



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

Факултет „Компютърни системи и технологии”

**КУРСОВ ПРОЕКТ
ПО СОФТУЕРНИ ПЛАТФОРМИ**

**НА ТЕМА
„Coindex – Приложение за управление на
колекции от монети и банкноти“**

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

Съдържание

Въведение	3
Задание	4
Проектиране	5
Архитектурен модел и концепция	5
Структура на проектите и папките	5
Coindex.Core	5
Coindex.App	6
Coindex.Cli	6
Модели на данни и връзки	6
Потребителски сценарии и достъп до функционалностите	7
Възможности за разширение	8
Реализация	8
Инициализация на базата данни	8
Регистрация на зависимости, MVVM и binding	9
Преизползваемост на View и ViewModel	10
Потребителско ръководство (Резултати)	10
Графично приложение	10
Конзолен интерфейс	15
Заклучение	16

Въведение

В съвременния свят управлението на лични колекции от монети и банкноти се превръща не само в хоби, но и в необходимост за много ентусиасти и колекционери. С нарастващото разнообразие от предмети и нуждата от тяхното систематизиране, възниква потребността от иновативни софтуерни решения, които да улеснят този процес. Настоящият проект е посветен на разработката на приложение за управление на колекция от монети и банкноти, което предоставя на потребителя възможност да организира, категоризира и анализира своите ценности по удобен и интуитивен начин.

За реализацията на този проект е избрана платформата .NET MAUI (Multi-platform App UI) – съвременна кроссплатформена рамка, която позволява създаването на мобилни и десктоп приложения с единен код. .NET MAUI предоставя богати възможности за изграждане на модерни потребителски интерфейси, като същевременно осигурява достъп до всички необходими функционалности на съответната операционна система. Това прави платформата изключително подходяща за създаване на приложения, които трябва да работят еднакво добре на различни устройства – смартфони, таблети и компютри.

В основата на архитектурата на приложението стои моделът MVVM (Model-View-ViewModel). Този архитектурен шаблон разделя приложението на три основни слоя: модел (Model), изглед (View) и модел на изгледа (ViewModel). По този начин се постига ясно разграничение между визуализацията, бизнес логиката и данните, което значително улеснява поддръжката, разширяемостта и тестването на софтуера. MVVM е особено популярен при разработката на XAML-базирани приложения, каквито са тези, създадени с .NET MAUI.

Значението на разработваната система се изразява в това, че тя предоставя на потребителя цялостно решение за управление на личната му колекция. Чрез приложението всеки колекционер може да въвежда, редактира и организира своите монети и банкноти, да ги категоризира чрез тагове, да търси и филтрира по различни критерии, както и да получава статистика за състоянието на колекцията си. Всички данни се съхраняват локално на устройството, което гарантира сигурност и независимост от интернет връзка. По този начин системата отговаря на нуждите на съвременния потребител, като съчетава удобство, сигурност и модерни технологични решения.

Задание

Трябва да се изготви софтуерно приложение, предназначено за управление на колекция от монети и банкноти. Приложението следва да бъде реализирано с помощта на .NET MAUI и да използва архитектурния модел MVVM за разделяне на логиката, данните и потребителския интерфейс. Основната цел на проекта е да предостави на крайния потребител интуитивен и сигурен инструмент за съхранение, организация и анализ на личната му колекция.

Проектът трябва да има следните функционалности:

- **Въвеждане и съхранение на данни:** Приложението трябва да позволява създаване и поддържане на база данни с колекционерски предмети – монети и банкноти. Всеки предмет трябва да бъде описан с атрибути като име, описание, година на издаване, държава, номинал, състояние и други специфични характеристики. Данните трябва да се съхраняват локално на устройството на потребителя.
- **Категоризация чрез тагове:** Всеки предмет трябва да може да бъде асоцииран с един или повече тагове (етикети), които да позволяват гъвкаво групиране и филтриране на колекцията според предпочитанията на потребителя. Таговете трябва да могат да се създават и редактират от потребителя.
- **Управление на предмети и тагове:** Приложението трябва да предоставя възможност за добавяне и редактиране на предмети и тагове. Интерфейсът трябва да бъде интуитивен и лесен за използване, така че всички основни операции да могат да се извършват бързо и без затруднения.
- **Търсене и филтриране:** Системата трябва да осигурява механизми за търсене и филтриране на предметите по различни критерии – име, таг и състояние. Това ще позволи на потребителя лесно да открива желаната информация в своята колекция.
- **Локално съхранение и сигурност:** Всички данни трябва да се съхраняват локално на устройството, като по този начин се гарантира поверителност и независимост от външни сървъри или интернет връзка.
- **Използване на едно и същото View от няколко ViewModel-a:** Определени визуални компоненти (View) трябва да могат да бъдат използвани от различни ViewModel-и, което улеснява повторната употреба на интерфейсни елементи и гъвкавостта на приложението.
- **Използване на един и същ ViewModel от няколко View-та:** Един и същ ViewModel трябва да може да бъде използван от различни View-та, което позволява споделяне на логика и данни между различни части на приложението.

Проектът трябва да реализира опростена система за управление на колекция от монети и банкноти, която да съчетава функционалност и потребителски интерфейс, както и сигурност на данните. Изпълнението на тези изисквания ще позволи на потребителите ефективно да организират, анализират и поддържат своите колекции.

