LIČNI PLANER BUDŽETA

WPF MVVM Aplikacija sa Entity Framework Core

Datum: 18.01.2026.

Projektna dokumentacija

Sadržaj

1. Uvod 3
2. Analiza 4
2.1. Use Case Dijagram 4
2.2. Opisi Use Case-ova 5
2.3. Korisničke Uloge 7
3. Modelovanje 8
3.1. Dijagram Klasa 8
3.2. Dijagram Paketa 9
3.3. Dijagrami Sekvenci 10
4. Implementacija 13
4.1. MVVM Arhitektura 13
4.2. Entity Framework Core 15
4.3. Dizajn Šabloni 16
4.4. Serijalizacija 18
4.5. PDF Izveštaji 19
5. Testiranje 20
6. Git i Verzionisanje 22
7. Zaključak 23

1. Uvod

Lični Planer Budžeta je desktop aplikacija razvijena koristeći Windows Presentation Foundation (WPF) sa .NET 6 platformom. Aplikacija omogućava korisnicima da efikasno upravljaju svojim ličnim finansijama kroz praćenje prihoda i rashoda, kategorisanje transakcija, postavljanje budžeta i generisanje detaljnih izveštaja.

1.1. Cilj Projekta

Cilj ovog projekta je razvoj potpuno funkcionalne desktop aplikacije koja demonstrira:

• Implementaciju MVVM arhitekturnog obrasca
• Korišćenje Entity Framework Core za pristup podacima
• Primenu dizajn šablona (Singleton, Factory, Observer)
• Serijalizaciju podataka u JSON i XML formatima
• Generisanje PDF izveštaja
• Implementaciju autentifikacije korisnika
• Jedinično testiranje kritičnih komponenti

1.2. Tehnologije

Kategorija	Tehnologija
UI Framework	WPF (.NET 6+), XAML
Arhitektura	MVVM (Model-View-ViewModel)
ORM	Entity Framework Core 6
Baza podataka	SQLite
Testiranje	MSTest
Serijalizacija	System.Text.Json, XmlSerializer
PDF Generisanje	iText7
Verzionisanje	Git, GitHub

2. Analiza

2.1. Use Case Dijagram

Napomena: PlantUML dijagrami se nalaze u folderu Documentation/UML. Dijagrami mogu biti pregledani koristeći PlantUML preglednike ili online alate na adresi: http://www.plantuml.com/plantuml

Use Case dijagram prikazuje sve glavne funkcionalnosti sistema koje su dostupne korisniku. Sistem podržava 8 glavnih slučajeva upotrebe sa dodatnim proširenjima i uključivanjima.

Glavni Use Case-ovi:

• UC1: Registracija/Login - Autentifikacija korisnika
• UC2: Upravljanje Kategorijama - CRUD operacije nad kategorijama
• UC3: Dodavanje Transakcija - Unos prihoda i rashoda
• UC4: Pregled Transakcija - Pregled i filtriranje podataka
• UC5: Postavljanje Budžeta - Definisanje mesečnih budžeta
• UC6: Generisanje Izveštaja - Kreiranje finansijskih izveštaja
• UC7: Export Podataka - Izvoz u JSON, XML i PDF formate
• UC8: Import Podataka - Uvoz prethodno izvezenih podataka

2.2. Opisi Use Case-ova

UC1: Registracija/Login

Atribut	Opis
Akteri	Korisnik
Preduslov	Aplikacija je pokrenuta
Tok doga■aja	1. Korisnik unosi korisni■ko ime i lozinku 2. Sistem validira unete podatke 3. Sistem kreira korisni■ku sesiju 4. Korisnik pristupa glavnom ekranu
Alternativni tok	Pogrešni kredencijali - prikazuje se poruka o grešci
Postuslov	Korisnik je autentifikovan i ima pristup sistemu

