ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

Факултет „Компютърни системи и технологии”

ППРАКТИЧЕСКИ ПРОЕКТ
ПО СОФТУЕРНИ ПЛАТФОРМИ

НА ТЕМА
„Списъци от опции“

Студент: Никита Радославов Койнов
ФАК. № 381222002 **Група:** 91

Съдържание

Въведение

Системата представлява десктоп приложение, разработено с помощта на WPF и .NET. Основната идея е да се предостави интуитивен и лесен за използване интерфейс, чрез който потребителите могат да въвеждат, редактират и съхраняват информация за себе си, като избират предпочитани дейности, технологии и други характеристики. Приложението е създадено с мисъл за разширяемост и лесна поддръжка, като използва утвърдени архитектурни принципи и шаблони за проектиране.

Една от ключовите цели на системата е да осигури гъвкавост при работа със списъци от стойности – част от тях са предварително дефинирани (например пол или програмни езици), докато други могат да бъдат добавяни динамично от самите потребители (като нови дейности или технологии). Това позволява на приложението да се адаптира към различни сценарии и нужди, без да изисква промяна в кода или повторно компилиране. Освен това, системата се стреми да гарантира целостта и валидността на данните чрез проверки за уникалност и валидация на входа, което е особено важно при работа с потребителски генерирани списъци.

В основата си приложението е предназначено за учебни и демонстрационни цели, като показва как може да се реализира модерен потребителски интерфейс, интегриран с база данни, и как се прилагат принципите на MVVM архитектурата. Въпреки това, структурата и подходът позволяват лесно надграждане и използване в по-реални проекти, където е необходимо управление на потребителски профили и свързани с тях данни.

Описание и цели на системата

Основна цел и функционалности

Основната цел на разработената система е да предостави удобен и структуриран начин за създаване и управление на потребителски профили, като се акцентира върху персонализацията и лесното разширяване на възможностите за избор. Приложението позволява на всеки потребител да въведе свои лични данни и предпочитания, като например пол, любима дейност, предпочитана технология и програмен език. Този подход не само улеснява събирането и съхранението на информация, но и създава усещане за индивидуалност и ангажираност у потребителя.

Функционалностите на системата са реализирани така, че да обхващат целия жизнен цикъл на данните – от въвеждането им през визуализацията до тяхното съхранение и последваща обработка. Потребителят може да създава нови профили, да редактира вече съществуващи, както и да избира случаен профил, което е полезно при демонстрации или тестове. Особено внимание е отделено на работата със списъци – системата позволява добавяне на нови дейности и технологии директно от интерфейса, без необходимост от намеса на администратор или промяна в кода. Това се реализира чрез специални диалогови прозорци, които валидират въведената информация и гарантират, че няма дублиране на стойности.

Освен основните операции по създаване и редакция, приложението осигурява и механизми за валидиране на данните, като например проверка за уникалност на имената на дейности и технологии. По този начин се предотвратяват грешки и се поддържа високо качество на информацията в системата. Всички тези функционалности са интегрирани в

единен и последователен потребителски интерфейс, който следва добрите практики за използваемост и достъпност.

Типове списъци (системни, потребителски, смесени)

В основата на приложението стои концепцията за работа с различни типове списъци, които отразяват разнообразието от възможни избори, достъпни за потребителя. Системата ясно разграничава между системни, потребителски и смесени списъци, като всеки от тях има своя роля и начин на управление.

Системните списъци съдържат предварително дефинирани стойности, които са част от бизнес логиката на приложението и не подлежат на промяна от страна на крайния потребител. Пример за такъв списък е изборът на пол или програмен език, където възможните стойности са описани чрез енъми в кода. Този подход гарантира консистентост и предотвратява въвеждането на невалидни или нежелани стойности.

Потребителските списъци, от своя страна, са динамични и позволяват на всеки потребител да добавя нови елементи според своите нужди и предпочитания. Такива са списъците с дейности и технологии, които могат да бъдат разширявани по всяко време чрез специални диалогови прозорци. Това дава възможност системата да се адаптира към различни сценарии на употреба и да отразява реалните интереси на потребителите, без да се налага промяна в програмния код.