Проектиране

Архитектурен модел и концепция

Coindex е проектиран като многослойна система, която ясно разделя бизнес логиката, данните и потребителския интерфейс. Основната цел на тази архитектура е да осигури преизползваемост, лесна поддръжка и възможност за бъдещо разширяване. В основата на системата стои ядрото Coindex.Core, което съдържа всички бизнес правила, модели на данни, услуги, интерфейси за достъп до данни и инфраструктурни компоненти. Този слой е напълно независим от потребителския интерфейс и може да бъде използван от различни приложения. Към него са изградени две основни приложения: Coindex.App – графично приложение с модерен потребителски интерфейс, реализирано с .NET MAUI и архитектура MVVM, и Coindex.Cli – конзолно приложение, което демонстрира преизползваемостта на Core пакета и позволява достъп до основните функционалности чрез команден ред.

Структура на проектите и папките

Coindex.Core

Coindex.Core е разделен на три основни слоя: Domain, Application и Infrastructure, всеки от които има ясно дефинирана роля в архитектурата на системата.

- **Domain слой** – Сърцето на бизнес модела, съдържащо само бизнес логика и абстракции, без зависимости към инфраструктурата:
 - **Domain/Entities/** – Съдържа основните модели на данни (CollectableItem, Coin, Bill, Tag и др.), които описват предметите в колекцията.
 - **Domain/Enums/** – Включва изброими типове, използвани в моделите, като например състоянието на предметите.
 - **Domain/Interfaces/** – Съдържа интерфейси за базови абстракции.
- **Application слой** – Реализира бизнес логиката на приложението, като координира работата между различни програмни единици и модели:
 - **Application/Services/** – Реализира бизнес услуги за работа с предмети, тагове и други бизнес процеси.
 - **Application/Interfaces/** – Съдържа интерфейси за услугите и репозиторията, които предоставя на потребителите на пакета.
- **Infrastructure слой** – Отговаря за техническите детайли и достъпа до външни ресурси, като база данни и други услуги:
 - **Infrastructure/Data/** – Реализира контекста на базата данни (ApplicationDbContext), инициализация и конфигурация на Entity Framework.
 - **Infrastructure/Repositories/** – Имплементира репозиторията за достъп до данни и операции с тях.

В бъдеще може всеки един от слоевете на Coindex.Core пакетът да бъде отделен в различен проект, който да отговаря за конкретен аспект от системата, което ще улесни мащабирането, повторната употреба и поддръжката на кода. Тази модулна структура следва принципите на чиста архитектура (Clean Architecture) и DDD (Domain-Driven Design), като гарантира добра разделяемост на отговорностите и улеснява внедряване на нови функционалности, без да се нарушава стабилността на останалите слоеве.

Coindex.App

Coindex.App проектът следва MVVM (Model-View-ViewModel) архитектурата, като разделя представянето на данни, потребителските взаимодействия и логиката, свързана с визуализацията. Това позволява лесна поддръжка, разширяемост и повторна употреба на компонентите в различни платформи.

- **Views/** – Съдържа XAML страници, реализиращи потребителския интерфейс (HomePage, CollectableItemsPage, TagsPage и др.).
- **ViewModels/** – Включва ViewModel-и, които съдържат логиката и binding към View.
- **Converters/** – Съдържа класове за преобразуване на данни за визуализация.
- **Resources/** – Включва ресурси като стилове, цветове, изображения и помощни файлове.
- **Platforms/** – Съдържа реализации за различни операционни системи.
- **MauiProgram.cs** – Основният конфигурационен файл за DI контейнера и стартиране на приложението.

Благодарение на MVVM модела и възможностите на .NET MAUI, интерфейсът е едновременно гъвкав и лесен за адаптиране към различни устройства и платформи.

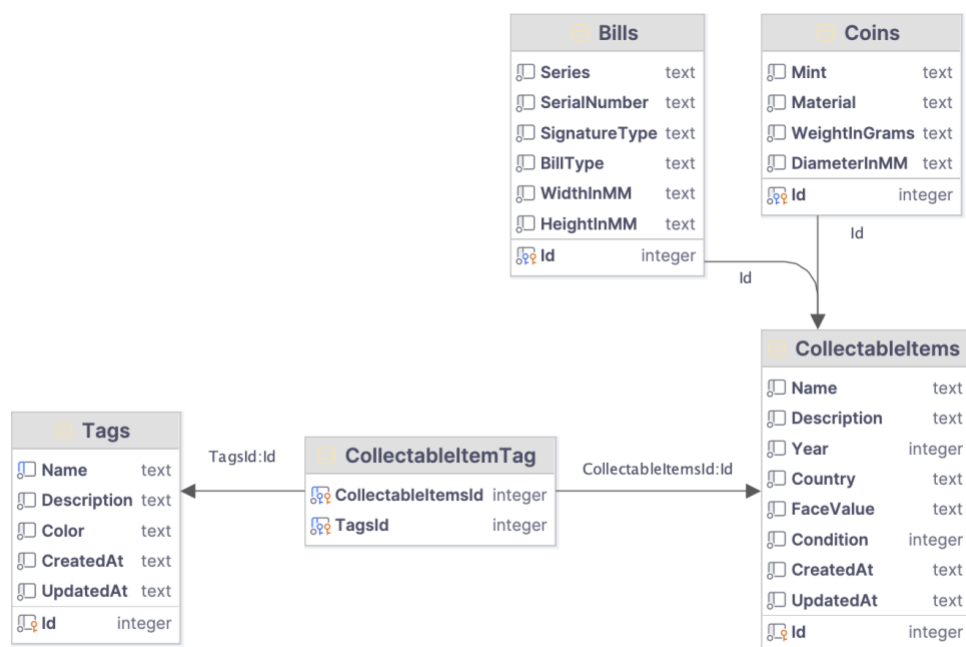
Coindex.Cli

Coindex.Cli предоставя възможност за извършване на основни операции, без нуждата от графичен интерфейс, което го прави подходящо за автоматизирани скриптове, администраторски действия или интеграция с други системи. **Program.cs** е основният входен файл, който дефинира CLI командите и логиката на конзолното приложение.

