Технически Университет – София
Факултет Компютърни системи и технологии

Система за следене и анализиране на разходи – ExpenseIt

Курсов Проект по Програмни Среди

Николай Георгиев Станишев
Фак. № 121218038
Група: 51

18.09.2021

1 Съдържание

2 Въведение

Зададената тема е „Система за следене и анализиране на разходи”. Тази система ще бъде разработена, като приложение за Windows операционна система, чието име ще бъде „ExpenseIt”. Основната функционалност на това приложение, ще бъде въвеждане на разходи от потребителя в различни профили, които ще могат да бъдат анализирани по разнообразни критерии. По този начин, клиента ще може да бъде наясно с неговите разходи.

3 Анализ на съществуващи разработки

3.1 Spending Tracker[1]

Spending Tracker позволява следенето на потребителските приходи и разходи, като позволява добавянето на деня в който са се осъществили. Другото нещо, по което може да бъдат разделяни в това приложение е по категорията им. В това приложение има зададени категории, за различните приходи и разходи, но има възможност и за добавяне на такива от потребителя. Интерфейсът му представлява един екран, в който може да се добавят приходи и разходи, да се наблюдават същите в изглед, който е лист и да се наблюдават в диаграма тип пай, разделени в три категории – приходи, разходи и общ баланс. Платената версия на това приложение има и възможност за задаване на приходи и разходи, които се повтарят във времето.

3.2 MoneyBook[2]

MoneyBook представлява приложение, в което може да се проследяват приходите и разходите, като се разделят в различни категории. В това приложение има възможност за добавяне на описание на транзакцията. По отношение на взаимодействието с транзакциите, потребителя може да ги наблюдава, редактира и изтрива. Началният екран на приложението съдържа диаграма с приходите и разходите за последните седем дни, както и информация за текущия баланс.

3.3 Money Fox[3]

Money Fox представлява приложение за следене на приходите и разходите. В него има повечето от опциите, които осигуряват предните две разгледани приложения. Интерфейсът му наподобява много на някое от приложенията, разработени от Microsoft, с които идва чистата инсталация на Windows 10. По-интересното при него е, че то дава възможността на потребителя да запази информацията си в OneDrive акаунт.

4 Проектиране

4.1 Целева група

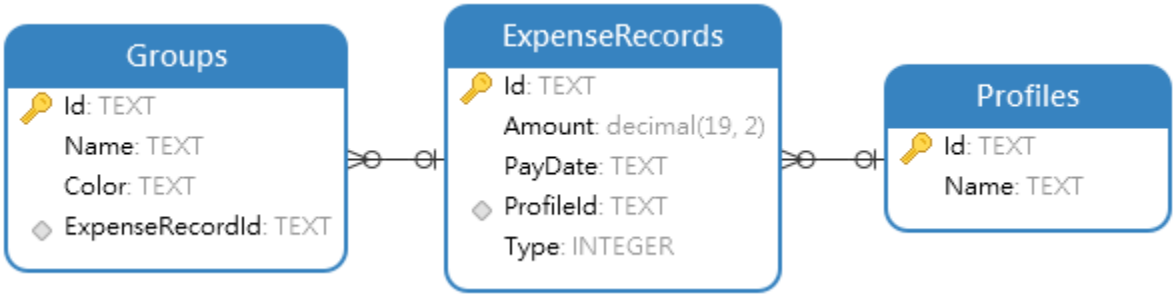
Очакваните потребители на това приложение са хора, които искат да имат поглед над своите финанси и по-точно да са наясно, за какво харчат своите пари. Благодарение на графиките, които ще бъдат имплементирани в приложението потребителя ще има възможността да знае всяка негова категория, разделена по групи, какъв дял е от неговите разходи и ако не му харесва, как изглежда ще може да направи корекция на разходите си в бъдеще.

Погледнато от гледна точка на възрастова група на потребителите, това се очаква да бъдат работещи хора, които искат да имат контрол върху финансите си.

4.2 Използвани данни

Изцяло всички данни в това приложение ще бъдат предоставени от потребителите му. За всеки потребител, те ще имат различни стойности. Различните потребители няма да имат достъп да гледат данните на другите.

Съхраняваните данни може да се разглеждат в три групи. Първата група са потребителските профили. Тези профили имат име и следващата група от данни, която са разходите. Нещата, които съдържат разходите са – размер, тип (от предефиниран изброим тип съдържащ категориите – електричество, вода, интернет, телевизия, горива и други), дата на плащането и следващата група от данни – групи. Групите съдържат две неща – име и цвят за визуализацията им в програмния интерфейс.



Фигура 1 ER Диаграма на базата от данни

4.3 Потребителски интерфейс

Разработеният потребителски интерфейс, представлява общ прозорец за различните изгледи, които ще могат да бъдат достъпни чрез меню.

В общият прозорец се намират няколко неща. Това са главна лента на програмата в горният и край, в която се намира името ѝ (ExpenseIt), бутони за минимизиране, отваряне в голям екран и затваряне на програмата, както и избор на профил с чиито данни да се оперира в различните изгледи. В левият края на прозореца може да се види менюто за избор на различните изгледи. Другото нещо, което се намира в този прозорец е контрола за промяна на размера на прозореца в долният десен ъгъл.

Изгледите, до които може да се стигне през менюто за техният избор са началното, за добавяне на разход, за изготвяне на статистики на база разходите и за добавяне на профили.

Началният изглед представлява таблица, която съдържа информация за разходите на избрания профил. Тази информация е размера, типа, датата на плащане и групите на разходите. Всеки един елемент от тази информация, може да се модифицира през таблицата. Всеки от редовете има и два бутона, единият е за обновяване на разхода (ако потребителя е направил промяна по него в таблицата) и другият е за изтриването на разхода.

Изгледа за добавяне на разход представлява форма, в която потребителя може да въведе един разход. Въвежданата информация е размера, типа и датата на плащане на разхода, след въвеждането на коректни данни потребителя може да натисне бутона в края на формата за запазването му в базата от данни.

Изгледа за изготвяне на статистики съдържа контрола за избор, на какво да бъдат групирани разходите. Възможните опции са групи, тип и месец на разхода. След избирането на едно от тези неща се изготвя диаграма от тип пай с разделяне на всички разходи, намиращи се в избраният профил.

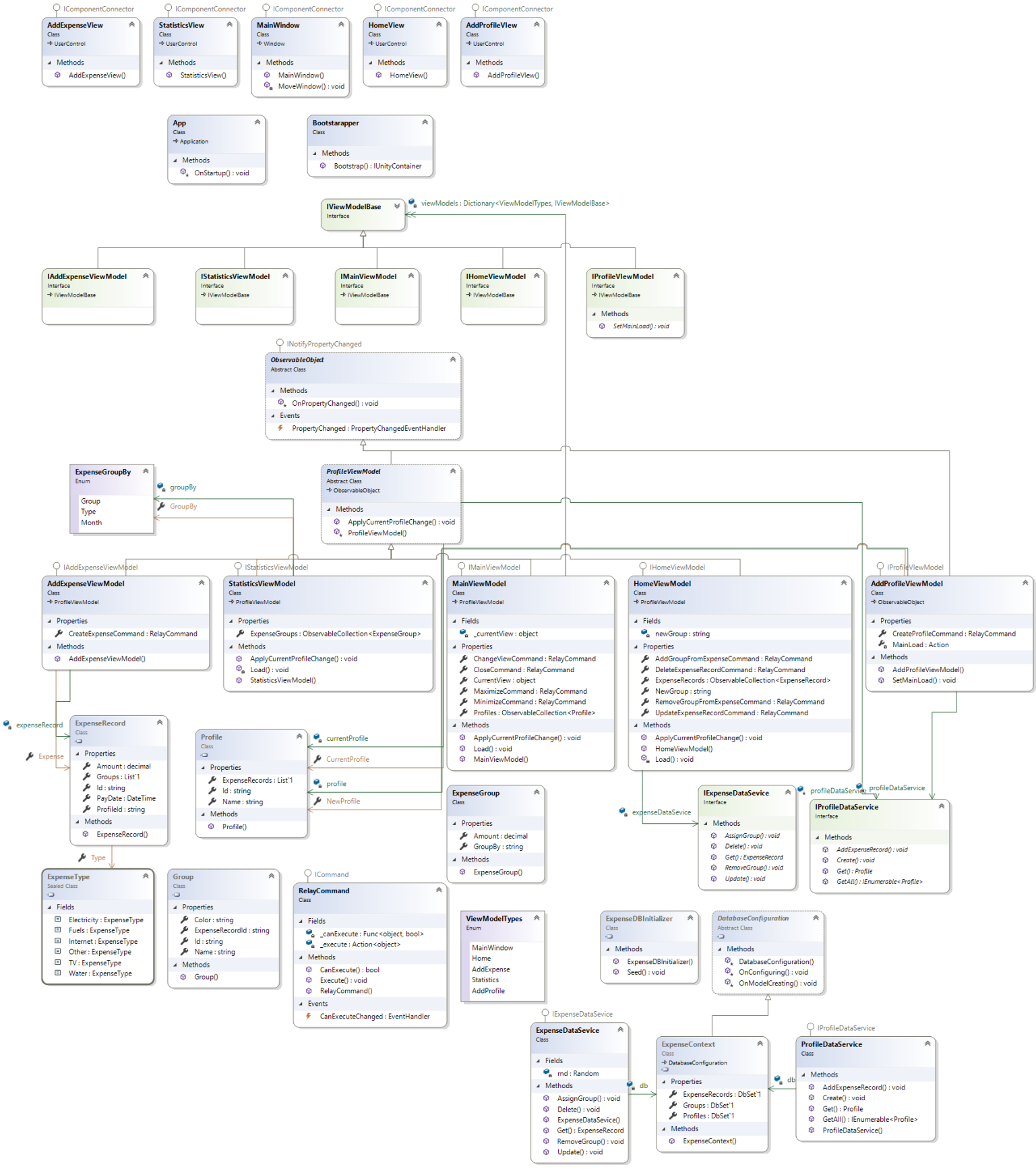
Последният изглед е за добавяне на нови профили, той представлява форма, в която се избира името на новият профил и бутон, с който може да бъде запазен в базата от данни.

