Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Розробка інтернет-ресурсу для колекціонерів монет з використанням ASP.NET Core

Текстова частина до курсової роботи
За спеціальністю «Інженерія програмного забезпечення» 121

Керівник курсової роботи
ст.в. Борозенний С.О.

_____
(підпис)
“____” _____ 2021 р.

Виконав студент 4-го курсу

Білокінь Данило

(прізвище та ініціали)

“____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,

доцент, к.т.н.

_____ О. П. Жежерун

(підпис)

“____”_____ 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Білокіню Данилу

__4-го__ курсу факультету інформатики

ТЕМА: Розробка інтернет-ресурсу для колекціонерів монет з використанням ASP.NET Core.

Вихідні дані:

- Веб-застосування для колекціонерів монет

Зміст ТЧ до курсової роботи:

Вступ
1 Аналіз предметної області. Постановка завдання курсової роботи
2 Теоретичні відомості
3 Опис реалізації програмного продукту
Висновки
Список джерел
Додатки (за необхідністю)

Дата видачі “___” _____ 2021 р. Керівник _____

(підпис)

Завдання отримано _____

(підпис)

Календарний план виконання курсової роботи

Тема: Розробка нейронної мережі для розпізнавання графічних об’єктів

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Жовтень-листопад 2020р.	
2.	Огляд літератури за темою роботи	Листопад-грудень 2020р.	
3.	Аналіз сучасних методів	Січень 2021р.	
4.	Реалізація серверної частини застосунку	Лютий 2021р.	
5.	Реалізація клієнтської частини застосунку та тестування	Квітень 2021р.	
6.	Написання текстової частини.	Квітень 2021р.	
7.	Перегляд курсової роботи науковим керівником	Квітень 2021р.	
9.	Перегляд змісту роботи керівником	07.04. 2021	
10.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	09.04. 2021	
11.	Створення презентації	10.04.2021	
12.	Захист курсової роботи	18.04.2021	

Студент Білокінь Д.Т. _____

Керівник Борозенний С. О. _____

“_____”_____ 2021 р

Зміст

Анотація

Робота присвячена вивченню роботи фреймворку для розробки веб-застосунків на мові програмування C# ASP.NET Core та побудова застосунку з користувацьким інтерфейсом з можливостями ведення реєстру наявних монет в колекції за певними критеріями та обміну монетами між користувачами, на основі отриманих знань.
Реалізація на мові C# з використанням ASP.NET Core та EntityFramework Core фреймворків .

Вступ

Для розробки програмних додатків використовують платформи, що називаються фреймворк (або програмний каркас). За визначенням Кембриджського словника “фреймворк” має наступну дефініцію: “підтримуюча структура, навколо якої можна будувати” тп “система правил, ідей або вірувань, що використовуються з метою планування або прийняття рішень”. Отже, фреймворк забезпечує основу, на якій будуються програми розробниками програмного забезпечення для конкретної платформи. Структура може складатися з визначених класів та функцій, що надалі спрощує етап розробки додатку.

Однією з таких програмних платформ є продукт від Microsoft, випущений у 2002 році, .NET - кросплатформенна відкрита для розробників платформа, що складається з інструментів, мов програмування та бібліотек для створення великого різноманіття типів додатків. Базова платформа передбачає такі мови програмування як C#, F#, Visual Basic, базові бібліотеки для роботи зі стрічками, датами, файлами тощо, а також редактори та інструменти для багатьох систем, а саме: macOS, Windows, Linux та Docker [1].

Актуальність:

ASP.NET - фреймворк для будування веб-додатків та сервісів за допомогою використання .NET та C#. На сьогоднішній день ASP.NET посідає другі місця в категоріях найпопулярніших фреймворків: серед всіх сайтів, серед топ-10000 сайтів, серед топ-100000 сайтів та серед топ-1000000 сайтів. В світі цю технологію використовують 5.92% сайтів (9258086), з яких 2671 сайтів в Україні. Крос-платформою .NET є .NET Core, що

використовується для веб-сайтів, служб, настільних додатків. ASP.NET Core через відносно недавню дату створення розповсюджений дуже мало, а саме 0.05% усіх сайтів світу (82838 сайтів) та лише 138 в Україні станом на 4 квітня 2021 року за статистикою [2].

Робота має в першу чергу **практичне** значення, адже результатом роботи буде готовий додаток, який можна використовувати для поставлених цілей.