Модели на данни и връзки

Моделите на данни в Coindex са дефинирани в слоя Domain на Coindex.Core и отразяват основните обекти, с които работи системата. Централно място заема абстрактният клас `CollectableItem`, който описва общите характеристики на всеки колекционерски предмет. От него наследяват конкретните класове `Coin` (монета) и `Bill` (банкнота), които добавят специфични атрибути според типа на предмета. Всеки `CollectableItem` съдържа информация за име, описание, година на издаване, държава, номинал, състояние и други характеристики, които позволяват детайлно описание на всеки елемент от колекцията.

Важна роля в модела играе и класът `Tag`, който представлява етикет за категоризация на предметите. Връзката между `CollectableItem` и `Tag` е реализирана като много-към-много, което позволява един предмет да бъде асоцииран с множество тагове, а всеки таг да се използва за различни предмети. Това осигурява гъвкава категоризация и възможност за ефективно филтриране и търсене в колекцията.



Фиг. 1. Диаграма на базата данни

Реализацията на връзките между моделите се осъществява чрез Entity Framework Core, който автоматично управлява асоциативните таблици и осигурява лесна работа с релационната база данни. Всички данни се съхраняват в локална база данни SQLite, което гарантира сигурност и независимост от външни сървъри. Този подход позволява разширяемост на модела и добавяне на нови типове предмети или допълнителни атрибути при необходимост.

Потребителски сценарии и достъп до функционалностите

Coindex е създаден така, че да отговаря на разнообразни потребителски нужди чрез различни интерфейси. Типичният потребител на Coindex.App може лесно да добавя нови предмети към колекцията си, да ги категоризира с тагове, да редактира или изтрива съществуващи записи, както и да търси и филтрира по различни критерии като име, държава или състояние. Всички тези действия се извършват през интуитивен графичен интерфейс, където всяка страница (View) е свързана с ViewModel, който управлява бизнес логиката и комуникацията с данните. Например, при добавяне на нова монета, потребителят попълва формата във View, а ViewModel-ът обработва въведената информация, валидира я и я записва в базата чрез съответния Service.

За по-напреднали потребители и за автоматизация на задачи е наличен Coindex.Cli, който позволява изпълнение на същите операции чрез команди в терминала. Например, с една команда може да се добави нов предмет или да се изведе списък с всички банкноти по определен критерий. Това е възможно благодарение на споделената бизнес логика в Coindex.Core, която гарантира еднакво поведение и валидиране на данните независимо от използвания интерфейс.

Тази архитектура позволява на Coindex да бъде едновременно лесен за използване от обикновени потребители и мощен инструмент за напреднали, като същевременно осигурява последователност, сигурност и разширяемост на функционалностите.

Възможности за разширение

Архитектурата на Coindex е проектирана така, че да улеснява бъдещото разширяване и интеграция на нови функционалности. Благодарение на ясното разделение между бизнес логика, инфраструктура и потребителски интерфейс, системата може лесно да се адаптира към нови изисквания и технологии. Сред основните възможности за разширение са:

- **Импорт и експорт на данни:** Могат да се добавят модули за импортиране и експортиране на колекцията във формати като CSV, Excel или JSON, което ще улесни прехвърлянето на данни между различни устройства или приложения.
- **Синхронизация между устройства:** Чрез интеграция с облачни услуги или собствен сървър може да се реализира синхронизация на колекцията между няколко устройства, така че потребителят винаги да има достъп до актуалните си данни.
- **Добавяне на снимки към предметите:** Възможно е разширяване на модела и интерфейса, така че към всеки предмет да се прикачват снимки, което ще направи колекцията по-визуална и информативна.
- **Локализация и превод на интерфейса:** Интерфейсът може да бъде преведен на различни езици, което ще направи приложението достъпно за по-широка аудитория.
- **Разширяване на CLI с нови команди:** Конзолното приложение може да бъде допълнено с нови команди за по-детайлно управление, анализ или автоматизация на задачи.
- **Добавяне на уеб интерфейс:** Към съществуващата архитектура може да се добави уеб приложение, което използва Coindex.Core за бизнес логиката и данните, като по този начин се осигурява достъп до колекцията през браузър.
- **Интеграция с външни каталози и API:** Възможно е да се реализира автоматично попълване на информация за монети и банкноти чрез връзка с външни каталози или публични API.

Тези възможности за разширение са лесно реализуеми благодарение на модулната архитектура и използването на утвърдени софтуерни принципи като MVVM, DDD и Dependency Injection. Това гарантира, че Coindex може да се развива и адаптира според нуждите на потребителите и технологичните тенденции.

Реализация

Инициализация на базата данни

Инициализацията на базата данни е напълно автоматизирана и се извършва при стартиране на приложението. Във файла MauiProgram.cs се определя пътят до локалния файл на базата данни чрез `Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "coindex.db3")`, което гарантира, че всеки потребител работи със собствена база. След това се инициализира SQLite с `Batteries_V2.Init()`, което е задължително за .NET MAUI.