UC3: Dodavanje Transakcija

Atribut	Opis
Akteri	Autentifikovan korisnik
Preduslov	Korisnik je prijavljen i nalazi se na Transactions View
Tok doga■aja	1. Korisnik unosi iznos transakcije 2. Korisnik bira kategoriju 3. Korisnik unosi opis transakcije 4. Korisnik bira datum 5. Korisnik potvr■uje unos klikom na Add 6. Sistem validira podatke 7. Sistem ■uva transakciju u bazi 8. Lista transakcija se ažurira
Alternativni tok	Nevalidni podaci - prikazuje se poruka o grešci
Postuslov	Nova transakcija je sa■uvana u bazi podataka

UC6: Generisanje Izveštaja

Atribut	Opis
Akteri	Autentifikovan korisnik
Preduslov	Korisnik ima unete transakcije u sistemu
Tok doga■aja	1. Korisnik bira mesec i godinu za izveštaj 2. Sistem generiše mese■ni izveštaj 3. Prikazuje se statistika prihoda i rashoda 4. Korisnik može izvesti izveštaj u PDF format
Postuslov	Izveštaj je prikazan/izvezen

2.3. Korisničke Uloge

Aplikacija podržava samo jednu korisničku ulogu - Korisnik. Svaki korisnik ima pristup svim funkcionalnostima aplikacije nakon autentifikacije. Podaci su izolovani po korisniku - svaki korisnik vidi samo svoje transakcije, kategorije i budžete.

3. Modelovanje

3.1. Dijagram Klasa

Dijagram klasa prikazuje strukturu aplikacije sa svim glavnim klasama, njihovim atributima, metodama i relacijama. Aplikacija sadrži 9 glavnih klasa sa implementacijom nasleđivanja, kompozicije i agregacije.

Glavne Klase:

Klasa	Tip	Opis
Transaction	Apstraktna	Bazna klasa za sve transakcije
Income	Konkretna	Predstavlja prihod (nasledjuje Transaction)
Expense	Konkretna	Predstavlja rashod (nasledjuje Transaction)
Category	Apstraktna	Bazna klasa za kategorije
IncomeCategory	Konkretna	Kategorija prihoda
ExpenseCategory	Konkretna	Kategorija rashoda
User	Konkretna	Predstavlja korisnika sistema
Budget	Konkretna	Mese■ni budžet korisnika
MonthlyReport	Konkretna	Mese■ni finansijski izveštaj

Relacije:

• Nasleđivanje: Income i Expense nasleđuju Transaction
• Nasleđivanje: IncomeCategory i ExpenseCategory nasleđuju Category
• Kompozicija: User poseduje kolekciju Transactions (1:N)
• Kompozicija: User poseduje kolekciju Budgets (1:N)
• Agregacija: Transaction referencira Category (N:1)
• Interfejs: ViewModelBase implementira INotifyPropertyChanged

3.2. Dijagram Paketa

Dijagram paketa prikazuje organizaciju projekta u logičke celine (namespaces).
Projekat je organizovan prema MVVM arhitekturi sa jasnom separacijom odgovornosti.

Paket	Opis
Models	Domenski modeli (entiteti baze podataka)
ViewModels	ViewModel klase sa logikom prezentacije
Views	XAML prikazi korisni■kog interfejsa
Services	Servisni sloj (Repository, Factory, Export, Report)
Data	DbContext i konfiguracija baze
Commands	ICommand implementacije (RelayCommand)
Helpers	Helper klase (Converters, Extensions)

3.3. Dijagrami Sekvenci

Dijagrami sekvenci prikazuju interakcije između objekata tokom izvršavanja određenih use case-ova. Implementirana su tri dijagrama sekvenci koja pokrivaju ključne funkcionalnosti.