Метою роботи є вивчення механізмів роботи з фреймворком ASP.NET Core з мовою програмування C# при розробці веб-додатку.

Основними завданнями роботи є:

1) вивчити принципи роботи мови програмування C# та фреймворку ASP.NET Core для розробки веб-сервісу;
2) розробити зручний веб-додаток для колекціонерів монет ;
3) вивчити механізми роботи EntityFramework для роботи додатку з базою даних.

Практичне значення отриманих результатів:

Даний додаток розрахований на ведення обліку колекції власниками, дозволить розрізняти монети за різними загальними характеристиками та власними коментарями до певної одиниці колекції. Головний екран відображення колекції дозволяє краще розуміти її склад та робить процес колекціонування якіснішим, чітким та прозорим, що допомагає уникнути набір однакових монет та вказує на відсутність деяких монет відповідної колекції. Використовуючи вже існуючі шаблони найбільш розповсюджених колекцій, в яких містяться царі разом з роками їх правління, щоб швидше знаходити відповідну комірку, а також розмірності монет за їхнім номіналом, зробить заповнення та/або перенесення колекції легким та швидким. Отже,

найбільша перевага користування даним додатком - репрезентативна система відображення даних колекції, зручність редагування та присутність даної колекції завжди поруч із собою (додаток зручно виглядає і на інтерфейсі мобільного пристрою). Зручний та зрозумілий інтерфейс додатку дозволить розібратися із користуванням та освоїти колекцію дуже швидко будь-якій категорії користувача.

Наступним пунктом зручності даного додатку є можливість передачі та обміну монетами з іншими користувачами даного додатку одразу із власної колекції за допомогою розділу “аукціон”. Тобто додаток ведення колекції ще підкріплений функцією аукціону, що має можливості додавання лотів, розміщення ставок на них до закінчення часу аукціону, переведення монети з колекції одного користувача до іншого за умови вдалої угоди, що не вимагає від колекціонерів самостійного редагування колекцій.

Використане програмне забезпечення:

1) інструменти для роботи з .NET Framework, зокрема з мовою програмування C#;
2) ASP.NET Core Framework для розробки веб-застосунку;
3) Entity Framework для роботи з базою даних;
4) IDE JetBrains Rider.

Розділ 1: Аналіз предметної області. Постановка завдання курсової роботи

1.1 Аналіз сучасного стану питання

Нумізматика – історична дисципліна, яка вивчає монети, гроші, історію їх виготовлення та обігу у якості речових пам’ятках культури. Нумізмати вивчають монети не лише як частину історію, а також як засіб грошового обігу певної епохи. Через високу доступність, різноманіття та великий ціновий діапазон це дуже популярний напрям колекціонування [3].

Нумізмати дуже розрізняються за віком, сферою праці, сімейним станом та рівнем матеріального забезпечення, але усіх їх об’єднує збирання та оформлення власної колекції монет та інших грошових об’єктів, що представляють собою певну історичну цінність.

Соціологічне дослідження показало, що колекціонерам не вистачає доступної та простої системи в якій вони могли б вести облік наявних в їх колекції монет. Більшість хотіло б мати доступ до ресурсу, як з мобільного телефону, так і з персонального комп’ютера, тому вирішено було зробити такий ресурс у вигляді веб-застосунку через його універсальність. Вирішення цієї проблеми і є метою створення проекту.

Із розвитком технологічного прогресу багато звичного змінюється та зазнає процесу діджиталізації. Тож, один із найдавніших видів захоплення не став винятком та за допомогою даного додатку зазнає нових змін, що полягає у зручності ведення та обліку власної колекції, можливості завжди її мати під рукою, краще розуміти наповнення власної колекції, зручніше редагувати та одразу із власної колекції обмінюватися та/або продавати/купувати нові монети через даний ресурс, що одразу розмістить/прибере монету з колекцій

відповідно до виконаної операції. Даний симбіоз можливостей додатку робить колекції репрезентативними, зручними та спрощує колекціонеру процес ведення та обліку колекцій за допомогою зручно налаштованих функцій, зручному та приємному інтерфейсу, готовим шаблоном колекції та підв’язаним аукціоном до власних колекції користувачів.

1.2 Огляд існуючих аналогів розробки

Наразі не існує аналогів, які б могли вести облік монет в колекції, тому можно назвати дану нішу голубим океаном, що призведе до великого поширення даного додатку серед своєї аудиторії.