Контекстът на базата данни (`ApplicationDbContext`) се регистрира в DI контейнера с помощта на `AddDbContext`, като се задава използването на SQLite и конкретния път до файла. Класът `DatabaseInitializer`, дефиниран в `Coindex.Core.Infrastructure.Data.DatabaseInitializer.cs`, получава чрез `dependency injection` инстанции на `ApplicationDbContext` и услуга за генериране на тестови данни (`ICollectionableItemDataGeneratorService`). В метода `Initialize()` се извиква `context.Database.EnsureCreated()`, което създава базата и всички необходими таблици, ако те не съществуват. Ако в базата няма монети или банкноти, се извиква методът `SeedTestData()`, който добавя примерни тагове, 25 монети и 10 банкноти с произволно генерирани атрибути като име, година, държава, номинал, състояние и други. За всяка монета и банкнота се асоциират и тагове, избрани на случаен принцип от предварително създадените. Например, в кода се вижда как се създава списък с тагове, които се добавят в базата чрез `context.Tags.AddRange(tags)` и се запазват с `context.SaveChanges()`. След това се създават обекти от тип `Coin` и `Bill`, като за всеки от тях се задават свойства чрез методи на `dataGenerator`, и накрая се добавят в базата с `context.Coins.AddRange(newCoins)` и `context.Bills.AddRange(newBills)`.

Този подход осигурява на потребителя начални данни за тест и демонстрация на функционалностите, без да се налага ръчна намеса. Всички данни се съхраняват локално, което гарантира сигурност и независимост от външни сървъри, а архитектурата позволява лесно разширяване на модела и добавяне на нови типове данни при необходимост.

Регистрация на зависимости, MVVM и binding

`Coindex.App` регистрацията на зависимости се реализира чрез DI контейнера на .NET MAUI, като всички основни компоненти на приложението се регистрират в метода `CreateMauiApp` на класа `MauiProgram.cs`. Например, контекстът на базата данни се регистрира с `„builder.Services.AddDbContext<ApplicationDbContext>(options => options.UseSqlite($"Data Source={dbPath}"));`, а репозиториите и услугите се регистрират с `AddScoped` или `AddSingleton` според нуждите. `ViewModel`-ите се регистрират с `AddTransient` или `AddSingleton`, като например `„builder.Services.AddTransient<TagEditViewModel>();` и `„builder.Services.AddSingleton<CollectableItemsViewModel>();`. Всяка страница (`View`) също се регистрира, за да може да получи нужния `ViewModel` чрез `dependency injection`.

MVVM архитектурата е реализирана чрез ясно разделение между визуализацията (`XAML` страници), `ViewModel`-ите (логика и `binding`) и моделите (данни). Например, във файла `CollectableItemsPage.xaml.cs` се инжектира `CollectableItemsViewModel` и се задава като `BindingContext` на страницата, което позволява всички `binding`-и в `XAML` да работят коректно. Във `ViewModel`-ите се използва `CommunityToolkit.Mvvm` за автоматично генериране на свойства и команди. Например, в `CollectableItemsViewModel` има `ObservableCollection`-и като `Items`, които са обвързани със списъци в `XAML` чрез `{Binding Items}`. Командите като `InitializeCommand`, `EditItemCommand` и `SaveCommand` се дефинират с атрибута `[RelayCommand]` и се използват в `XAML` за реакция на действия от потребителя, например чрез бутони или жестове. Във файла `CollectableItemsPage.xaml` списъкът с предмети е обвързан с `ItemsSource="{Binding Items}"`, а командата за зареждане на още елементи е обвързана с `RemainingItemsThresholdReachedCommand="{Binding LoadMoreItemsCommand}"`. Този подход осигурява двупосочна връзка между потребителския интерфейс и `ViewModel`-а, като интерфейсът винаги отразява актуалното състояние на данните, а действията на потребителя се обработват от `ViewModel`-а.

Преизползваемост на View и ViewModel

Архитектурата на Coindex е проектирана така, че да осигурява максимална преизползваемост на View и ViewModel-и, което е ключово за мащабируемостта и поддръжката на приложението. Един и същ View може да работи с различни ViewModel-и, като например страницата TagEditPage, която може да използва както TagEditViewModel за редакция на съществуващ таг, така и TagCreateViewModel за създаване на нов таг.

Това се реализира чрез динамично задаване на BindingContext според параметрите на навигацията. В кода на TagEditPage.xaml.cs се използва IServiceProvider, за да се избере подходящият ViewModel в зависимост от това дали се редактира или създава таг, като BindingContext се задава динамично: ако е подаден валиден идентификатор, се използва TagEditViewModel, в противен случай – TagCreateViewModel.

Освен това, един и същ ViewModel може да бъде използван от няколко View-та, както е при TagsViewModel, който се използва както в началната страница HomePage, така и в страницата за тагове TagsPage. Това позволява споделяне на логика и данни между различни части на приложението, без дублиране на код, и гарантира, че промени в логиката се отразяват автоматично навсякъде, където се използва този ViewModel. За по-голяма гъвкавост се използват интерфейси като ITagEditViewModel, което позволява различни реализации да бъдат използвани от един и същ View, улеснявайки разширяването и поддръжката на кода.