3.3.1. Login Sekvenca

Učesnici: Korisnik, LoginView, LoginViewModel, Repository, BudgetDbContext, UserSession

Tok:
1. Korisnik unosi kredencijale i klikće na Login dugme
2. LoginView poziva LoginCommand u LoginViewModel-u
3. LoginViewModel poziva Repository.ValidateUser()
4. Repository upituje BudgetDbContext
5. DbContext izvršava SQL upit i vraća User objekat
6. Repository vraća rezultat u ViewModel
7. LoginViewModel postavlja UserSession.CurrentUser
8. LoginView prikazuje MainView

3.3.2. Dodavanje Transakcije

Učesnici: Korisnik, TransactionView, TransactionViewModel, TransactionFactory, Repository, BudgetDbContext

Tok:
1. Korisnik unosi podatke o transakciji i klikće Add dugme
2. TransactionView poziva AddTransactionCommand
3. TransactionViewModel validira unete podatke
4. ViewModel poziva TransactionFactory.CreateTransaction()
5. Factory kreira Income ili Expense objekat (Factory pattern)
6. ViewModel poziva Repository.AddTransaction()
7. Repository dodaje transakciju u DbContext
8. DbContext čuva promene u bazi
9. ViewModel osvežava listu transakcija (Observer pattern)
10. View se ažurira kroz data binding

3.3.3. Generisanje Izveštaja

Učesnici: Korisnik, MainView, MainViewModel, ReportService, Repository, ExportService, BudgetDbContext

Tok:
1. Korisnik bira mesec/godinu i klikće Generate Report dugme
2. MainView poziva GenerateReportCommand
3. MainViewModel poziva ReportService.GenerateMonthlyReport()
4. ReportService poziva Repository za preuzimanje transakcija
5. Repository izvršava LINQ upit preko DbContext-a
6. ReportService kreira MonthlyReport objekat sa statistikom
7. MainViewModel prikazuje izveštaj
8. Opciono: Korisnik klikće Export to PDF
9. ExportService.ExportReportToPdf() kreira PDF dokument

10. Sistem čuva PDF fajl na disk

4. Implementacija

4.1. MVVM Arhitektura

Aplikacija je implementirana striktno prema MVVM (Model-View-ViewModel) arhitekturnom obrascu. Ovaj obrazac omogućava jasnu separaciju odgovornosti i olakšava testiranje i održavanje koda.

Model	• Domenski entiteti (User, Transaction, Category, Budget) • Entity Framework Core ma
View	• XAML datoteke sa UI definicijama • Minimalan code-behind (samo inicijalizacija)
ViewModel	• Logika prezentacije • ICommand implementacije (RelayCommand) • INotifyPrope

Primeri implementacije:

ViewModelBase sa INotifyPropertyChanged:

```
public class TransactionViewModel : ViewModelBase { private readonly IRepository _repository; private ObservableCollection<Transaction> _transactions; public ICommand AddTransactionCommand { get; } public TransactionViewModel(IRepository repository) { _repository = repository; AddTransactionCommand = new RelayCommand(AddTransaction); LoadTransactions(); } private void AddTransaction() { // Logika za dodavanje transakcije OnPropertyChanged(nameof(Transactions)); } }
```

4.2. Entity Framework Core

Entity Framework Core je korišćen kao ORM (Object-Relational Mapping) za pristup SQLite bazi podataka. Implementiran je Code-First pristup sa migracijama.

• User: Korisnik sistema (Id, Username, PasswordHash, Email, CreatedAt)
• Transaction: Apstraktna klasa za transakcije (Id, Amount, Description, Date, UserId, CategoryId)
• Income/Expense: Konkretne implementacije transakcija (TPH - Table Per Hierarchy)
• Category: Apstraktna klasa za kategorije (Id, Name, Color)
• IncomeCategory/ExpenseCategory: Konkretne kategorije (TPH)
• Budget: Mesečni budžet (Id, Month, Year, PlannedAmount, UserId, CategoryId)
• MonthlyReport: Izveštaj (Id, Month, Year, TotalIncome, TotalExpense, Balance)