4.4 Програмна организация

4.4.1 Структура на приложението

Приложението ще бъде разделено в три проекта. Основният проект ще бъде за потребителският интерфейс (expensit.UI). Другите два проекта ще се грижат за данните. Единият от тях ще е за моделът им (expensit.Moedl), а другият ще съдържа логиката за комуникацията с базата от данни (expensit.DataAccess). Проектът за потребителският интерфейс, ще реферира към другите два проекта, а проекта за комуникацията с базата от данни ще реферира само към проекта с модела, който от своя страна няма да реферира към никой от другите два проекта.

Кодът в проекта за потребителският интерфейс ще следва архитектурата MVVM.



Фигура 2 UML Диаграма на класовете

4.4.2 Използвани технологии

Проектът ще бъде имплементиран на C# с помощта на рамката WPF. Използваните външни библиотеки са:

- DotNetProjects.WpfToolkit.DataVisualization[4] – тази библиотека се използва в проекта за потребителският интерфейс. Тя осигурява готови контроли, които може да се използват в изгледите. От тази библиотека е използвана само една контрола, която представлява диаграма тип пай.
- MaterialDesignThemes[5] – тази библиотека се използва в проекта за потребителският интерфейс. Тя съдържа различни стилове за WPF контролите, които са използвани на повечето места в приложението.
- Unity[6] – тази библиотека се използва в проекта за потребителският интерфейс. Чрез нея се осъществява инжектирането на различните зависимости от обекти.
- Microsoft.EntityFrameworkCore[7] – тази библиотека се използва за комуникацията с базата от данни. Тя представлява ORM, който ни дава приятен интерфейс за работата с данните.
- Microsoft.EntityFrameworkCore.Design[7] – тази библиотека се използва в проекта за потребителският интерфейс.
- Microsoft.EntityFrameworkCore.Proxies[7] – тази библиотека се използва в проекта за комуникация с базата от данни.
- Microsoft.EntityFrameworkCore.Sqlite[7] – тази библиотека се използва в проекта за комуникация с базата от данни.
- Microsoft.EntityFrameworkCore.Tools[7] – тази библиотека се използва в проекта за комуникация с базата от данни.

5 Реализация

Тук ще бъдат разгледани трите реализирани проекта по отделно:

5.1 expensit.Model

Този проект съдържа моделите на данните, използвани от Entity Framework. Това са моделите за профилите, разходите и групите, както и изброим тип съдържаш различните типове разходи. В тези модели, освен информацията за

визуализираните данни на потребителя се пази и идентификатор. Ето и фрагмент описващ данните за разходите:

```
public enum ExpenseType
{
    Electricity,
    Water,
    Internet,
    TV,
    Fuels,
    Other
}

public class ExpenseRecord
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public string Id { get; set; }

    [Required]
    [Column(TypeName = "decimal(19,2)")]
    public decimal Amount { get; set; }

    public ExpenseType Type { get; set; }

    public DateTime PayDate { get; set; }

    public List<Group> Groups { get; } = new List<Group>();

    public virtual string ProfileId { get; set; }

}
```

5.2 expensit.DataAccess

Този проект се грижи за комуникацията с базата. Основният клас, които съдържа е ExpenseContext, чиято роля е да извършва самата комуникация. В него се съдържат DbSet-овете за различните таблици, а връзката с базата се усъществява в неговият родителски клас – DatabaseConfiguration.

```
public class ExpenseContext : DatabaseConfiguration
{
    public DbSet<Profile> Profiles { get; set; }

    public DbSet<ExpenseRecord> ExpenseRecords { get; set; }

    public DbSet<Group> Groups { get; set; }

}

public abstract class DatabaseConfiguration : DbContext
{
    private readonly string baseDir =
System.IO.Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);

    protected override void OnConfiguring(DbContextOptionsBuilder options)
```

```
        {
            options.UseSqlite(@$"Data Source={baseDir}\expenseit.db");
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            ExpenseDBInitializer.Seed(modelBuilder);
        }
    }
```

В класа за извършване на връзката с базата данни, се забелязва, че при създаването на модела към базата се задават начални стойности. За това се грижи класът – ExpenseDBInitialiizer.

Оставащите, класове в този проект, се намират в пакета Migrations. Те се грижат за осъществяване на миграциите на базата от данни при промяна в моделите и/или контекста, които са част от Entity Framework.

5.3 expensit.UI

Това е главният проект на приложението. Той е разделен на множество пакети, всеки от които има своя роля.

5.3.1 Основният пакет

В основният пакет се намират всички разгледани по долу пакети, както и началните файлове на приложението App.xaml и App.xaml.cs.

Във файлът, който седи зад модела се осъществява инжектирането на зависимостите, както и се инициализира основният прозорец, които се показва след това.

```
IUnityContainer container = Bootstarapper.Bootstrap();

MainWindow window = container.Resolve<MainWindow>();
window.Show();
```

В файла с модела се осъществява задаването на ресурсите за използваните теми в приложението, както и осъществява връзката между различните изгледи и изглед моделите.

5.3.2 Core

Този пакет съдържа ключови неща използвани от другите пакети в проекта. По-интересните класове в него са:

- ObservableObject – това е абстрактен клас, чиято роля е да известява изгледите при извикването на метода OnPropertyChanged. Този клас имплементира INotifyPropertyChanged.

```
public abstract class ObservableObject : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged([CallerMemberName] string name = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }
}
```

- IViewModelBase – това е интерфейс имплементиран от всички изгледи. Основното му приложение е при смяната на изгледите.
- ProfileViewModel – този абстрактен клас има ролята да се грижи за избраният профил в общият прозорец. Има приложение в изглед моделите, в които е нужно да се знае кой е избраният профил. За да стигне нужната информация до тях, те само трябва да го наследят.

```
public abstract class ProfileViewModel : ObservableObject
{
    protected readonly IProfileDataService profileDataService;

    private Profile currentProfile;
    public Profile CurrentProfile
    {
        get => currentProfile;
        set
        {
            currentProfile = value != null ? profileDataService.Get(value) : null;
            ApplyCurrentProfileChange();
            OnPropertyChanged();
        }
    }

    public virtual void ApplyCurrentProfileChange() { }

    protected ProfileViewModel(IProfileDataService profileDataService)
    {
        this.profileDataService = profileDataService;
    }
}
```

- RelayCommand – това е клас, който представлява команда с един параметър. Тя приема две ламбда функции. Първата е тази която е нещото

което ще се изпълнява от тази команда, а втората е тази която ще проверява дали може да се изпълни командата.

```
    public class RelayCommand : ICommand
    {
        private readonly Action<object> _execute;
        private readonly Func<object, bool> _canExecute;

        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }

        public RelayCommand(Action<object> execute, Func<object, bool> canExecute =
null)
        {
            _execute = execute;
            _canExecute = canExecute;
        }

        public bool CanExecute(object parameter)
        {
            return _canExecute == null || _canExecute(parameter);
        }

        public void Execute(object parameter)
        {
            _execute(parameter);
        }
    }
```

- Bootstarapper – това е клас, който съдържа статична функция, чиято роля е да инициализира контейнера използван за инжектиране на зависимостите. Класовете в този контейнер, които се грижат за работата с базата от данни следват шаблона Singleton.

```
        public static IUnityContainer Bootstrap()
        {
            IUnityContainer container = new UnityContainer();

            container.RegisterType<ExpenseContext>(TypeLifetime.Singleton);
            container.RegisterType<IExpenseDataSevice,
ExpenseDataSevice>(TypeLifetime.Singleton);
            container.RegisterType<IProfileDataService,
ProfileDataService>(TypeLifetime.Singleton);

            container.RegisterType<IMainViewModel, MainViewModel>();
            container.RegisterType<IHomeViewModel, HomeViewModel>();
            container.RegisterType<IAddExpenseViewModel, AddExpenseViewModel>();
            container.RegisterType<IStatisticsViewModel, StatisticsViewModel>();
            container.RegisterType<IProfileVIewModel, AddProfileViewModel>();

            return container;

        }
```