Смесените списъци представляват комбинация от системни и потребителски стойности. Например, при избора на предпочитана технология, потребителят може да избере както от предварително зададени програмни езици, така и от технологии, които сам е добавил. Този модел осигурява максимална гъвкавост и удобство, като съчетава стабилността на системните стойности с динамиката на потребителските.

Така реализираната система за списъци позволява едновременно да се поддържа контрол върху критичните за приложението стойности и да се предоставя свобода на потребителя да персонализира своя опит според индивидуалните си нужди.

Изисквания и ограничения

За да функционира коректно, системата изисква наличието на .NET среда и операционна система, която поддържа WPF приложения (Windows). Данните се съхраняват в локална база данни SQLite, което улеснява разгръщането и използването на приложението без необходимост от сложна конфигурация или външни сървъри. При първото стартиране на приложението базата данни се създава автоматично, а всички необходими таблици и връзки се инициализират чрез Entity Framework Core.

Едно от основните ограничения на системата е свързано с уникалността на имената за дейности и технологии. При добавяне на нови елементи в тези списъци се извършва проверка дали вече съществува такъв запис, като при опит за дублиране потребителят получава съобщение за грешка. Това гарантира, че данните остават чисти и последователни, а работата със списъците е предвидима и надеждна.

Ограниченията на приложението са съобразени с неговата учебна и демонстрационна цел. Например, не се поддържа работа с множество потребители едновременно или синхронизация с външни източници на данни. Въпреки това, архитектурата е изградена така, че да позволява лесно разширяване и интеграция на нови функционалности при необходимост. По този начин системата може да бъде използвана като основа за по-сложни

решения, които изискват по-високо ниво на сигурност, мащабируемост или интеграция с други платформи.

Структура на кода и MVVM архитектура

Организация по MVVM

Организацията на кода в проекта стриктно следва принципите на MVVM архитектурата, като всеки слой е ясно отделен и изпълнява специфични задачи. Това разделение не само улеснява разбирането и поддръжката на системата, но и позволява лесно разширяване и модифициране на отделни компоненти без риск от неочаквани странични ефекти.

Моделите (Model) са разположени в отделна папка и съдържат класове, които описват основните бизнес обекти и техните свойства. Например, класът User енкапсулира информация за потребителя, включително предпочитания за дейност, технология и програмен език. Класовете Activity и Technology описват съответно дейностите и технологиите, които могат да бъдат избирани от потребителите. Всички тези модели са проектирани така, че да бъдат лесно интегрирани с Entity Framework Core, което позволява автоматично създаване и управление на базата данни.

ViewModel-ите са разположени в собствена папка и служат като посредник между моделите и визуалните компоненти. Те съдържат логиката за обработка на данните, управление на състоянието и реакция на действията на потребителя. Например, UserFormViewModel отговаря за зареждането на списъците с възможни избори, обработката на командите за добавяне на нови елементи и запис на данните в базата. Всеки ViewModel наследява BaseViewModel, който предоставя механизъм за известяване при промяна на свойствата, така че интерфейсът да се обновява автоматично при промяна на данните.

Визуалните компоненти (View) са реализирани чрез XAML файлове, които описват подредбата и стилизирането на елементите в потребителския интерфейс. Те са свързани с ViewModel-ите чрез data binding, което позволява двупосочна комуникация между логиката и визуализацията. Например, MainWindow.xaml съдържа форма за въвеждане и избор на данни, като всички контроли са обвързани с подходящи свойства и команди от UserFormViewModel. Това позволява на потребителя да взаимодейства с приложението по интуитивен начин, без да се налага ръчно обновяване на интерфейса или обработка на събития.

Тази организация на кода гарантира, че всеки компонент е отговорен само за своята част от функционалността, което прави системата лесна за разбиране, поддръжка и разширяване. Нови функционалности могат да се добавят чрез създаване на нови модели, ViewModel-и или визуални компоненти, без да се нарушава съществуващата логика или структура.