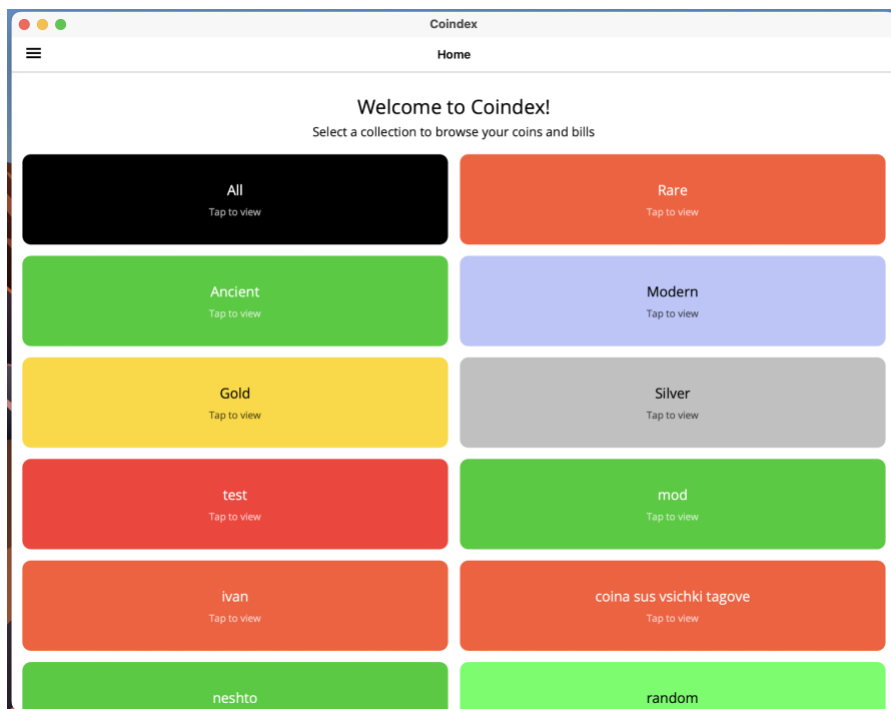
Навигацията между страниците се реализира чрез Shell навигация, като параметрите се предават към ViewModel-а чрез атрибута [QueryProperty], което позволява един и същ ViewModel да обработва различни данни в зависимост от контекста на навигацията. Например, CollectableItemEditViewModel се използва както за създаване, така и за редакция на предмети, като в зависимост от това дали е подаден ID на съществуващ предмет, ViewModel-ът зарежда данните за редакция или инициализира празни полета за създаване на нов предмет. Във ViewModel-а се използват свойства като IsCreateMode, които определят режима на работа, а методи като Initialize() и LoadItem() се грижат за зареждането и обработката на данните.

Този подход осигурява висока степен на преизползваемост, гъвкавост и разширяемост, като същевременно поддържа ясна архитектура и улеснява работата на бъдещи програмисти, които могат лесно да добавят нови функционалности или да разширяват съществуващите, без да се налага дублиране на код или промяна на вече работещи компоненти.

Потребителско ръководство (Резултати)

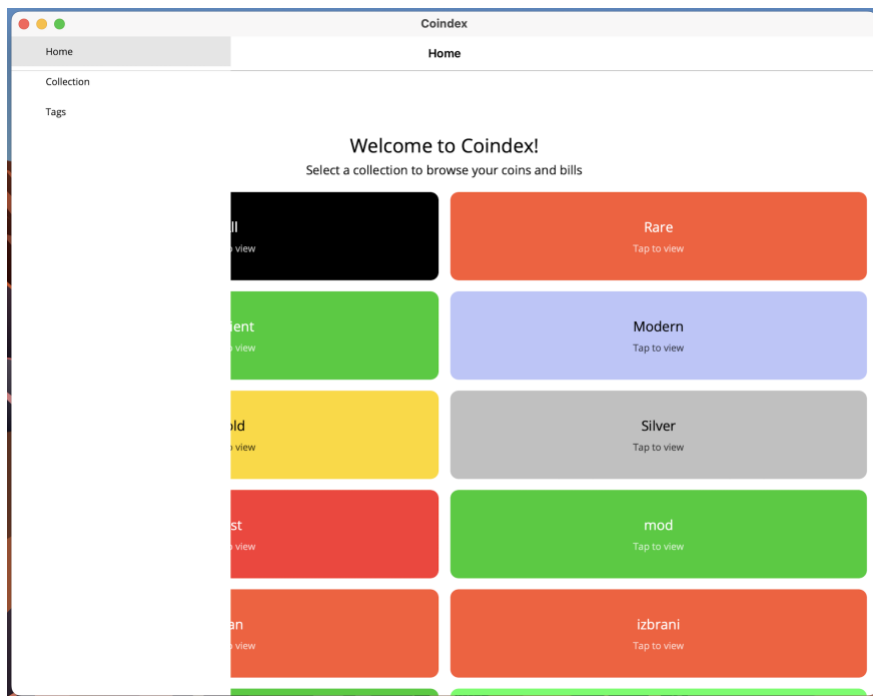
Графично приложение

Началната страница на приложението Coindex показва списък с наличните колекции от монети и банкноти. Всяка колекция е представена като бутон с име и надпис „Tap to view“. При натискане върху някой от тях се отваря съдържанието на съответната колекция. Бутонът „All“ показва всички добавени елементи, без значение от категорията. В горния ляв ъгъл има бутон за отваряне на менюто с допълнителни опции. Началният екран представлява един смесен списък, който винаги ще има опция за всички елементи. След като потребителят добави нови тагове, те автоматично ще бъдат добавяни тук.