Існує велика кількість тематичних веб-ресурсів для колекціонерів, форумів, сайтів-аукціонів та іншого. Найпопулярніший такий веб-ресурс в Україні – Violity [4]. Крім розділу нумізматики там також присутні дуже багато інших розділів інших антикварних виробів. Користувачі можуть як виставляти на аукціон, щоб продати свою монету, так і взяти участь в лоті.

Також є дочірня платформа Violity Marketplace, де продаються товари по фіксованим цінам та тематичний форум для спілкування [5]. Існує додаток для мобільних пристроїв.

Аналіз ринку вказує на велику диференціацію форумів та аукціонів для колекціонерів, але не існують сервіси та програми для ведення своїх колекцій, що стає ресурсозатратним, вимагає багато часу та відповідного місця для обліку. Розміщення колекцій у додатку, що втілений і на персональних комп’ютерах і на мобільних пристроях, робить колекціонування ще більш приємним заняттям та дозволяє це робити зручніше та оперативніше, завжди мати змогу звернутися до колекції.

1.3 Постановка задачі

1) Провести збір інформації шляхом анкетування та опитування колекціонерів, щодо того як вони бачать такий застосунок;
2) Спроектувати базу даних;
3) Реалізація серверної частини застосунку;
4) Реалізація клієнтської частини;
5) Тестування.

Розділ 2: Теоретичні відомості

2.1 Основні відомості про розробку за допомогою ASP.NET Core

ASP.NET Core – кросплатформенний, високопродуктивний фреймворк з відкритим кодом для побудови веб-застосунків.

ASP.NET Core як основу використовує ASP.NET 4 версії з певними архітектурними змінами, що перейшли до більш модульної логіки. Але це не просто чергове оновлення, це якісно інший рівень, революція всієї платформи.

Розробка платформи почалась ще в 2014 році з назвою ASP.NET vNext. В 2016 році вийшов перший реліз, а в листопаді 2020 року вийшла версія ASP.NET Core 5.0.

ASP.NET Core може працювати з середою .NET Core, головна різниця між .NET Core та .NET Framework в тому, що вона крос-платформна та може бути розвернута на основних популярних операційних системах, таких як: Windows, MacOs, Linux.

ASP.NET Core можна охарактеризувати розширюванністю через те, що фреймворк побудований з незалежних модулів. Модулями можно легко керувати за допомогою менеджеру пакетів NuGet. Функціональність модулів ми можемо розширити за допомогою механізмів наслідування або використовувати вбудовану реалізацію.

1.4 Класи Program та Startup

1.4.1 Клас Program

Клас Program – клас, з якого починається виконання будь-якого застосунку. Стандартна реалізація класу Рисунок 0.1

```
1 using Microsoft.Extensions.Configuration;
2 using Microsoft.Extensions.Hosting;
3 using Microsoft.Extensions.Logging;
4
5 namespace TestApp
6 {
7     public class Program
8     {
9         public static void Main(string[] args)
10        {
11            CreateHostBuilder(args).Build().Run();
12        }
13
14        public static IHostBuilder CreateHostBuilder(string[] args) =>
15            Host.CreateDefaultBuilder(args)
16                .ConfigureWebHostDefaults(webBuilder =>
17                {
18                    webBuilder.UseStartup<Startup>();
19                });
20    }
21 }
```

Рисунок 0.1

Для запуску застосунку потрібен об’єкт IHost, в рамках якого і розвертається застосунок, для його створення використовується об’єкт IHostBuilder. За замовчуванням в статичному методі CreateHostBuilder створюється і налаштовується IHostBuilder. Створення за допомогою методу Host.CreateDefaultBuilder(args). Цей метод встановлює кореневу директорію, встановлює конфігурацію хосту та конфігурацію застосунку з файлу appsettings.json. Потім викликається метод ConfigureWebHostDefaults(), цей метод і забезпечує конфігурацію хоста. Останній крок – встановлення стартового класу Startup за допомогою webBuilder.UseStartup<>()

1.4.2 Клас Startup

Клас Startup – вхідна точка для застосунку, цей клас проводить першочергову конфігурацію застосунку, налаштовує сервіси, які будуть використовуватись та встановлює компоненти для обробки запитів.