BudgetDbContext konfiguracija:

```
public class BudgetDbContext : DbContext { public DbSet<User> Users { get; set; } public DbSet<Transaction> Transactions { get; set; } public DbSet<Category> Categories { get; set; } public DbSet<Budget> Budgets { get; set; } protected override void OnModelCreating(ModelBuilder modelBuilder) { // TPH konfiguracija za Transaction modelBuilder.Entity<Transaction>() .HasDiscriminator<string>("TransactionType") .HasValue<Income>("Income") .HasValue<Expense>("Expense"); // Relacije modelBuilder.Entity<Transaction>() .HasOne(t => t.User) .WithMany(u => u.Transactions) .HasForeignKey(t => t.UserId); } }
```

4.3. Dizajn Šabloni

U aplikaciji su implementirana dva dizajn šablona prema zahtevima projekta:

4.3.1. Singleton Pattern (Kreacioni)

Klasa: UserSession
Svrha: Obezbeđuje da postoji samo jedna instanca korisničke sesije u celoj aplikaciji. Koristi se za čuvanje trenutno ulogovanog korisnika.
Implementacija: Thread-safe Singleton sa lazy initialization.

```
public class UserSession { private static readonly Lazy<UserSession> _instance = new Lazy<UserSession>(() => new UserSession()); public static UserSession Instance => _instance.Value; public User CurrentUser { get; set; } public bool IsAuthenticated => CurrentUser != null; private UserSession() { } }
```

4.3.2. Factory Pattern (Kreacioni)

Klasa: TransactionFactory
Svrha: Kreira odgovarajući tip transakcije (Income ili Expense) na osnovu ulaznih parametara. Enkapsulira logiku kreiranja objekata.
Prednost: Centralizovano kreiranje objekata, laka proširivost.

```
public class TransactionFactory { public static Transaction CreateTransaction( string type, decimal amount, string description, DateTime date, Category category, User user) { return type.ToLower() switch { "income" => new Income { Amount = amount, Description = description, Date = date, Category = category, User = user }, "expense" => new Expense { Amount = amount, Description = description, Date = date, Category = category, User = user }, _ => throw new ArgumentException("Invalid transaction type") }; } }
```

4.3.3. Observer Pattern (Ponašajni)

Implementacija: INotifyPropertyChanged interfejs u ViewModelBase
Svrha: Automatsko obaveštavanje View-a o promenama u ViewModel-u. Omogućava reaktivno ažuriranje UI-a.
Mehanizam: PropertyChanged event koji se aktivira pri promeni svojstava.

```
public abstract class ViewModelBase : INotifyPropertyChanged { public event
PropertyChangedEventHandler PropertyChanged; protected void
OnPropertyChanged([CallerMemberName] string propertyName = null) {
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName)); } protected
bool SetProperty<T>(ref T field, T value, [CallerMemberName] string propertyName =
null) { if (EqualityComparer<T>.Default.Equals(field, value)) return false; field =
value; OnPropertyChanged(propertyName); return true; } }
```

4.4. Serijalizacija

Aplikacija podržava serijalizaciju i deserijalizaciju podataka u JSON i XML formatima. Implementiran je ExportService koji omogućava izvoz i uvoz podataka.

```
// JSON Serijalizacija public void ExportToJson(List<Transaction> transactions, string filePath) { var options = new JsonSerializerOptions { WriteIndented = true, PropertyNamingPolicy = JsonNamingPolicy.CamelCase }; var json = JsonSerializer.Serialize(transactions, options); File.WriteAllText(filePath, json); } // XML Serijalizacija public void ExportToXml(List<Transaction> transactions, string filePath) { var serializer = new XmlSerializer(typeof(List<Transaction>)); using var writer = new StreamWriter(filePath); serializer.Serialize(writer, transactions); }
```