5.3.3 Data

Този пакет съдържа сървисите, използвани за записване и четене от базата данни. Дефинираните в него сървиси са:

- ExpenseDataService – ролята на този сървис е да се грижи за работата с един разход. Действията, които осигурява са вземане, обновяване, изтриване, добавяне на групи и премахване на групи от разход. При добавянето на група, ще бъде избран цвят с помощта на класа Random.
- ProfileDataService – ролята на този сървис е да се грижи за работата с един профил. Действията, които осигурява са вземане на всички профили, създаване, добавяне на разход и вземане на един профил. При вземане на всички профили не се добавят разходите, към връщаният резултат, ако искаме да вземем и разходите, тогава трябва да използваме метода за вземане на един профил, в който вече ще има и разходите му.

```
        public IEnumerable<Profile> GetAll()
        {
            return db.Profiles.AsEnumerable();
        }

        public Profile Get(Profile profile)
        {
            return db.Profiles.Include(p => p.ExpenseRecords).ThenInclude(er =>
er.Groups).Where(p => p == profile).First();
        }
```

5.3.4 Theme

В този пакет се намират теми разработени с цел използването им в различните изгледи. Там може да се види файлът MenuButtonTheme.xaml, който съдържа темата използвана за осъществяване на бутоните в менюто за смяна на изгледите.

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Style BasedOn="{StaticResource {x:Type ToggleButton}}" TargetType="{x:Type
RadioButton}" x:Key="MenuButtonTheme">
        <Style.Setters>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="RadioButton">
                        <Grid VerticalAlignment="Stretch"
HorizontalAlignment="Stretch"
                              Background="{TemplateBinding Background}">
```

```
                            <TextBlock Text="{TemplateBinding Property=Content}"
VerticalAlignment="Center" Margin="50,0,0,0"/>
                        </Grid>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>

            <Setter Property="Background" Value="Transparent"/>
            <Setter Property="BorderThickness" Value="0"/>
        </Style.Setters>

        <Style.Triggers>
            <Trigger Property="IsChecked" Value="True">
                <Setter Property="Background" Value="#22202f"/>
            </Trigger>
        </Style.Triggers>
    </Style>

</ResourceDictionary>
```

5.3.5 View

В този пакет се намират изгледите на приложението. Те са:

- MainWindow – в този изглед се намира решетка, която разделя прозореца на четири части:
  - Името на приложението.
  - Лента с бутоните за управление на приложението – към тази лента има събитие, чиято роля е да позволява преместването на приложението с мишката.

```
        private void MoveWindow(object sender,
System.Windows.Input.MouseButtonEventArgs e)
        {
            DragMove();
        }
```

  - Меню с бутони за смяна на изгледа – всеки от бутоните съдържа команда, чиято роля е смяната на изгледа, параметърът, който приема тази команда е типа на новият изглед.
  - Място в което се зарежда избраният от бутоните изглед – осъществява се чрез ContentControl.

```
<ContentControl Grid.Column="1" Content="{Binding CurrentView}" Grid.Row="2"/>
```

- HomeView – за визуализирането на разходите, се използва DataGrid. Групите за всеки един от разходите, също се намират в DataGrid. В

началото на този изглед има ресурс, чиято роля е да вземе възможните типове за разходите:

```
    <UserControl.Resources>
        <ObjectDataProvider x:Key="types" MethodName="GetValues" ObjectType="{x:Type
System:Enum}">
            <ObjectDataProvider.MethodParameters>
                <x:Type TypeName="Models:ExpenseType"/>
            </ObjectDataProvider.MethodParameters>
        </ObjectDataProvider>

    </UserControl.Resources>
```

Binding-ите във вложеният DataGrid за групите съдържат и RelativeSource, чрез който се избира главният контекст, в който се намира и DataContext-а.

```
Command="{Binding DataContext.AddGroupFromExpenseCommand,
RelativeSource={RelativeSource AncestorType=UserControl, AncestorLevel=1}}"
                                        CommandParameter="{Binding Id}"
```

- AddExpenseView – този изглед съдържа Card компонента, който идва от MaterialDesign.
- StatisticsView – този изглед съдържа Card компонента, който идва от MaterialDesign. Статистиките се визуализират в PieSeries компонент, който идва от ChartingTools.
- AddProfileView – този изглед съдържа Card компонента, който идва от MaterialDesign.

5.3.6 ViewModel

В този пакет се намират изглед моделите съветващи на изгледите. Всеки от тях има и съответветсваш интерфейс, който се използва за инжектирането на зависимостите. Те следват MVVM архитектурата. Съответно това са:

- MainViewModel – в този модел на изглед се съдържа логиката за преминаване от изглед в изглед. За целта имаме поле, което пази текущият изглед, речник с всички възможни изгледи и команда, чиято роля е да осъществи смяната. Интересното за смяната е че се извиква функцията идваща от абстактният клас ProfieViewModel ApplyCurrentProfileChange, чиято роляе да зададе текущият профил на всички изглед модели.

```
        private object _currentView;
        public object CurrentView
        {
```

```
            get => _currentView;
            set
            {
                _currentView = value;
                OnPropertyChanged();
            }
        }

        private readonly Dictionary<ViewModelTypes, IViewModelBase> viewModels =
new();


        public RelayCommand ChangeViewCommand { get; }

            ChangeViewCommand = new RelayCommand(NextView =>
            {
                CurrentView = viewModels.GetValueOrDefault((ViewModelTypes)NextView);

                if (CurrentView is ProfileViewModel)
                {
                    (CurrentView as ProfileViewModel).ApplyCurrentProfileChange();
                }

            });

        public override void ApplyCurrentProfileChange()
        {
            viewModels.Values.ToList().Select(vm => vm as ProfileViewModel).Where(vm
=> vm != null).ToList().ForEach(vm => vm.CurrentProfile = CurrentProfile);

        }
```

- AddExpenseViewModel – този модел на изглед наследява ProfileViewModel и се грижи за добавянето на разходи към профила чрез сървиса за профилите.
- HomeViewModel – този модел на изглед наследява ProfileViewModel и се грижи за визуализирането и модифицирането на разходите в табличен вид.
- StatisticsViewModel HomeViewModel – този модел на изглед наследява ProfileViewModel и се грижи за визуализирането на разнообразни статистики за разходите. Това става с помощта на поле GroupBy от изброим тип ExpenseGroupBy. При настъпила промяна в него се извършва ново групиране с помощта на Linq.

```
        private ExpenseGroupBy groupBy;
        public ExpenseGroupBy GroupBy
        {
            get => groupBy;
            set
            {
                groupBy = value;
                OnPropertyChanged();

                if (CurrentProfile == null)
```

```
                {
                    return;
                }

                switch (groupBy)
                {
                    case ExpenseGroupBy.Group:
                        ExpenseGroups = new
ObservableCollection<ExpenseGroup>(CurrentProfile.ExpenseRecords.Select(er => (
                            er.Amount,
                            Groups: string.Join("; ", er.Groups.OrderBy(g =>
g.Name).Select(g => g.Name))
                        )).GroupBy(
                            er => er.Groups,
                            er => er.Amount,
                            (key, group) => new ExpenseGroup(
                                key,
                                group.Sum(er => er)
                        )).ToList());
                        break;
                    case ExpenseGroupBy.Type:
                        ExpenseGroups = new
ObservableCollection<ExpenseGroup>(CurrentProfile.ExpenseRecords.GroupBy(
                            er => er.Type,
                            er => er.Amount,
                            (key, group) => new ExpenseGroup(
                                key.ToString(),
                                group.Sum(er => er)
                        )).ToList());
                        break;
                    case ExpenseGroupBy.Month:
                        ExpenseGroups = new
ObservableCollection<ExpenseGroup>(CurrentProfile.ExpenseRecords.GroupBy(
                            er => er.PayDate.Month.ToString() + "/" +
er.PayDate.Year.ToString(),
                            er => er.Amount,
                            (key, group) => new ExpenseGroup(
                                key.ToString(),
                                group.Sum(er => er)
                        )).ToList());
                        break;
                    default:
                        break;
                }