Startup повинен мати метод Configure() та опціонально ConfigureServices(). При чому спочатку виконується ConfigureServices(), а потім Configure(). ConfigureServices() – реалізує систему введення залежностей, яка робить сервіси доступними по всьому застосунку. В якості параметра приймає IServiceCollection – колекцію сервісів. Для додавання нових сервісів використовується метод Add(), використання Рисунок 0.2

```
1 public void ConfigureServices(IServiceCollection services)
2 {
3     services.AddControllersWithViews();
4 }
```

Рисунок 0.2

Configure() – налаштовує поведінку обробки запиту, приймає обов’язковий параметр IApplicationBuilder – визначає клас, який забезпечує механізми налаштування конвеєра запитів програми, необов’язковий IWebHostEnvironment, який надає інформацію про середовище веб-хостингу, в якому працює програма та будь-який зареєстрований в конфігурації сервіс. За домовленістю, щоб додати компонент в конвеєр запитів використовуються методи розширення, які починаються з слова Use, наприклад щоб додати можливості маршрутизації, ми використовуємо app.UseRouting()

2.2 Паттерн MVC в ASP.NET Core

2.2.1 Основні відомості

ASP.NET Core в своїй структурі використовує розповсюдженний паттерн MVC.

Цей паттерн реалізує три головні компоненти: Model, View, Controller, з яких і формується назва паттерну.

Контролер(Controller) – клас, який забезпечує зв’язок між користувачем та системой, шаром представлення та шаром даних. В ньому реалізується бізнес-логіка, він отримує дані користувача, обробляє з огляду на задану логіку та в залежності від результату відправляє користувачу відповідь, наприклад, в вигляді представлення(View).

Модель(Model) – клас, який описує дані на шарі даних.

Представлення(View) – це елемент шару представлення, візуальна частина, яку саме і бачить користувач додатку. Частіше – html-сторінка.

Схему взаємодії можна представити так – Рисунок 0.3

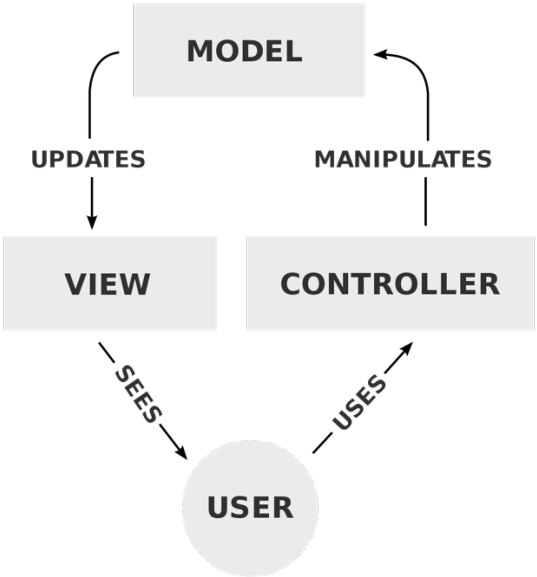

Рисунок 0.3

За допомогою цього реалізується принцип єдиної відповідальності, через це легше побудувати роботу над окремими компонентами програми та полегшується процес тестування. ASP.NET Core має досить чітку структуру, для кожного виду компонентів відведена конкретна папка, наприклад, для контролерів це папка Controllers та по аналогії для інших, в папці wwwroot зберігаються статичні файли стилів, javascript коду та інші типи статичних файлів. Базову структуру можна побачити на Рисунок 0.4

