje, gdje bi korisnik mogao postaviti financijske ciljeve i pratiti njihov napredak, te integracija s bankovnim računima koja bi korisniku smanjila ručni unos transakcija i povećala točnost te automatski povlačila podatke o transakcijama. Uz to, dodavanje mogućnosti generiranja detaljnih financijskih izvještaja te automatsko slanje tih izvještaja putem e-maila bi dodatno poboljšalo korisničko iskustvo. Korisna nadogradnja bi također bila, implementacija opcije za analizu potrošačkih navika, koja bi korisnicima pružala uvid u njihove troškovne obrasce i preporuke za optimizaciju troškova. S obzirom na fleksibilnost i skalabilnost korištenih

tehnologija, ova aplikacija ima potencijal za daljnji razvoj i prilagodbu različitim potrebama korisnika.

## LITERATURA

1. Bužić, Dalibor, (2016), Baze podataka, Visoka škola za informacijske tehnologije
2. Syncfusion, <https://www.syncfusion.com/> (pristupljeno 8.6.2024.)
3. MVC, <https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-8.0> (pristupljeno 23.6.2024.)
4. NuGet paketi, <https://learn.microsoft.com/en-us/nuget/> (pristupljeno 29.6.2024.)
5. Entity Framework Core, <https://learn.microsoft.com/en-us/ef/core/> (pristupljeno 3.7.2024.)
6. Ubrizgavanje ovisnosti, <https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection> (pristupljeno 3.7.2024.)
7. ApplicationDbContext, <https://learn.microsoft.com/en-us/ef/core/dbcontext-configuration/> (pristupljeno 3.7.2024.)
8. Migracije, <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli> (pristupljeno 7.7.2024.)
9. HTML, <https://en.wikipedia.org/wiki/HTML> (pristupljeno 12.7.2024.)
10. CSS, [https://developer.mozilla.org/en-US/docs/Learn/Getting\\_started\\_with\\_the\\_web/CSS\\_basics](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics) (pristupljeno 12.7.2024.)
11. JavaScript, <https://ej2.syncfusion.com/aspnetcore/documentation/introduction> (pristupljeno 12.7.2024.)
12. C#, <https://dotnet.microsoft.com/en-us/languages/csharp> (pristupljeno 12.7.2024.)
13. ASP.NET Core, <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0> (pristupljeno 12.7.2024.)
14. .NET 6, <https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-6> (pristupljeno 12.7.2024.)

## **SAŽETAK**

Ovaj rad se fokusira na razvoj aplikacije za praćenje prihoda i troškova, koja korisnicima omogućava učinkovito upravljanje financijama putem kategorija, transakcija i statistika. Aplikacija je pisana u programskom jeziku C# te koristi ASP.NET Core MVC za strukturu i funkcionalnost te SQL Server za sigurno pohranjivanje i dohvaćanje podataka. Baza je povezana s aplikacijom uz pomoć Entity Framework Corea. Za sigurnije iskustvo i uporabu aplikacije, korisnici su dužni proći proces autorizacije, odnosno registrirati se i prijaviti.

**Ključne riječi:** SQL Server, ASP.NET Core MVC, Entity Framework Core, kategorije, transakcije, statistike, autorizacija



## **SUMMARY**

This thesis focuses on the development of an application for tracking income and expenses, enabling users to manage their finances efficiently through categories, transactions and statistics. The application is written in C# and utilizes ASP.NET Core MVC for structure and functionality, as well as SQL Server for secure data storage and retrieval. The database is connected to the application using Entity Framework Core. For a more secure experience and application usage, users are required to undergo an authorization process (register and log in).

**Keywords:** SQL Server, ASP.NET Core MVC, Entity Framework Core, categories, transactions, statistics, authorization