                OnPropertyChanged(nameof(ExpenseGroups));
            }

        }
```

- AddProfileViewModel – този модел на изглед се грижи за добавянето на нови профили. Интересното в него е че приема Action, чиято роля е да задейства MainWindow-а да презареди данните за профилите си.

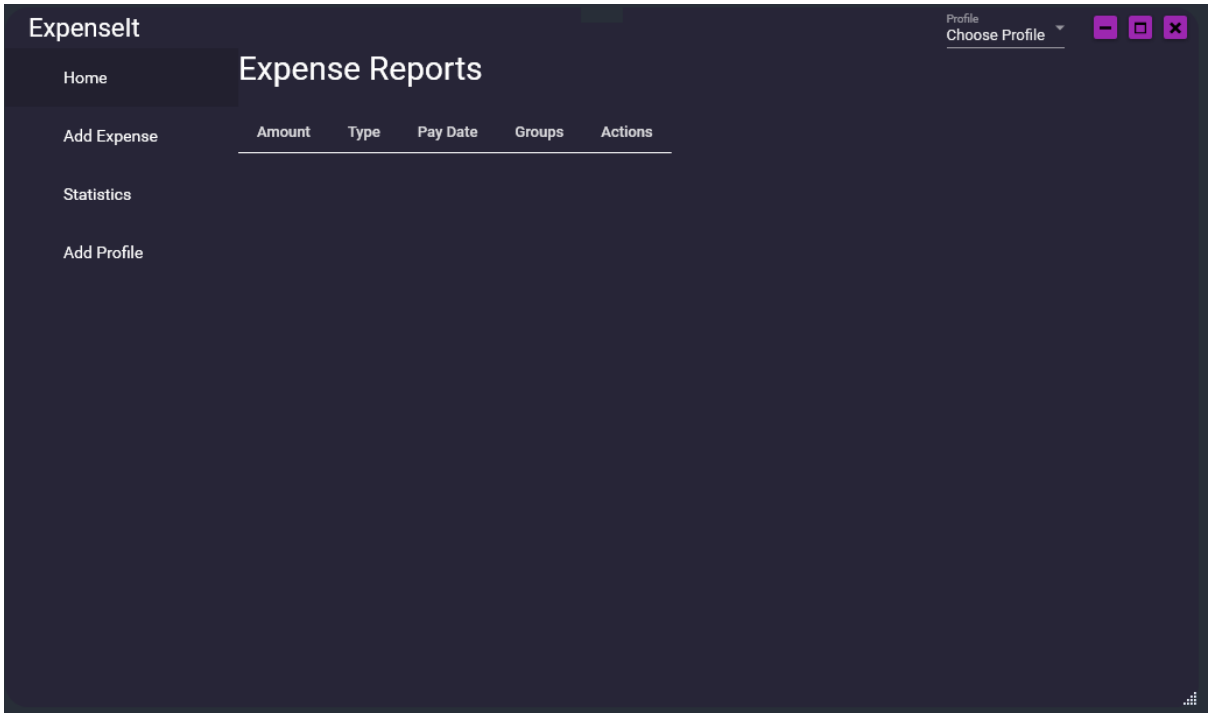
```
            CreateProfileCommand = new RelayCommand(o =>
            {
                this.profileDataService.Create(NewProfile);
                MainLoad.Invoke();
```