Основни модели и класове

В основата на приложението стоят няколко ключови модела и класа, които дефинират структурата на данните и начина, по който те се обработват и съхраняват. Всеки от тези класове изпълнява специфична роля в системата и е проектиран така, че да улеснява разширяването и поддръжката на кода.

Класът User представлява основния модел за потребителски профил. Той съдържа информация за пола на потребителя, предпочитаната дейност, любима технология и програмен език. Връзките между потребителя и останалите обекти са реализирани чрез навигационни свойства, които позволяват лесно извличане и промяна на свързаните данни. Например, всеки потребител има референция към предпочитана дейност (FavouriteActivity), която е обект от клас Activity, както и към предпочитана технология (FavouriteTechnology), която е обект от клас Technology. Освен това, програмираните езици са реализирани като енъми, което гарантира, че изборът винаги е валиден и ограничен до предварително дефинирани стойности.

Класовете Activity и Technology описват съответно дейностите и технологиите, които могат да бъдат избирани от потребителите. Всеки от тези класове съдържа уникален идентификатор и име, както и списък с потребители, които са ги избрали като предпочитани. Това позволява реализиране на двупосочни връзки и улеснява изпълнението на заявки към базата данни, като например извличане на всички потребители, които предпочитат дадена дейност или технология.

В слоя на ViewModel-ите основна роля играе класът UserFormViewModel. Той управлява логиката за формата на потребителя, зарежда и обработва списъците с възможни избори, обработва команди за добавяне на нови елементи и запис на данните. Всички ViewModel-и наследяват базовия клас BaseViewModel, който предоставя имплементация на интерфейса INotifyPropertyChanged и осигурява автоматично обновяване на интерфейса при промяна на данните.

Тази структура на моделите и класовете осигурява стабилна основа за развитието на приложението, като позволява лесно добавяне на нови функционалности и гарантира високо ниво на повторна употреба на кода.

Връзка с базата данни

Връзката между приложението и базата данни е реализирана чрез използването на Entity Framework Core – модерен ORM (Object-Relational Mapping) инструмент, който позволява лесно и ефективно управление на данните без необходимост от писане на SQL заявки. В основата на тази интеграция стои класът ApplicationDbContext, който наследява DbContext и дефинира всички основни таблици чрез свойства от тип DbSet<T>. В случая това са таблиците за потребители (Users), дейности (Activities) и технологии (Technologies).

При стартиране на приложението се инициализира контекстът на базата данни, като се използва локален SQLite файл (app.db). Ако базата данни не съществува, тя се създава автоматично, а всички необходими таблици и релации се инициализират според дефинициите в моделите. Това улеснява първоначалното разгръщане на системата и елиминира нуждата от ръчно създаване на структурата на базата.

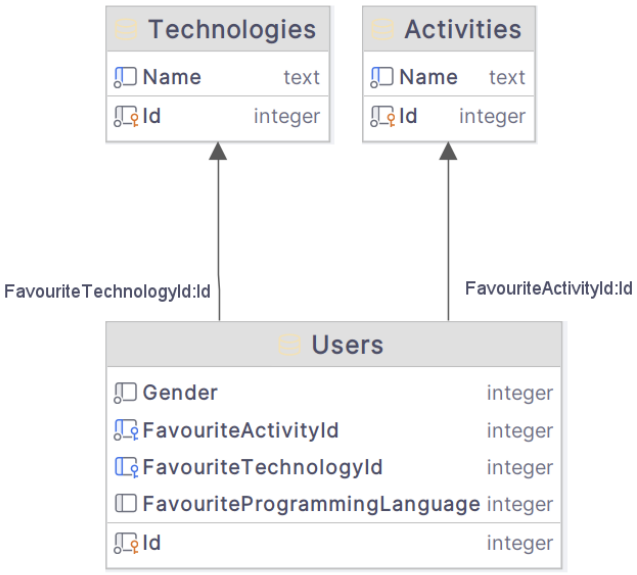
Връзките между различните обекти са реализирани чрез навигационни свойства и Fluent API конфигурация в метода OnModelCreating. Например, всеки потребител има връзка към предпочитана дейност и технология, а всяка дейност и технология поддържа списък с потребители, които са ги избрали. Освен това, при създаването на нови дейности и технологии се извършва проверка за уникалност на имената, което се реализира чрез индекси в базата данни. Това гарантира, че не могат да съществуват дублиращи се записи и се поддържа целостта на данните.