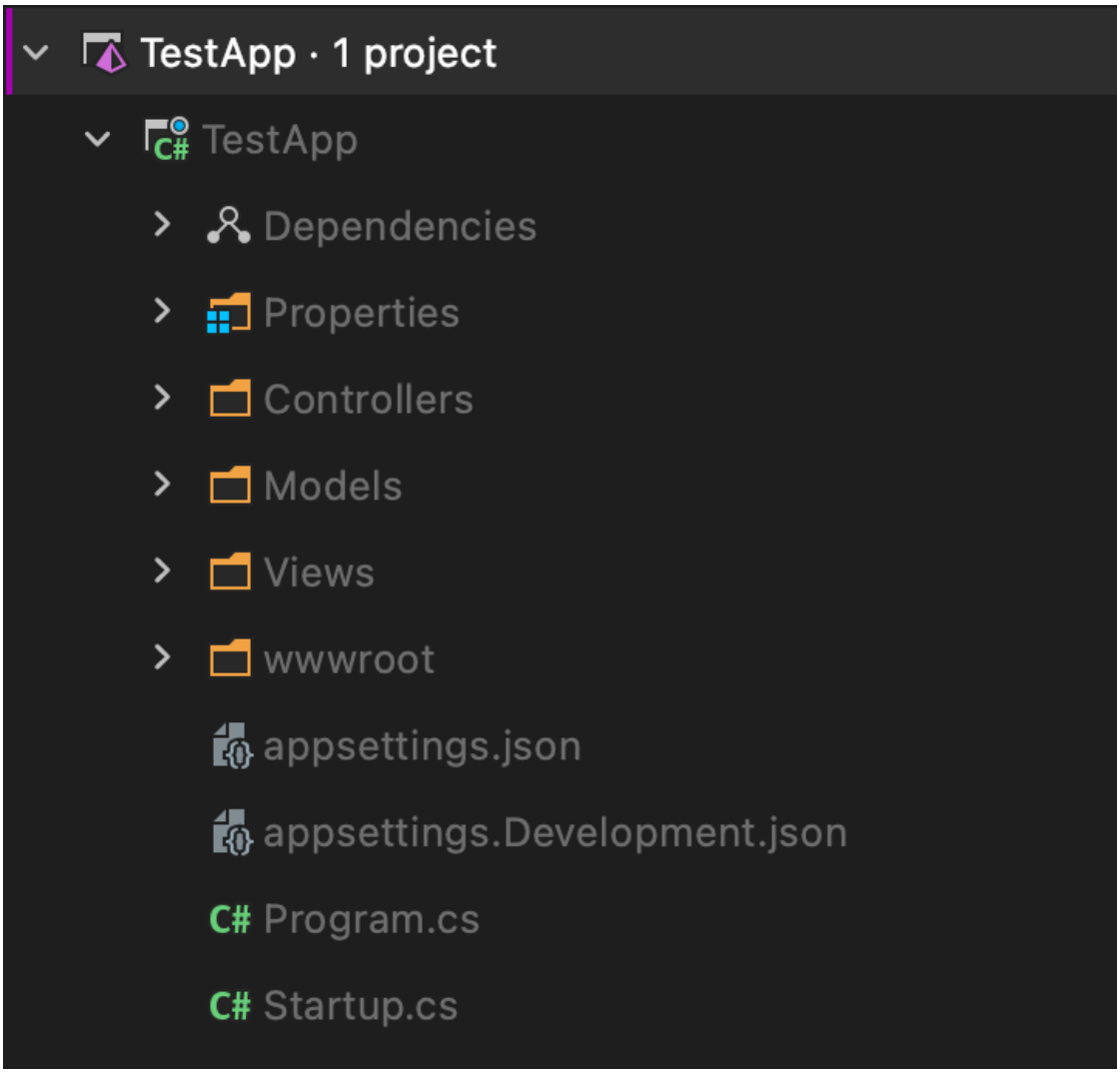

Рисунок 0.4

2.2.2 Представлення

2.2.2.1 Основні відомості про представлення

В якості сторінок представлення в ASP.NET Core використовуються побудовані за допомогою Razor сторінки, які мають розширення .cshtml. Головна їх особливість в тому, що вони дозволяють використовувати C# код всередині базової html сторінки. Це можливо тому що такі представлення компілюються в збірку, а потім використовуються для генерації html сторінок для браузера.

Для кожного контролера в папці Views створюється папка з назвою контролера. Наприклад для HomeController, папка буде називатися Home. Нижче розглянемо сторінки, які створюються за замовчуванням в директорії _Shared, в якій зберігаються шаблони та інші спільні для всіх контролерів представлення:

- _Layout.cshtml – використовуються, як шаблон для всіх інших сторінок;
- Error.cshtml – використовується для відображення помилок;
- _ValidationScripsPartial.cshtml – часткове представлення, яке підключає скріпти валідації форми.

Для рендерингу представлення в веб-сторінку використовується C# об’єкт ViewResult, приклад використання на Рисунок 0.5

```
1 public class HomeController : Controller
2 {
3     public IActionResult Index()
4     {
5         return View();
6     }
7 }
```

Рисунок 0.5

Виклик методу View() повертає об’єкт ViewResult, який передає об’єкти двигуну Razor на рендеринг відповідного представлення. Якщо викликається View() без параметрів, то відбувається пошук представлення з іменем, яке відповідає назві методу контролера в папці з назвою контролера.
Також метод має ряд перезавантажень для більш гнучкого використання.