```
            NewProfile = new Profile();
        });
```

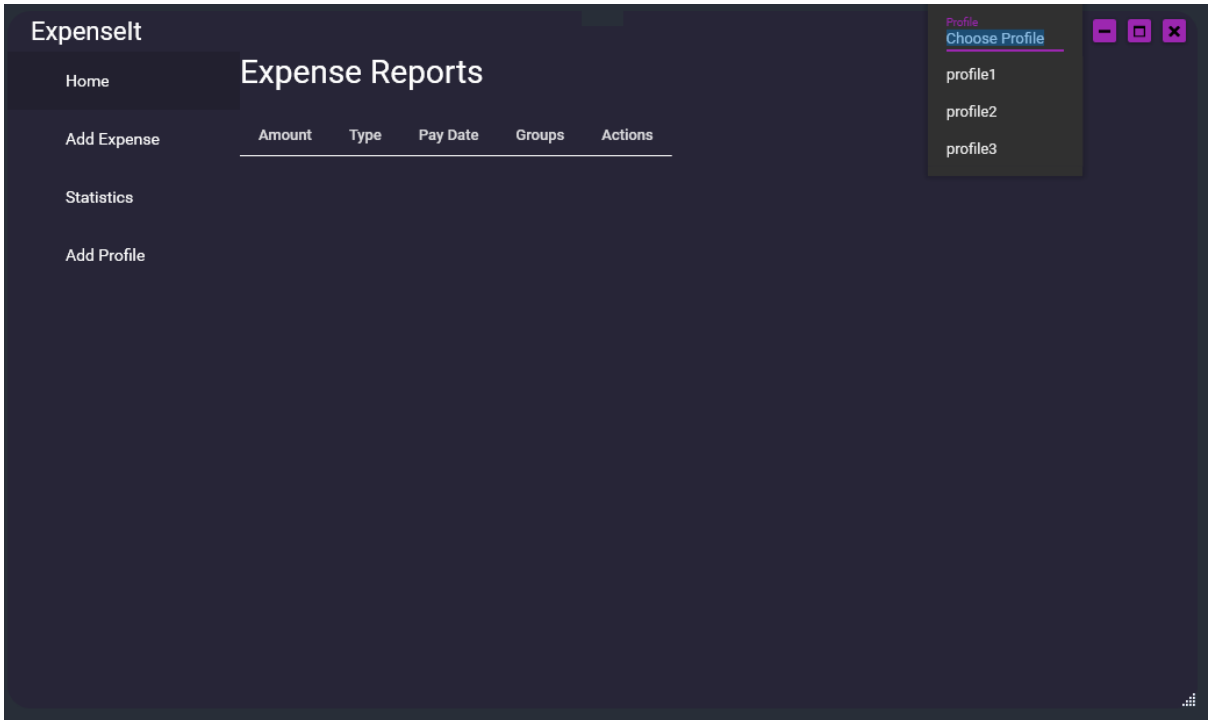
6 Потребителски ръководство

6.1 Общ прозорец

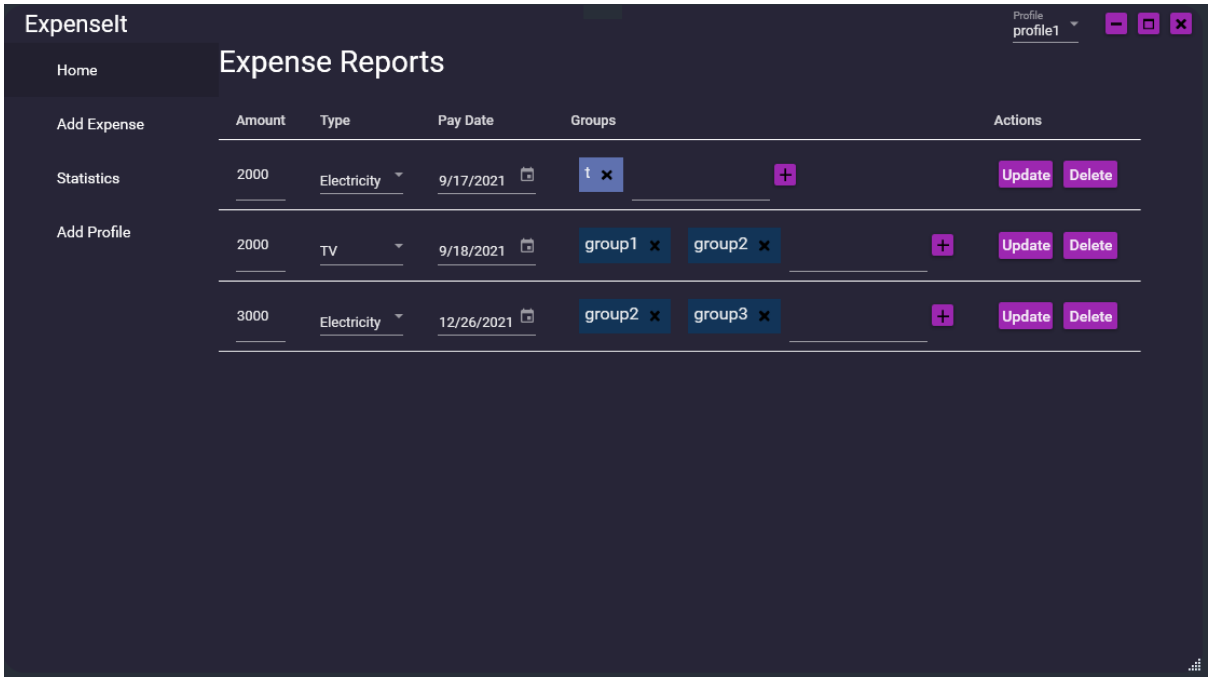
След отваряне на програмата се показва този прозорец, чиито ъгли за закръглени. Нещата, които се съхраняват в него са горната лента с опции, лявото меню за избор на изглед, десният ъгъл, позволяваш промяна на размера на прозореца. Оставащото място в него е за избрания изглед от менюто. В лентата с опции се намират бутони за контрол на прозореца (минимизиране, максимизиране, затваряне и преместване по екрана), както и меню за избор на профил. При избор на профил, ако в него има данни за разходи, те ще се появят в таблицата, която ги съдържа.



Фигура 3 Прозореца на приложението след неговото отваряне



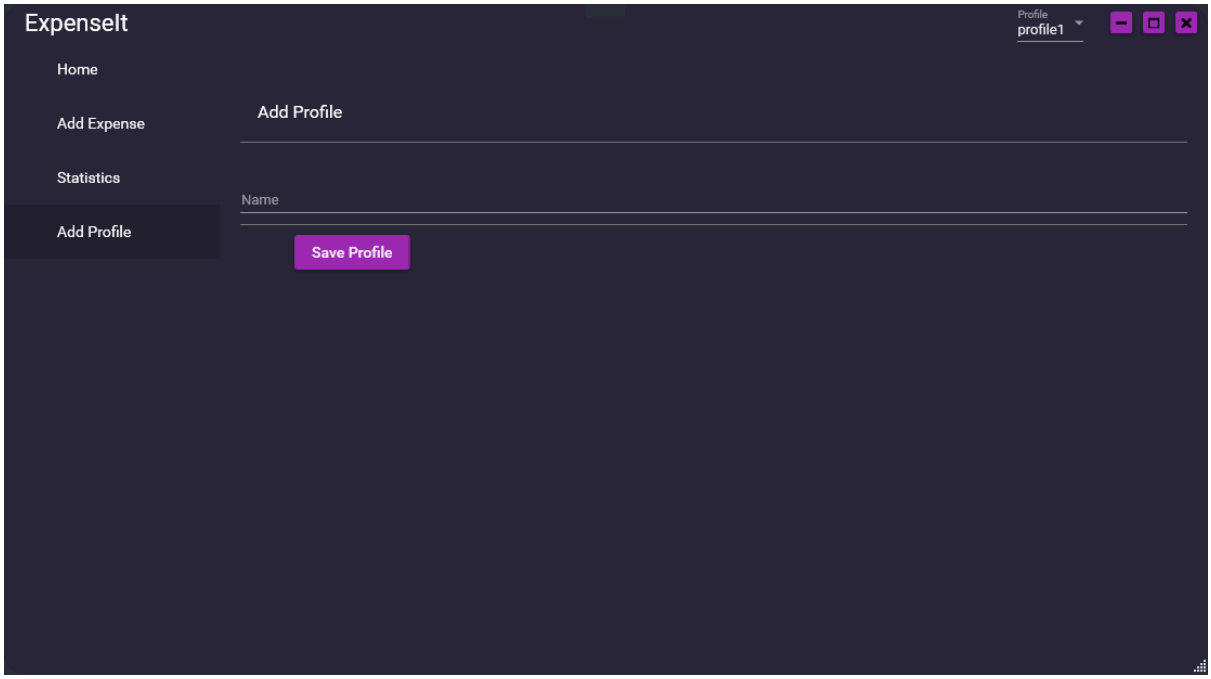
Фигура 4 Меню за избиране на профил



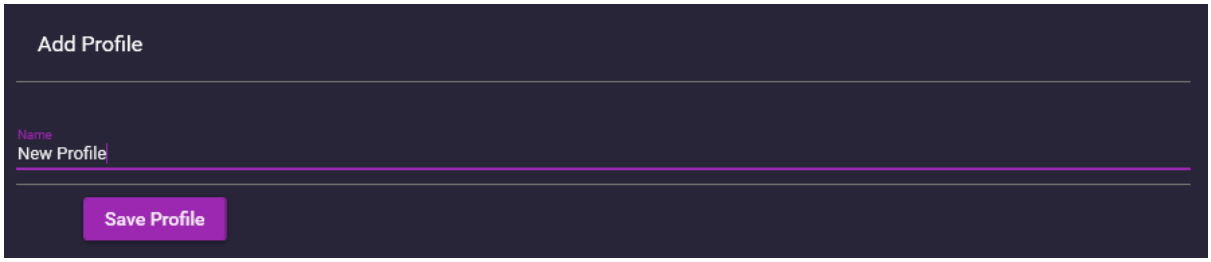
Фигура 5 Прозореца след избиране на профил

6.2 Изглед за добавяне на профил

Изгледа за добавяне на профил представлява форма в която може да се избере името на новият профил.



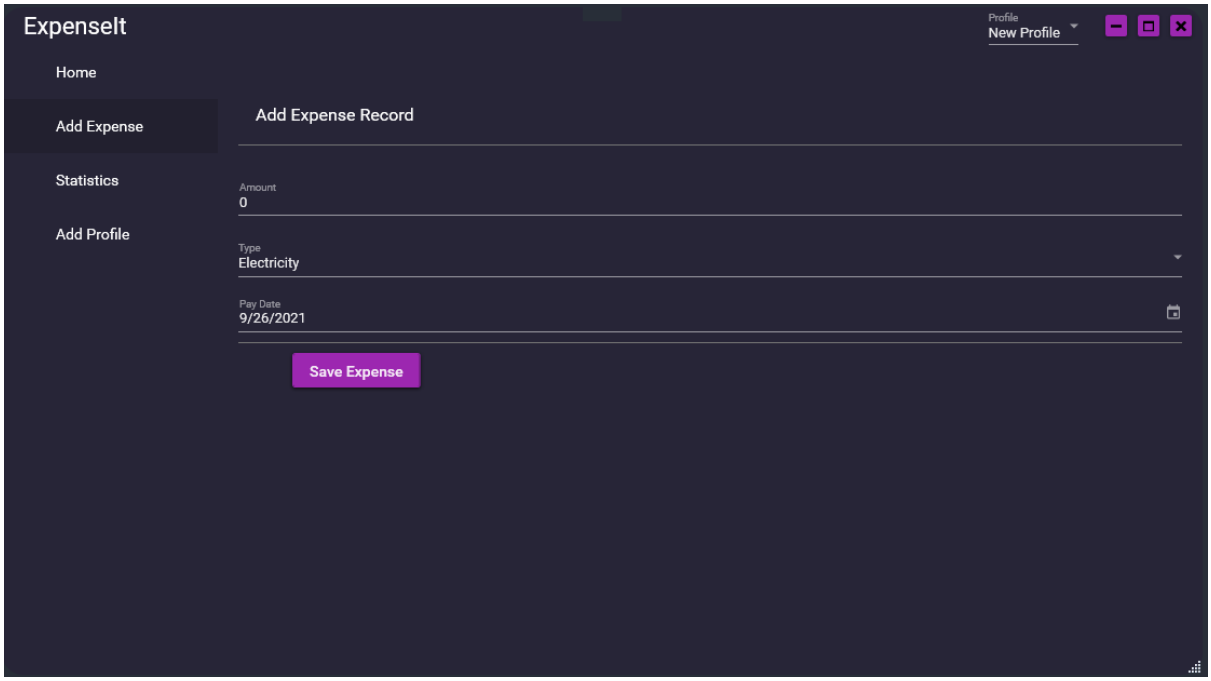