Фиг. 2. Начална страница

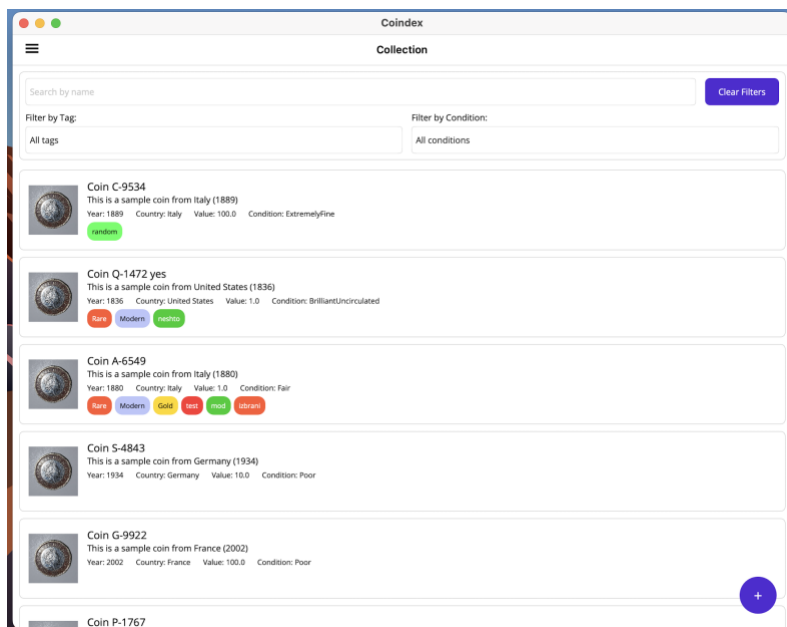
При натискане на бутона в горния ляв ъгъл се отваря страничното меню. Оттук потребителят има достъп до основните секции на приложението – началната страница (Home), всички записани елементи (Collection) и списъка с тагове (Tags), които е създал. Менюто позволява бърза навигация между различните части на приложението, без да се връщаме назад ръчно.



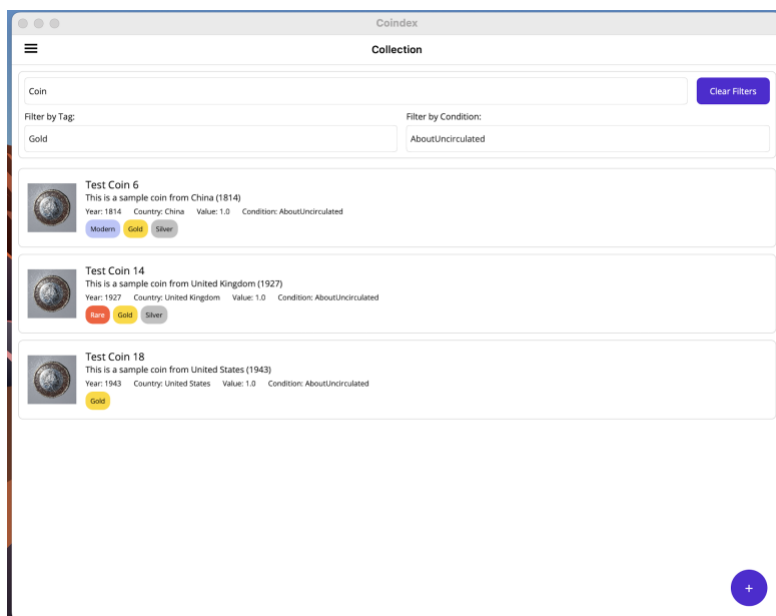
Фиг. 3. Странично меню

Екранът „Колекция от елементи“ показва пълната колекция от добавени елементи. Всеки елемент съдържа информация като име, година, държава, стойност, състояние и тагове. Горе има търсачка за филтриране по име, както и два допълнителни филтъра: Filter by Condition – списък с предварително дефинирани стойности, зададени от програмиста, Filter by Tag – списък с потребителски тагове, зареждани от базата данни.

Потребителите могат да добавят нови тагове. Бутонът Clear Filters премахва активните филтри. При натискане върху конкретен елемент се отваря страница с повече подробности за него. В долния десен ъгъл има бутон за добавяне на нов елемент към колекцията.