2.2.2.2 Двигун Razor для рендерингу

Razor дозволяє використовувати стандартний код C# в представленнях.

Синтаксис використовує @ як початок коду на C#, приклад використання для відображення поточної дати на Рисунок 0.6

```
1 <h1>Дата:@DateTime.Now.ToString("dd/MM/yyyy")</h1>
```

Рисунок 0.6

Також за допомогою @ ми можемо використовувати всі стандартні конструкції мови C#, такі як: цикли, умови, свіч, try, функції або для ініціалізації змінних.

Приклад використання конструкцій на Рисунок 0.7

```
1 @{
2     int number = 10;
3 }
4 @functions{
5
6     private bool IsMoreThan5(int x){
7         if(x > 5){
8             return true;
9         }
10        return false;
11    }
12 }
13 @if(IsMoreThan5(number)){
14    <h1>@number більше ніж 5</h1>
15 }
16    else{
17    <h1>@number менше або рівне 5</h1>
18 }
19
```

Рисунок 0.7

В результаті буде відображено Рисунок 0.8

Рисунок 0.8

При використанні коду Razor за лаштунками створює класи для роботи зі сторінкою, наприклад для сторінки з таким кодом Рисунок 0.9

```
1 @functions {
2     public string GetHello()
3     {
4         return "Hello";
5     }
6 }
7
8 <div>From method: @GetHello()</div>
```

Рисунок 0.9

Буде згенеровано такий клас Рисунок 0.10

```
1 using System.Threading.Tasks;
2 using Microsoft.AspNetCore.Mvc.Razor;
3
4 public class _Views_Home_Test_cshtml : RazorPage<dynamic>
5 {
6     public string GetHello()
7     {
8         return "Hello";
9     }
10
11 #pragma warning disable 1998
12     public override async Task ExecuteAsync()
13     {
14         WriteLiteral("\r\n<div>From method: ");
15         Write(GetHello());
16         WriteLiteral("</div>\r\n");
17     }
18 #pragma warning restore 1998
```

Рисунок 0.10

Також всередині сторінок використовується модель даних, за допомогою директиви @model визначається тип моделі, що передається. Модель представляє собою звичайний клас. Члени моделі можна використовувати за зверненням @Model.Назва_члену

2.2.3 Контролер

Контролер в архітектурі є центральною фігурою. Його обирає система маршрутизації при обробленні запиту і передає йому дані запиту. Задача контролера обробити дані та на виході віддати результати.

Контролер є звичайним C# класом, який наслідується від абстрактного класу Microsoft.AspNetCore.Mvc.Controller. Звернення до контролеру через браузер відбувається через запит типу Адреса_сайту/Назва_контролеру/Назва_дії. Також контролер може приймати параметри, які вказуються в тілі запиту, наприклад Адреса_сайту/Users/Edit/2, де 2 – id користувача

2.2.4 Робота з базою даних та моделі даних.

Entity Framework Core(далі EF Core) – фреймворк для роботи з базами даних, покращена та полегшена версія Entity Framework. EF Core може служити об'єктно-реляційним відображенням (O / RM), який дозволяє розробникам .NET працювати з базою даних за допомогою об’єктів .NET та усуває необхідність у великій кількості коду для доступу до даних. EF Core підтримує низку найпоширеніших провайдерів баз даних.
Основний об’єкт з яким працює EF Core – модель. Модель – звичайний клас, який представляє програмний вигляд даних в базі даних. Є кілька методів створення моделі: вручну на основі предметної області або згенерувати з існуючої бази даних. Модель має обов’язково мати поле Id, яке буде використовувати база даних як ключ. Також в моделі можна використовувати атрибути, наприклад, для конкретизації типу даних в базі даних. Приклад класу Рисунок 0.11

```
1  using System;
2  using System.ComponentModel.DataAnnotations;
3
4  namespace TestApp.Models
5  {
6      public class User
7      {
8          public int Id { get; set; }
9          public string Name { get; set; }
10
11         [DataType(DataType.Date)]
12         public DateTime Birthdate { get; set; }
13
14     }
15 }
```

Рисунок 0.11

Для зв'язку моделей з основною логікою додатку застосовують спеціальний клас, який наслідується від класу DbContext. Контекст зберігає в собі публічні generic поля DbSet, які є колекціями моделей даних для кожної таблиці та перевантажений віртуальний метод OnConfiguring, в якому потрібно вказати яку базу даних ми будемо використовувати та строки для її під’єднання.