Фигура 6 Изглед за избиране на профил



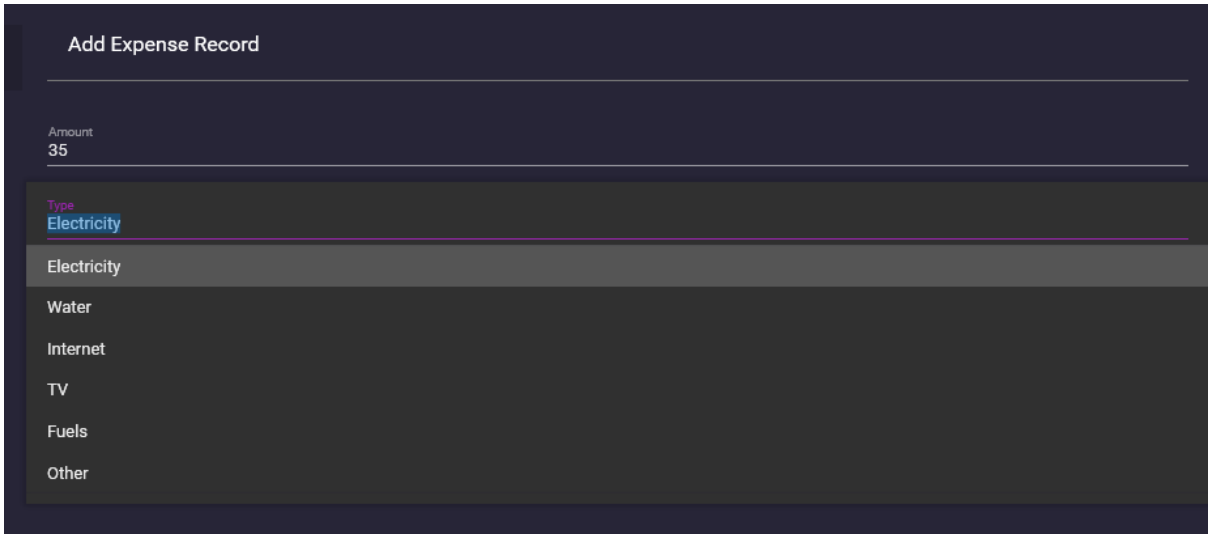
Фигура 7 Изглед за избиране на профил след въвеждане на необходимите за него данни

6.3 Изглед за добавяне на разходи

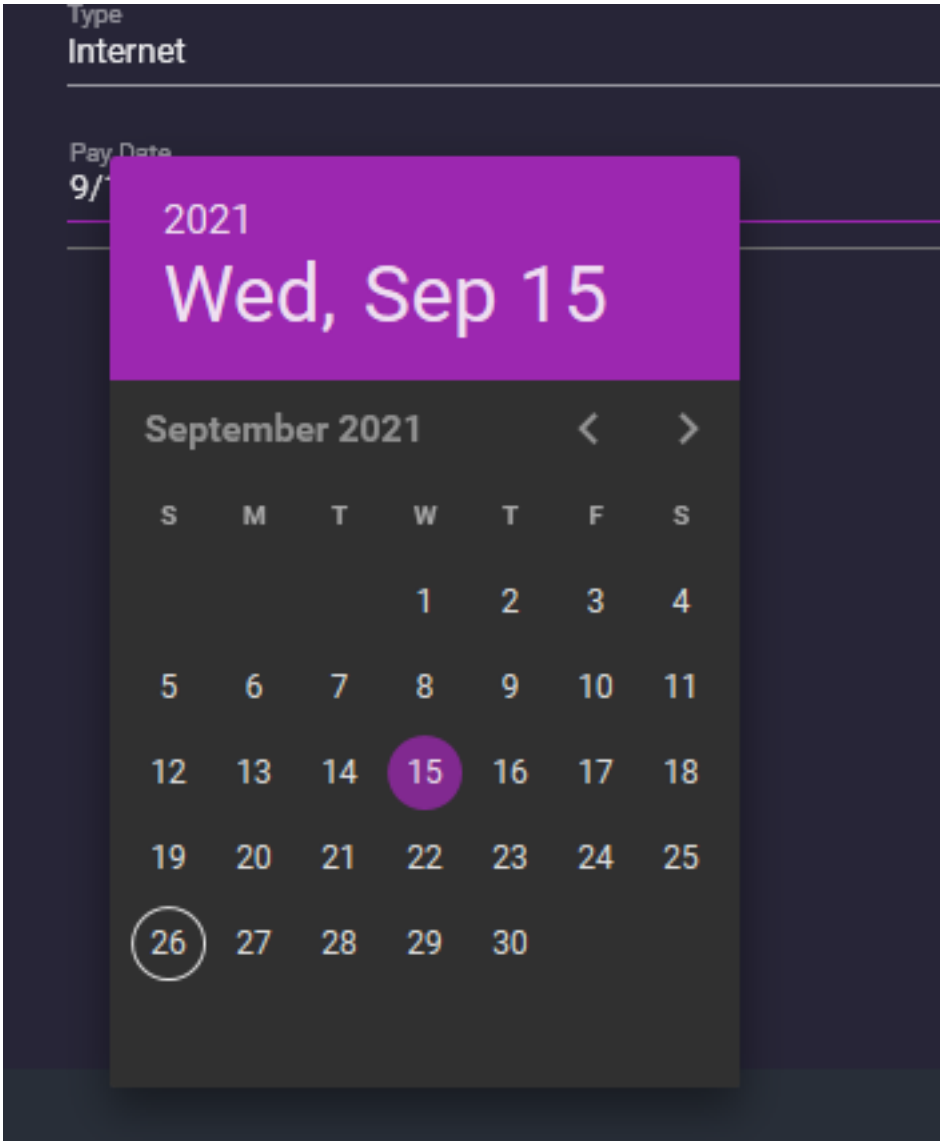
Изгледа за добавяне на разход съдържа форма, в която може да се въведе необходимата информация за него. Това са неговата големина, тип и дата на извършване на транзакцията.



Фигура 8 Изглед за добавяне на разход



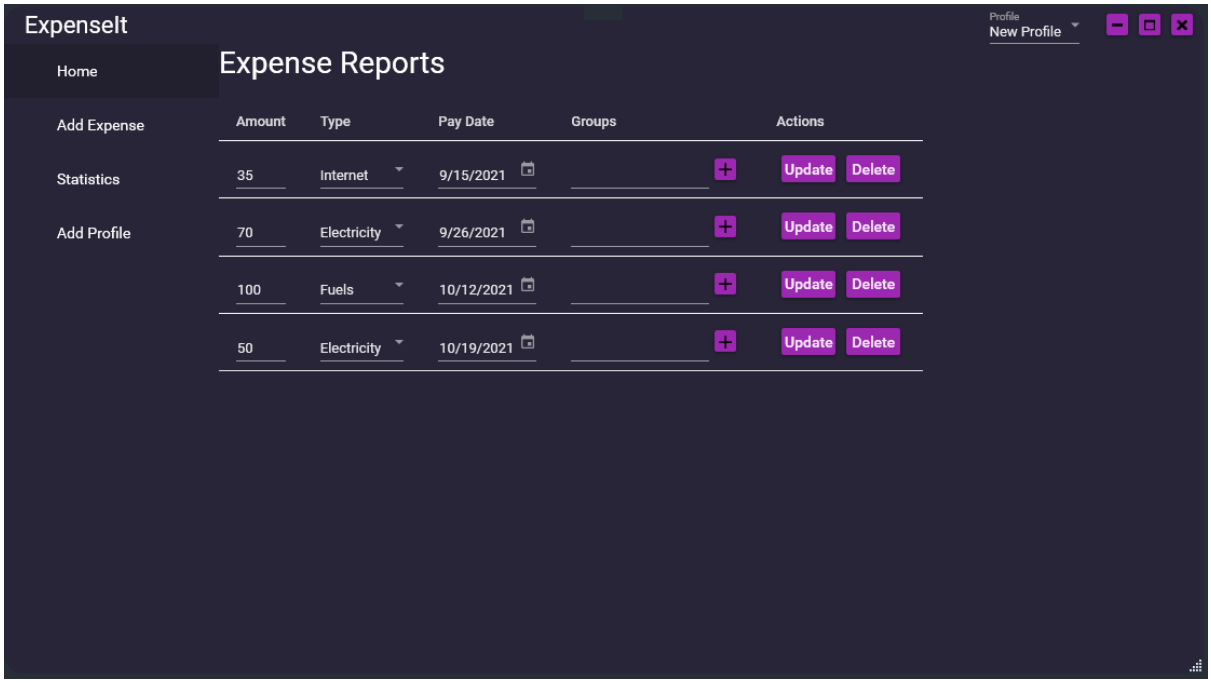
Фигура 9 Меню за избиране на тип на разхода



Фигура 10 Меню за избиране на датата на извършване на разхода

6.4 Изглед за разглеждане на разходите

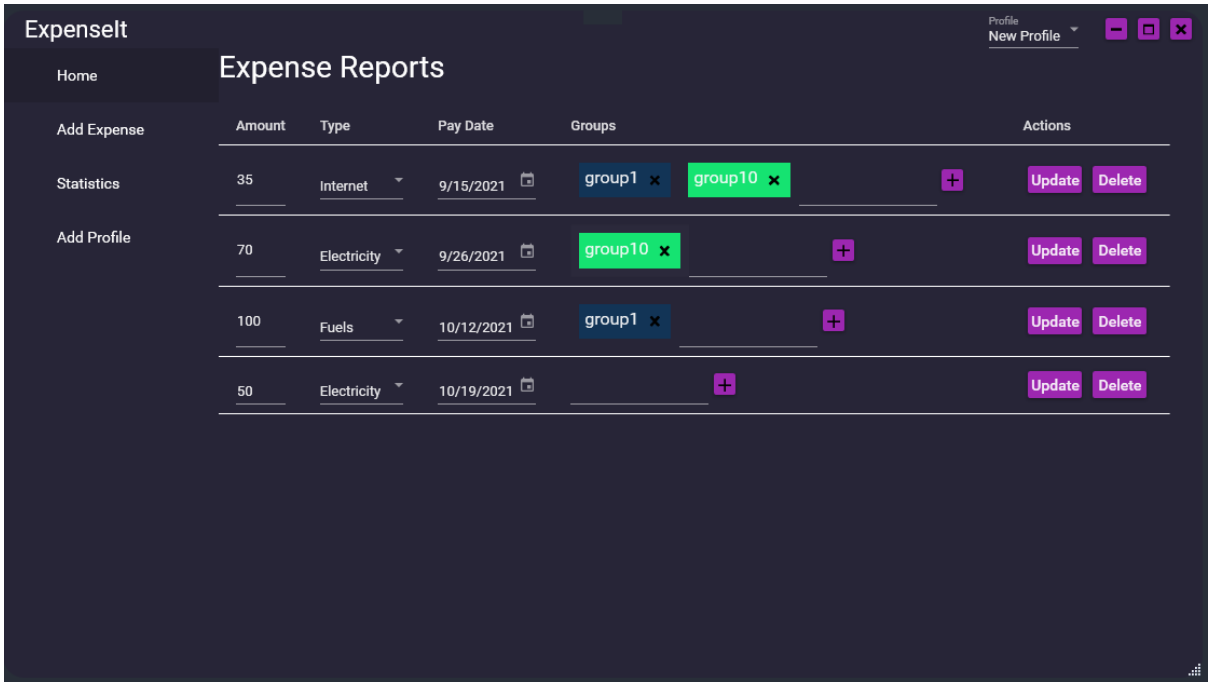
Изгледа за разглеждане на разходи съдържа таблица в която може да се наблюдават и редактират различните разходи.



Фигура 11 Изглед за разглеждане на разходите

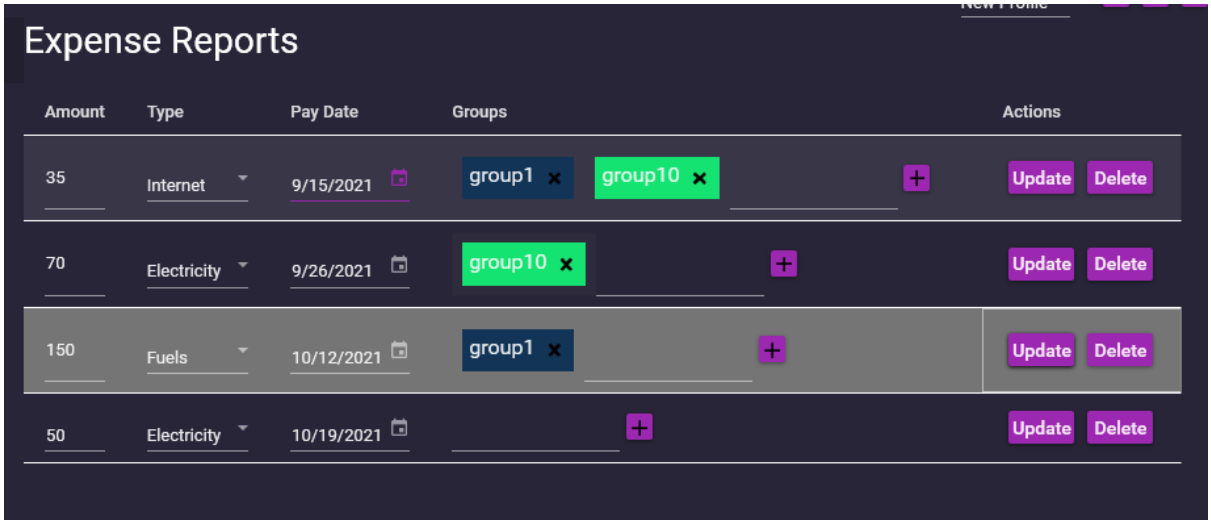
В тази таблица може да се добавят и премахват групи чрез съответните бутони.

Фигура 12 Добавяне на групи за разходите



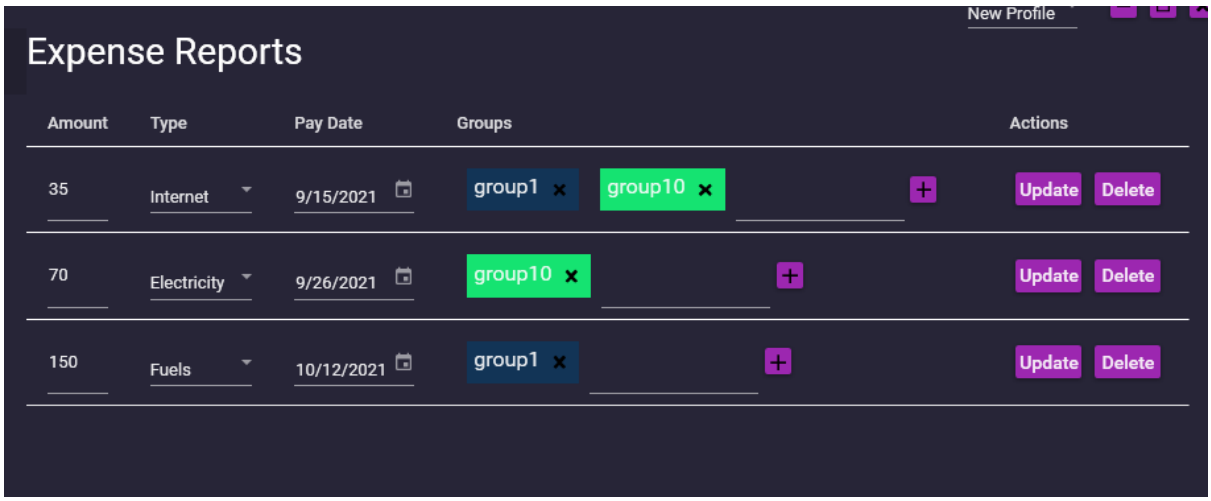
Фигура 13 Премахване на групи от разходите