4.5. PDF Izveštaji

Za generisanje PDF izveštaja koristi se iText7 biblioteka. ReportService kreira mesečne izveštaje sa statistikom prihoda i rashoda, koje ExportService konvertuje u PDF format.

```
public void ExportReportToPdf(MonthlyReport report, string filePath) { using var writer = new PdfWriter(filePath); using var pdf = new PdfDocument(writer); var document = new Document(pdf); // Naslov document.Add(new Paragraph($"Mese■ni Izveštaj - {report.Month}/{report.Year}") .SetFontSize(20).SetBold()); // Statistika document.Add(new Paragraph($"Ukupni Prihodi: {report.TotalIncome:C}")); document.Add(new Paragraph($"Ukupni Rashodi: {report.TotalExpense:C}")); document.Add(new Paragraph($"Bilans: {report.Balance:C}")); // Tabela transakcija var table = new Table(4); table.AddHeaderCell("Datum"); table.AddHeaderCell("Opis"); table.AddHeaderCell("Kategorija"); table.AddHeaderCell("Iznos"); foreach (var transaction in report.Transactions) { table.AddCell(transaction.Date.ToString("dd.MM.yyyy")); table.AddCell(transaction.Description); table.AddCell(transaction.Category.Name); table.AddCell(transaction.Amount.ToString("C")); } document.Add(table); document.Close(); }
```

5. Testiranje

Implementirano je jedinično testiranje (unit testing) kritičnih komponenti aplikacije koristeći MSTest framework. Testovi pokrivaju ViewModel logiku, Factory pattern i Singleton pattern.

5.1. Test Klase

Test Klasa	Broj Testova	Pokriva
TransactionViewModelTests	3	ViewModel logiku, data binding, validaciju
TransactionFactoryTests	2	Factory pattern, kreiranje objekata
UserSessionTests	3	Singleton pattern, autentifikaciju

5.2. Primeri Testova

```
[TestClass] public class TransactionFactoryTests { [TestMethod] public void CreateTransaction_IncomeType_ReturnsIncomeObject() { // Arrange var category = new IncomeCategory { Name = "Salary" }; var user = new User { Username = "test" }; // Act var transaction = TransactionFactory.CreateTransaction( "income", 1000m, "Monthly salary", DateTime.Now, category, user); // Assert Assert.IsInstanceOfType(transaction, typeof(Income)); Assert.AreEqual(1000m, transaction.Amount); } [TestMethod] public void CreateTransaction_ExpenseType_ReturnsExpenseObject() { // Arrange var category = new ExpenseCategory { Name = "Food" }; var user = new User { Username = "test" }; // Act var transaction = TransactionFactory.CreateTransaction( "expense", 50m, "Groceries", DateTime.Now, category, user); // Assert Assert.IsInstanceOfType(transaction, typeof(Expense)); Assert.AreEqual(50m, transaction.Amount); } } [TestClass] public class UserSessionTests { [TestMethod] public void Instance_ReturnsSameInstance() { // Act var instance1 = UserSession.Instance; var instance2 = UserSession.Instance; // Assert Assert.AreSame(instance1, instance2); } [TestMethod] public void IsAuthenticated_WithCurrentUser_ReturnsTrue() { // Arrange var session = UserSession.Instance; session.CurrentUser = new User { Username = "test" }; // Act & Assert Assert.IsTrue(session.IsAuthenticated); } }
```

5.3. Pokretanje Testova

Testovi se mogu pokrenuti iz Visual Studio-a preko Test Explorer-a ili korišćenjem komandne linije:

```
dotnet test BudgetPlanner.Tests/BudgetPlanner.Tests.csproj
```

6. Git i Verzionisanje

Projekat koristi Git za verzionisanje koda i GitHub za hosting repozitorijuma. Implementirana je strategija grananja (branching) sa feature granama i povremenim merge-ovima u main granu.