Благодарение на Entity Framework Core, всички операции по създаване, четене, обновяване и изтриване на данни се извършват по интуитивен и типово безопасен начин,

което значително намалява риска от грешки и улеснява поддръжката на приложението. Освен това, архитектурата позволява лесно преминаване към друга база данни или разширяване на модела при бъдещи нужди.

Структура на базата данни

Връзките между основните обекти в базата данни са представени на диаграмата на фигура 1.



Фиг. 1. Структура на базата данни

В центъра на модела стои класът User, който съдържа информация за всеки отделен потребител. Всеки потребител има връзка към един обект от тип Activity (чрез свойството FavouriteActivity), който представлява предпочитаната от него дейност. Аналогично, потребителят може да има връзка към един обект от тип Technology (чрез свойството FavouriteTechnology), който отразява предпочитаната технология. Освен това, всеки потребител има поле за предпочитан програмен език, реализирано като енъм ProgrammingLanguage, което гарантира, че изборът е ограничен до валидни стойности.

Всички основни ентитита в системата наследяват базовия клас BaseEntity, който съдържа уникален идентификатор (Id). Това осигурява унифицирана структура на всички таблици в базата данни и улеснява управлението на връзките между тях.

Тази структура на данните позволява лесно разширяване – например, ако е необходимо да се добавят нови типове връзки или допълнителни свойства към съществуващите класове. Връзките са реализирани така, че да поддържат целостта на данните и да осигуряват бърз достъп до необходимата информация, независимо от сложността на заявките.

Класове за създаване на списъци

В папката Model се намират класове, представляващи типове за списъци с опции:

- BaseOption – Абстрактен клас, представляващ основна опция със свойства Id (int) – уникален идентификатор на опцията, Name (string) – име на опцията. Предоставя метод ToString(), който връща името на опцията.
- BaseOptionList<TOption> – Абстрактен темплейтен клас, управляващ колекция от опции от тип TOption, където TOption разширява BaseOption. Съдържа Options – колекция от налични опции, SelectedOption – пропърти с текущо избрана опция, което може да се променя и protected метод SetOptions(), който заменя текущите опции с нови, запазвайки избраната опция, ако е налична. Метод SelectOption(int?), който избира опция по нейното id, ако има налична такава в списъка. Метод ContainsOptionWithName(string), който връща дали в списъка вече има опция с подаденото име. Методите IsSelected(TOption) и IsSelected(int), които проверяват дали опция е селектирана.
- DynamicOption – Клас, наследяващ BaseOption, предназначен за динамично добавени от потребителя опции. Не добавя нова функционалност, но служи за разграничаване на динамичните опции.
- DynamicOptionList – Клас, наследяващ BaseOptionList<DynamicOption>, управляващ списък от динамични опции. Предоставя метод SetOptions(), който позволява задаване на нови опции.
- SystemOption<TEnum> – Клас, наследяващ BaseOption, представляващ системно дефинирана опция, свързана с определен изброим тип (enum). Съжиржа SystemId (TEnum) изброима стойност, свързана с опцията. Използва се за предоставяне на предварително дефинирани опции.
- SystemOptionList<TEnum> - Клас, наследяващ BaseOptionList, управляващ списък от системно дефинирани опции. Конструкторът на класа е частен и не може да бъде извикван директно от други класове. Предоставят се статични методи ForGender() и ForProgrammingLanguages(), които създават списъци с опции съответно за пол и програмни езици. Метод SelectOption(TEnum?), който избира опция по дадена изброима стойност, метод IsSelected(TEnum), който проверява дали дадена изброима стойност е избрана.
- SystemDefinedOptions – Статичен клас. Съдържа предварително дефинираните колекции от системни опции – Genres, ProgrammingLanguages. Използва метода CreateReadonlyCollection<TEnum>(), който създава колекция от SystemOption<TEnum> за даден изброим тип.
- MixedOptionList<TEnum> – Класът наследява BaseOptionList<BaseOption> и съдържа като частно поле от тип SystemOptionList<TEnum>, като по този начин комбинира системно дефинирани и динамични опции. Конструкторът на класа е частен и не може да бъде извикван директно от други класове. Предоставят се статични методи като ForProgrammingLanguages(), който създава списък с опции за програмни езици, но с възможност за разширение. Първоначалната имплементация на повечето методи от BaseOptionList са предефинирани, като се запазва еднотипната работа с абстракцията, но се комбинират опциите от двата вътрешни списъка.