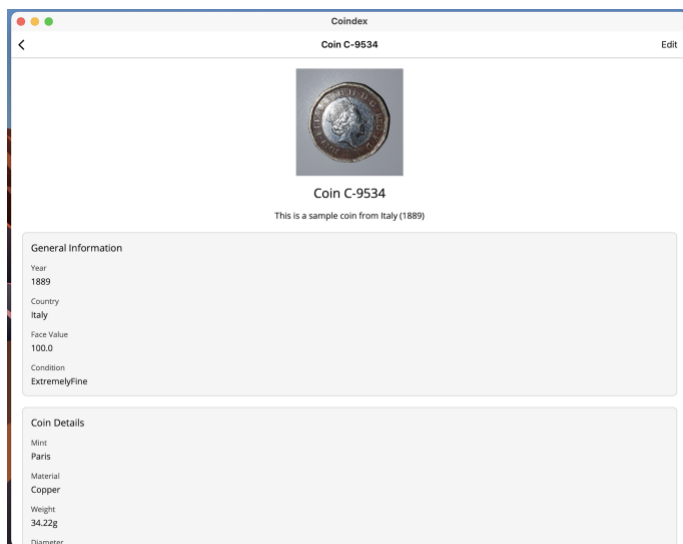


Фиг. 4. Колекция от елементи



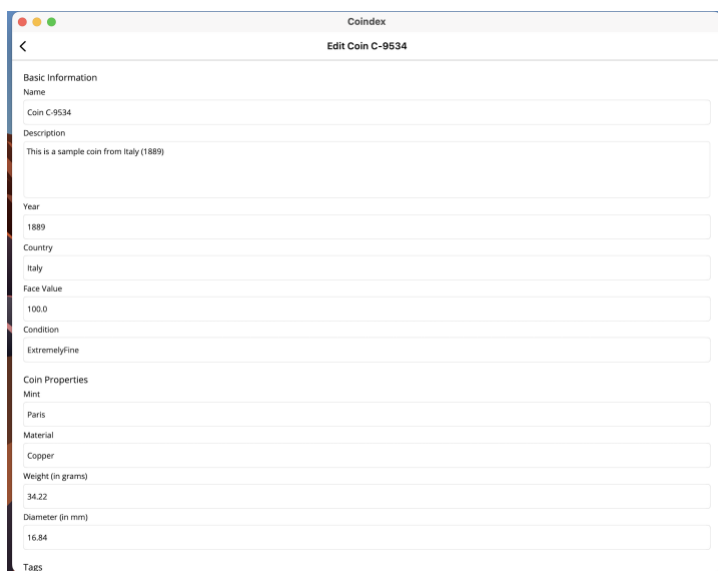
Фиг. 5. Колекция с приложени филтри

След избиране на конкретен елемент от колекцията, се зарежда страница с подробна информация за него. В горната част се вижда изображението на монетата, нейното име и кратко описание. Под тях са подредени две секции: General Information и Coin Details. В тях са представени основните характеристики на монетата като година на емисия, държава, номинал, състояние, монетен двор, материал, тегло и диаметър. Отдясно горе се намира бутон „Edit“, чрез който може да се премине към редактиране на данните.



Фиг. 6. Информация за монета

На екрана за редактиране потребителят може да редактира цялата информация за конкретната монета. Формата съдържа всички полета от страницата с детайли, като те са предварително попълнени с текущите стойности. Полетата са разделени в две групи – Basic Information (име, описание, година, държава, номинал, състояние) и Coin Properties (монетен двор, материал, тегло, диаметър). Промените се запазват чрез бутон, който се намира по-надолу.

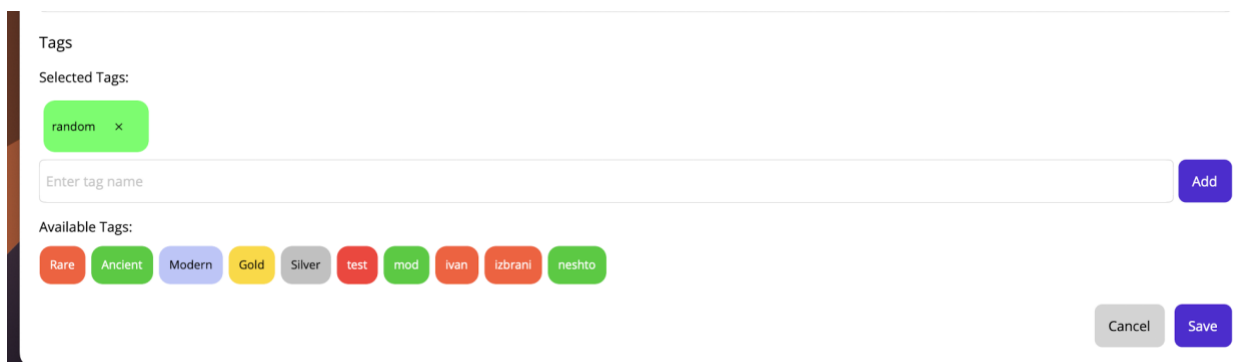


Фиг. 7. Редактиране на монета

В долната част на формата за редакция на монета се намира секция за управление на таговете. Потребителят може да избира измежду вече съществуващи тагове или да добави нов, като въведе името му в текстовото поле и натисне бутона Add. При въвеждане на текст, списъкът с налични тагове автоматично се филтрира, което улеснява избора.

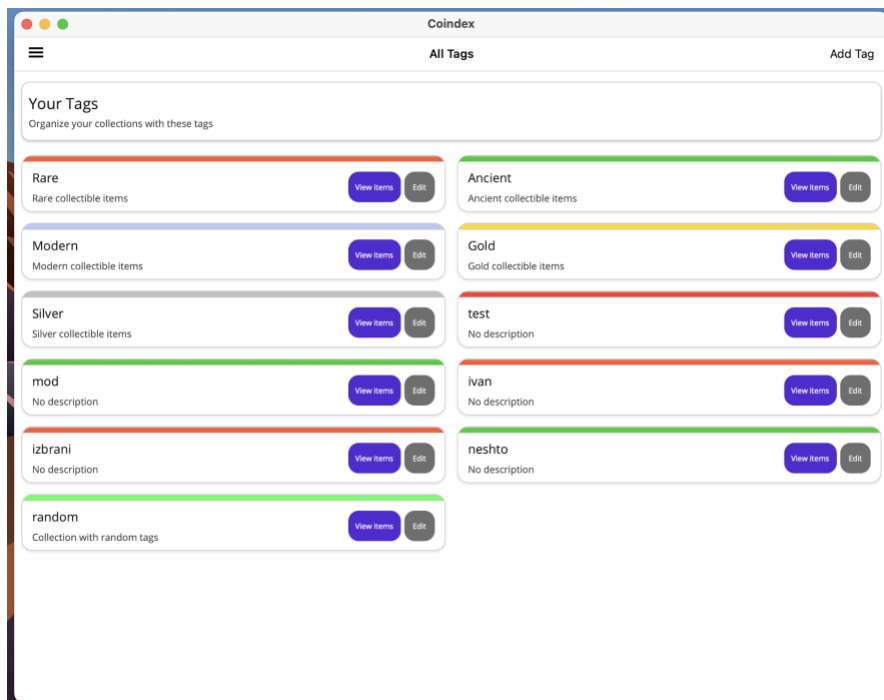
Избраните тагове се показват в отделно поле с възможност за премахване (бутон "x"). Създадените нови тагове по-късно могат да бъдат редактирани от друг екран, където може да се промени тяхното описание и цвят.