6.1. Struktura Repozitorijuma

```
BudgetPlanner/
■■■ BudgetPlanner.App/ # Glavni WPF projekat
■ ■■■ Models/ # Domenski modeli
■ ■■■ ViewModels/ # ViewModel klase
■ ■■■ Views/ # XAML prikazi
■ ■■■ Services/ # Servisni sloj
■ ■■■ Data/ # DbContext
■ ■■■ Commands/ # ICommand implementacije
■ ■■■ Helpers/ # Helper klase
■■■ BudgetPlanner.Tests/ # Test projekat
■■■ Documentation/ # Dokumentacija
■ ■■■ UML/ # PlantUML dijagrami
■■■ .gitignore # Git ignore fajl
■■■ README.md # Projekat README
■■■ BudgetPlanner.sln # Solution fajl
```

6.2. Commit Istorija

Projekat sadrži više od 15 commit-ova koji prate razvoj aplikacije od početne strukture do finalne verzije. Commit-ovi su pravilno imenovani i opisani.

1. Initial project structure with MVVM folders
2. Add Entity Framework Core and configure DbContext
3. Implement Transaction model with inheritance (TPH)
4. Add Category models with inheritance
5. Implement User and Budget models
6. Add Repository pattern implementation
7. Implement Singleton pattern for UserSession
8. Add Factory pattern for Transaction creation
9. Implement LoginViewModel and LoginView
10. Add TransactionViewModel with CRUD operations
11. Implement CategoryViewModel
12. Add BudgetViewModel
13. Implement ReportService for monthly reports
14. Add JSON and XML serialization in ExportService
15. Implement PDF export functionality
16. Add unit tests for ViewModels
17. Add unit tests for design patterns
18. Update README with project documentation
19. Add PlantUML diagrams
20. Final documentation and cleanup

6.3. Grane

- main - Glavna grana sa stabilnim kodom
- feature/ef-core-setup - Podešavanje Entity Framework Core
- feature/viewmodels - Implementacija ViewModel klasa
- feature/serialization - JSON i XML serijalizacija
- feature/testing - Dodavanje jediničnih testova

7. Zaključak

Projekat Lični Planer Budžeta uspešno demonstrira primenu MVVM arhitekture u WPF aplikacijama sa integracijom Entity Framework Core ORM-a. Kroz implementaciju aplikacije ostvareni su svi postavljeni zahtevi projekta:

• Funkcionalni zahtevi: Implementirano 8 glavnih use case-ova sa CRUD operacijama, filtiranjem, serijalizacijom i generisanjem izveštaja
• MVVM arhitektura: Striktna primena MVVM obrasca sa jasnom separacijom Model-View-ViewModel slojeva
• Entity Framework Core: Konfiguracija DbContext-a sa 5 entiteta, relacijama, migracijama i Table-Per-Hierarchy strategijom nasleđivanja
• Dizajn šabloni: Implementacija Singleton (UserSession), Factory (TransactionFactory) i Observer (INotifyPropertyChanged) šablona
• Serijalizacija: Podršk za JSON i XML formate sa mogućnošću eksporta i importa podataka
• PDF izveštaji: Generisanje profesionalnih mesečnih izveštaja sa tabelama i statistikom
• Testiranje: 8 jediničnih testova koji pokrivaju ViewModel logiku i dizajn šablone
• UML modelovanje: Kompletna dokumentacija sa Use Case, Class, Package i Sequence dijagramima
• Git verzionisanje: Preko 15 commit-ova sa feature granama i pravilnim commit porukama

Moguća proširenja aplikacije:
• Integracija sa bankovnim API-jem za automatski uvoz transakcija
• Dodavanje grafičkih prikaza (charts) za vizualizaciju budžeta
• Implementacija multi-user podrške sa različitim ulogama
• Cloud sinhronizacija podataka
• Mobilna aplikacija za praćenje rashoda u pokretu
• Machine learning predikcije budućih troškova

Aplikacija je u potpunosti funkcionalna, testirana i pripremljena za deployment. Izvorni kod je organizovan, dokumentovan i dostupan na GitHub-u. Sva dokumentacija, uključujući UML dijagrame i ovu PDF dokumentaciju, pruža kompletnu sliku arhitekture i implementacije projekta.