Добавяне на нов списък: Пример с видове домати

1. Дефиниране на изброим тип

```
public enum TomatoVariety  3 exposing APIs
{
    Cherry,
    Roma,
    Beefsteak
}
```

2. Създаване на списък със системни опции

```
public static class SystemDefinedOptions  2 usages
{
    public static readonly IReadOnlyCollection<SystemOption<TomatoVariety>> TomatoVarieties =
        CreateReadOnlyCollection<TomatoVariety>();
```

3. Добавяне на нов статичен метод за инициализация на SystemOptionList от изброимият тип

```
public class SystemOptionList<TEnum> : BaseOptionList<SystemOption<TEnum>> where TEnum : struct, Enum
{
    public static SystemOptionList<TomatoVariety> ForTomatoVarieties() =>
        new(SystemDefinedOptions.TomatoVarieties);
```

4. Създаваме нов списък със системно дефинирани опции от нашия изброим тип.

```
{
    var tomatoSystemOptionList = SystemOptionList<TomatoVariety>.ForTomatoVarieties();
}
```

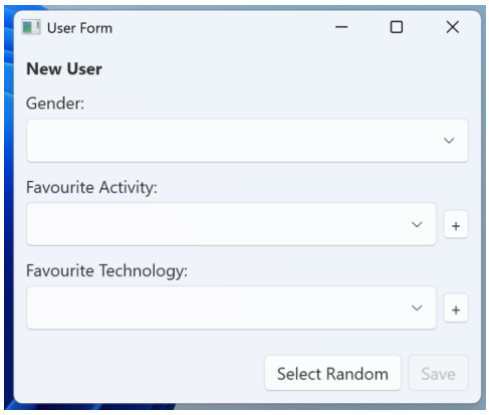
5. Разширяване до смесен списък.
   В MixedOptionList добавяме нов статичен метод за инициализация, който да зарежда базовата, системно дефинирана колекция от изброимият ни тип.

```
public class MixedOptionList<TEnum> : BaseOptionList<BaseOption> where TEnum : struct, Enum
{
    public static MixedOptionList<TomatoVariety> ForTomatoVarieties() =>
        new(SystemOptionList<TomatoVariety>.ForTomatoVarieties());
```

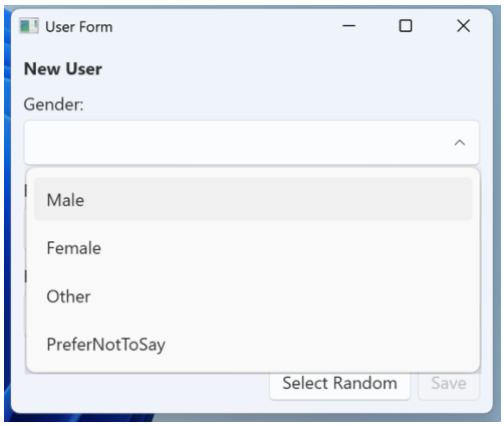
Демонстрация на приложението

Приложението е проектирано така, че да предоставя интуитивен и лесен за използване интерфейс, който позволява на потребителя да взаимодейства с всички основни