Редактирането на данните за разхода става като се промени стойността им в таблицата и след това се натисне бутона за обновяване, за да бъдат отразени промените в базата от данни.



Фигура 14 Редактиране на разход

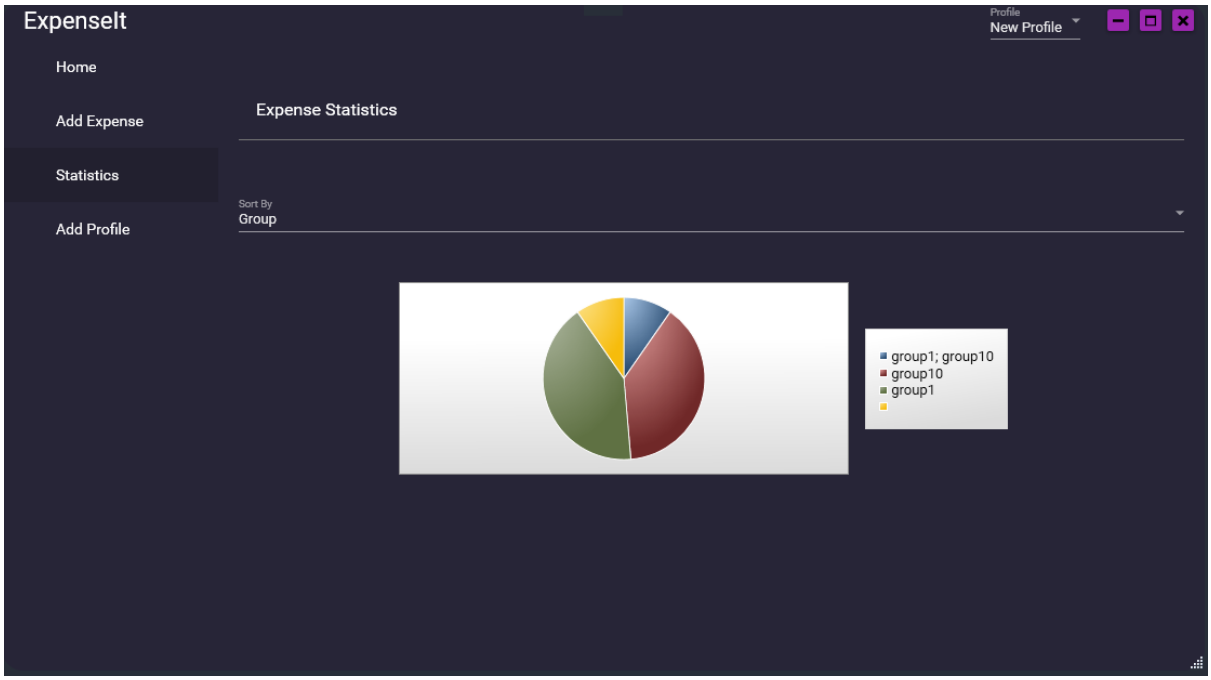
Изтриването на разходи става, чрез бутона за изтриване.



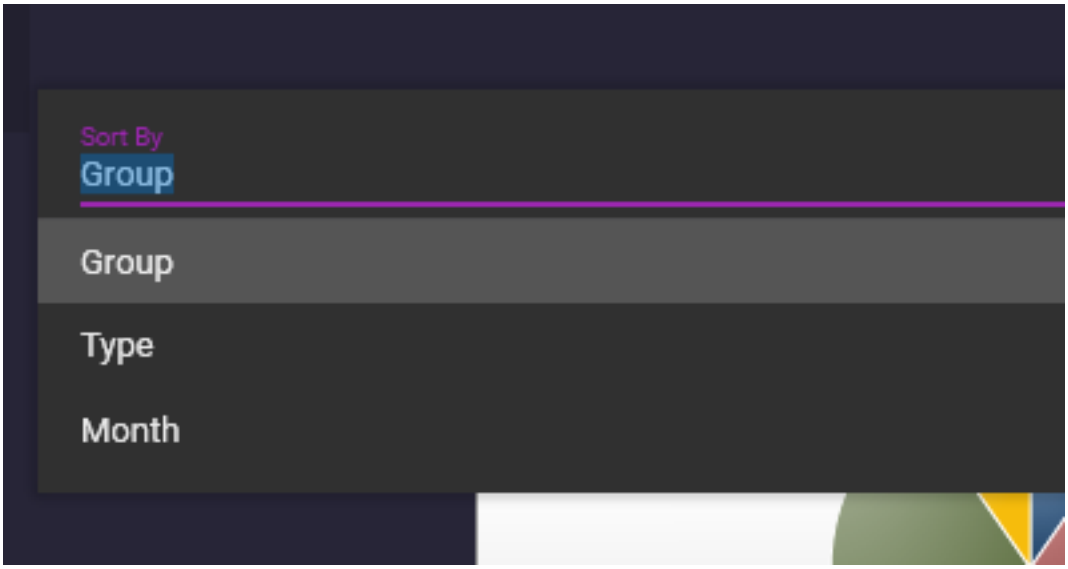
Фигура 15 Изтриване на разход

6.5 Изглед за изготвяне на статистики

Изгледа за изготвяне на статистики, съдържа меню за избор на стойността, по която ще бъдат групирани разходите. Възможните опции са по група, тип и по месец на извършване на разхода. Предварително избраната опция е групиране по групи.

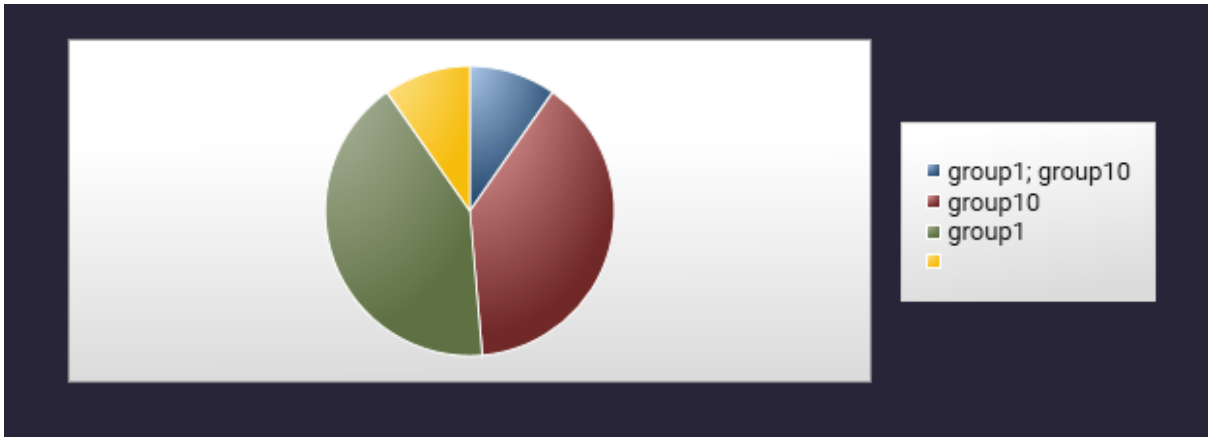


Фигура 16 Изглед за изготвяне на статистики



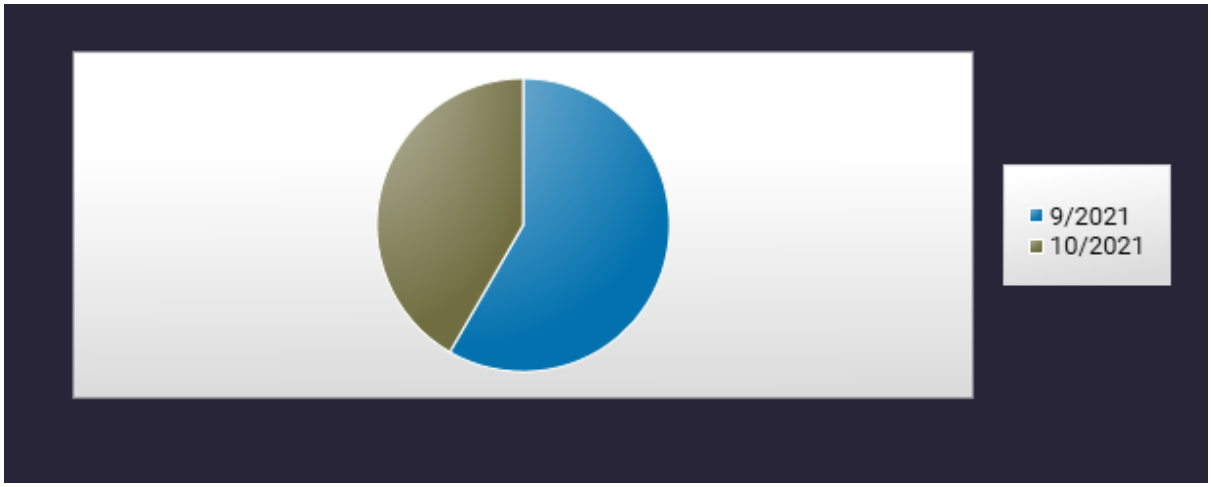
Фигура 17 Опции за групиране на разходите

Фигура 18 Данни, с които са изготвени долните графики



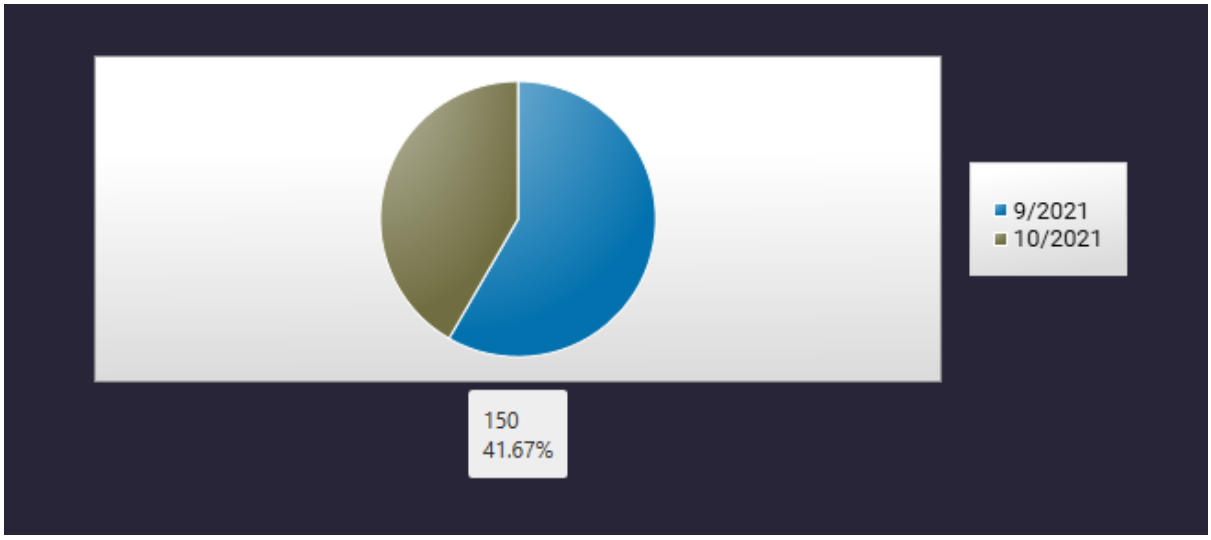
Фигура 19 Диаграма изготвена с групиране по групи

Фигура 20 Диаграма изготвена с групиране по тип



Фигура 21 Диаграма изготвена с групиране по месец на извършване

При минаване с мишката през някоя от областите на диаграмата може да се види, какъв е размерът на разходите в тази група, както и какъв процент от всички разходи заемат.



Фигура 22 Проверка на размерът на разходите в групата

7 Заключение

Разработеното приложение, отговаря на всички изисквания. Чрез него може да бъдат записвани различни разходи, които да бъдат разделяни на база категории и групи. Също така с него могат да се изготвят и статистики на база зададени параметри. Спрямо другите съществуващи решения, има функционалности, които липсват, но има и такива, които в тях ги няма.

8 Литература

[1] https://www.microsoft.com/en-us/p/spending-tracker/9wzdncrfhmw8?activetab=pivot:overviewtab

[2] https://www.microsoft.com/en-us/p/moneybook/9nblggh3z8xq?activetab=pivot:overviewtab

[3] https://www.microsoft.com/en-us/p/money-fox/9nblggh6ck9d?activetab=pivot:overviewtab

[4] https://github.com/dotnetprojects/WpfToolkit

[5] https://github.com/MaterialDesignInXAML/MaterialDesignInXamlToolkit

[6] https://github.com/unitycontainer/unity

[7] https://docs.microsoft.com/en-us/ef/core

9 Приложение – анализ на кода

Реализацията на приложението, както тази документация и графиките от нея могат да бъдат намерени - https://github.com/nikolaystanishev/expenseit.