В долния десен ъгъл има бутона Cancel за отказ и Save за запазване на промените.



Фиг. 8. Редактиране на монета - тагове

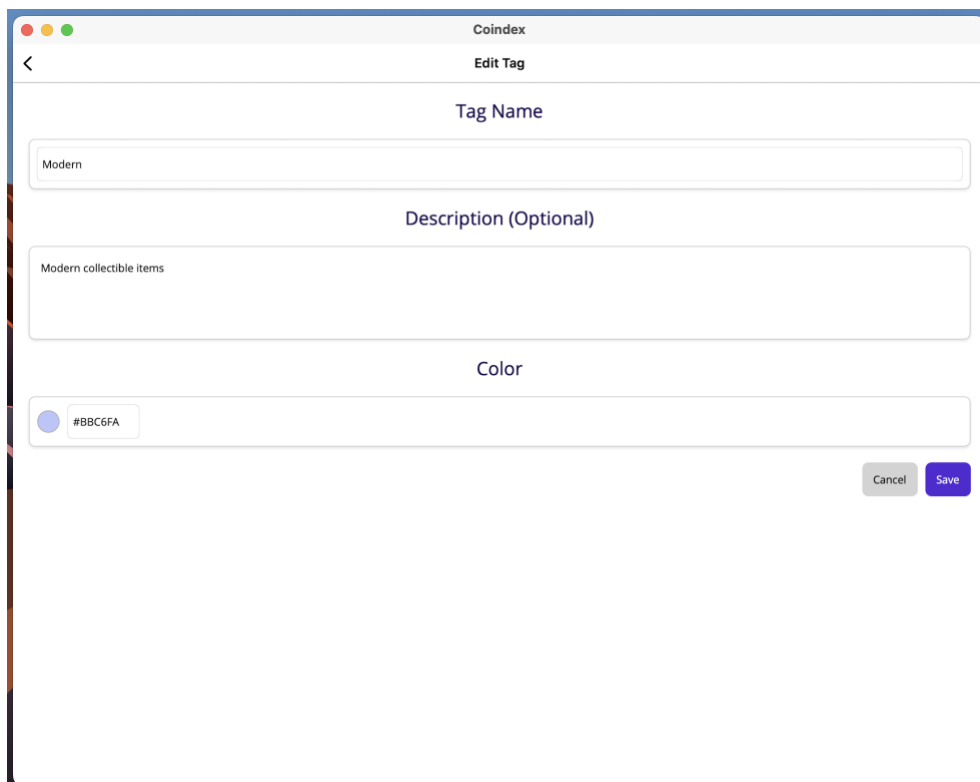
Екранът „Списък с тагове“ показва всички тагове, създадени от потребителя. Всеки таг е представен като отделна карта, която съдържа неговото име, кратко описание (ако има такова), както и два бутона: View items – показва всички елементи, които използват съответния таг, Edit – отваря форма за редакция на таг, където могат да се променят името, описанието и цветът му. В горната част на екрана има бутон Add Tag, чрез който може да се създаде нов таг. По този начин потребителят лесно организира и управлява своите колекции.



Фиг. 9. Списък с тагове

На екран „Редактиране на таг“ потребителят може да редактира избран таг. Формата включва три основни полета: Tag Name – името на тага, Description – кратко описание, което помага за по-добра организация на колекциите, Color – визуалният цвят на тага, зададен чрез hex код.

Промените могат да бъдат запазени с бутона Save, а с Cancel се отказва редакцията и се връща към предишния екран.



Фиг. 10. Редактиране на таг

Конзолен интерфейс

Coindex CLI е лесен за използване инструмент за достъп до същата база данни, която използва и графичното приложение. Позволява преглед, търсене и филтриране на колекционерски обекти директно от терминала.

Команда

```
coindex help
coindex list
coindex list --page <номер>
coindex list --tag <име>
coindex list --condition <стойност>
coindex show <ID>
coindex tags
coindex search <текст>
coindex stats
```

Описание

Показва помощ и налични команди
Списък с всички обекти
Преглед на конкретна страница
Филтриране по етикет
Филтриране по състояние
Детайли за конкретен обект
Показва всички етикети
Търсене по име
Статистика за колекцията

Заклучение

Разработката на Coindex демонстрира как чрез съвременни технологии и добри архитектурни практики може да се създаде ефективно и интуитивно приложение за управление на лични колекции. Чрез използването на .NET MAUI и архитектурния модел MVVM, системата съчетава модерен потребителски интерфейс с добре структурирана и лесно разширяема логика. Разделянето на приложението на независими модули, както и ясното разграничение между бизнес логика, данни и визуализация, осигурява стабилна основа за поддръжка и бъдещо развитие.

Съчетаването на графичен и конзолен интерфейс прави Coindex достъпно както за обикновени потребители, така и за напреднали, които се нуждаят от по-бърз и автоматизиран достъп до данните си. Системата предоставя всички основни функционалности за създаване, редактиране, категоризация и търсене на колекционерски обекти, като същевременно гарантира сигурност чрез локално съхранение на информацията.

Проектът притежава силен потенциал за развитие – както чрез добавяне на нови функции (импорт/експорт, синхронизация, визуализация чрез изображения), така и чрез интеграция с външни ресурси и разширяване на платформената му достъпност. Coindex не просто решава конкретен проблем на колекционерите, но служи и като пример за изграждане на добре организирани, модулни и мащабируеми приложения, базирани на съвременни софтуерни концепции.

Сорс кодът на приложението е публикуван на: github.com/novabg03/coindex