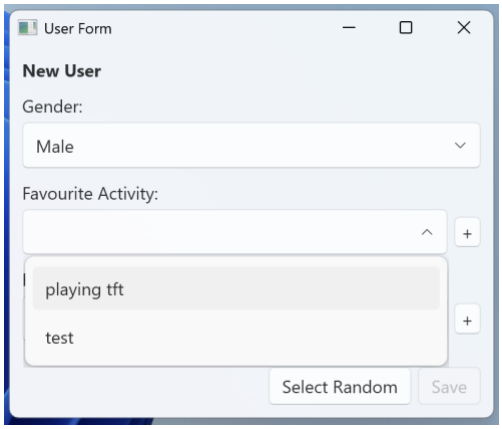
функционалности без необходимост от предварителни технически познания. Главният прозорец на системата съдържа форма, в която потребителят може да въведе или избере своите предпочитания – пол, любима дейност, предпочитана технология и програмен език. Всички тези избори се осъществяват чрез падащи менюта (ComboBox), които са обвързани с динамично зареждани списъци.



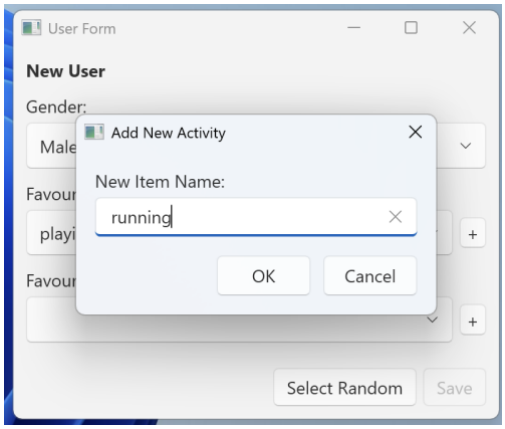
Фиг. 2. Начално състояние на приложението



Фиг. 3. Избиране на пол (статичен списък)

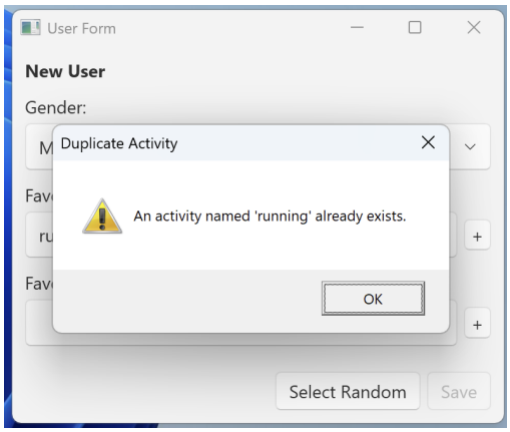


Фиг. 4. Избиране на любима активност (динамичен списък)

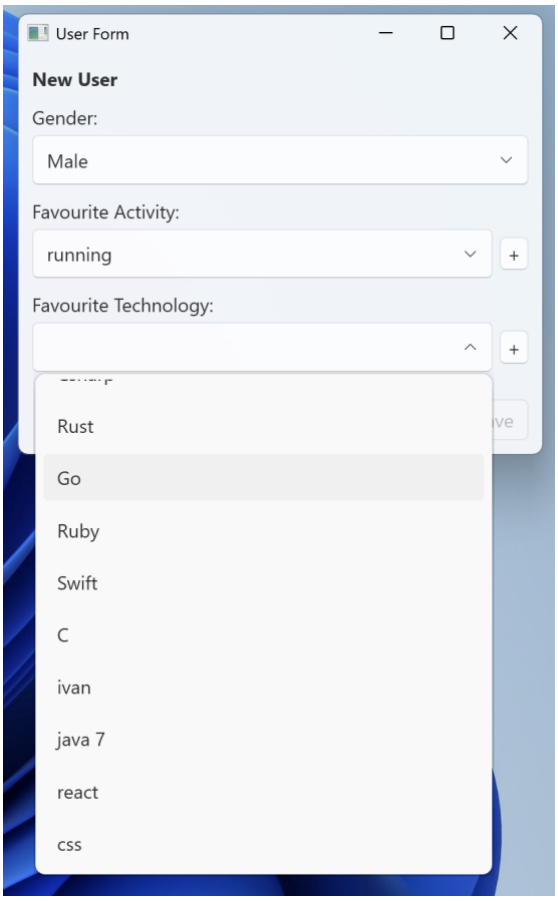


Фиг. 5. Добавяне на любима активност (към динамичен списък)

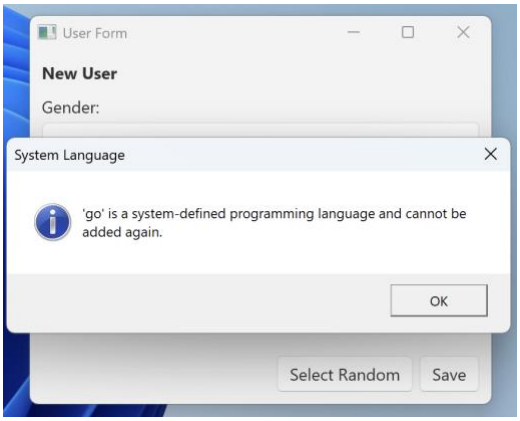
Особено внимание е отделено на възможността за добавяне на нови елементи към списъците с дейности и технологии. Това се реализира чрез специални бутони, разположени до съответните падащи менюта. При натискане на такъв бутон се отваря диалогов прозорец, в който потребителят може да въведе името на новата дейност или технология. След потвърждение, новият елемент се добавя към съответния списък и става достъпен за избор от всички потребители. Този механизъм осигурява гъвкавост и персонализация, като позволява на системата да се адаптира към индивидуалните нужди на всеки потребител.



Фиг. 6. Добавяне на вече съществуваща активност



Фиг. 7. Избиране на любима технология (смесен списък)



Фиг. 8. Грешка при добавяне на динамична стойност, при вече съществуваща системно дефинирана такава

Освен въвеждането и редакцията на данни, приложението предоставя възможност за избор на случаен потребител, което е особено полезно при демонстрации или тестове. След като потребителят направи своя избор и попълни необходимите данни, може да ги запише чрез бутонът Save. Системата автоматично валидира въведената информация и, при необходимост, уведомява потребителя за допуснати грешки или дублиране на стойности.

Интерфейсът е изчистен и последователен, като всички действия са ясно обозначени и достъпни. Това прави приложението подходящо както за индивидуални потребители, така и за учебни цели, където е важно да се демонстрират добри практики при работа с данни и потребителски интерфейс.

Заключение и приложения

В заключение, разработената система успешно реализира всички основни изисквания за съвременно настолно приложение, което управлява потребителски профили и свързаните с тях предпочитания. Чрез използването на MVVM архитектура и Entity Framework Core, проектът демонстрира добри практики в разделянето на логика, визуализация и достъп до данни. Това не само улеснява поддръжката и разширяването на системата, но и я прави подходяща за обучение и бъдещо развитие.

Гъвкавостта при работа със списъци – както системни, така и потребителски – позволява на приложението да се адаптира към различни сценарии и нужди, без да се налага промяна в кода. Възможността за динамично добавяне на нови дейности и технологии дава на потребителя усещане за контрол и персонализация, а валидирането на данните гарантира тяхната цялост и надеждност. Интерфейсът е изчистен, последователен и лесен за използване, което прави приложението достъпно както за крайни потребители, така и за студенти и преподаватели, които искат да се запознаят с принципите на WPF и MVVM.

Възможностите за бъдещо развитие на системата са многобройни. Архитектурата позволява лесно добавяне на нови типове списъци, разширяване на профилите с допълнителни характеристики, интеграция с външни източници на данни или дори преминаване към уеб-базирана версия. Това прави проекта отлична основа за по-сложни решения, които изискват стабилност, мащабируемост и добра организация на кода.

Сорс кодът на приложението е публикуван на: github.com/novabg03/wpf-option-lists