Для взаємодії з даними ми можемо використовувати Language Integrated Query (

LINQ). Приклад Рисунок 0.12

```
1 using (var db = new CoinsContext())
2 {
3     var coins = db.Coins
4         .Where(c => c.Year == 1699)
5         .OrderBy(c => c.Id)
6         .ToList();
7 }
```

Рисунок 0.12

Також ми можемо редагувати колекції та окремі екземпляри Рисунок 0.13.

```
 1 using (var db = new CoinsContext())
 2 {
 3     var coinForEdit = db.Coins
 4         .FirstOrDefault(c => c.Id == 1);
 5     coinForEdit.Year = 1500;
 6     db.Update(coinForEdit);
 7     var coinForRemove = db.Coins
 8         .FirstOrDefault(c => c.Id == 2);
 9     db.Coins.Remove(coinForRemove);
10
11     db.SaveChanges();
12 }
```

Рисунок 0.13

Розділ 3: Опис реалізації програмного продукту

3.1 Аналіз технічного завдання

Для реалізації цілі було вирішено зробити додаток, що зручно і наглядно відображає дані колекції як і з великого монітору персонального комп’ютеру, так і з екрану мобільного телефону з метою швидкого доступу до своєї колекції та проглядання її в будь-який час. Через вибір реалізації завдання в вигляді веб-застосунку його можна використовувати на будь-якому девайсі з веб-браузером та підключенням до мережі інтернет. Користувач має можливість через зручну форму вносити свої монети, вказуючи певні характеристики, маючи можливість редагування. Також користувач має змогу продати свою монету або поповнити свою колекцію за допомогою системи аукціону. Після вдалого обміну монета автоматично додається до колекції.

3.2 Обґрунтування алгоритму та структури програми

Для розділення монет та лотів між користувачами було вирішено зробити систему авторизації та реєстрації. Управління цими процесами винесено в окремий контролер.

Наш веб-застосунок має два основні розділи: колекція та аукціон. Доцільно також розділити логіку кожного розділу на кілька контролерів.

Управління запитами колекції відбувається в окремому контролері. В залежності від вхідного запиту контролер визначає, яке представлення має бути повернуте. В розділі колекції є три різні розділи видів монет по певним критеріям.

Для можливої розширюваності та зручності системи процеси підрахунку кількості монет в колекції було винесено в окремі класи CollectionCount та його нащадок CollectionRegionalCount для розділу з регіональними монетами. В подальшому систему буде легко розширювати за допомогою механізму наслідування.

Управління аукціоном також реалізовано в окремому контролері, який контролює всю роботу з аукціоном.

Також було вирішено розробити API для зв'язку клієнтської частини з серверною, де потрібно динамічно та швидко змінювати дані без перезавантаження сторінки.

3.3 Обґрунтування вибору засобів розробки

Темою даної роботи є розробка застосунку за допомогою фреймворку ASP.NET Core, тому мовою програмування для розробки була обрана мова C#. C# об’єктно-орієнтована мова програмування та для реалізації цього завдання підходить найкраще.

Системою управління базами даних було обрано PostgreSql через її швидкість та інтеграцію з EntityFramework Core через спеціальний пакет Npgsql.EntityFrameworkCore.PostgreSQL.

Для роботи з базами даних було обрано EntityFramework Core через його швидкість та простий і зрозумілий механізм роботи.

Для взаємодії з сервісом зберігання зображень монет було використано пакет CloudinaryDotNet.

Для розробки користувацького інтерфейсу було використані такі інструменти як html, css та javascript разом зі встановленою бібліотекою JQuery.

Git було обрано як систему контроля версій.

3.4 Опис розробки програми

3.4.1 Розробка механізму автентифікації

Для розподілення відповідальності було вирішено розробити кілька ролей для користувачів. В нашому випадку це – user та admin.

User – звичайний користувач системи, який має можливість створювати та управляти своїми монетами, виставляти лоти, робити ставки.

Admin – має ті ж можливості що і User з додаванням нових можливостей, а саме: керувати наявними категоріями монет(номіналами, країнами та царями), керувати користувачами.

Для реалізації механізму роботи з ролями та аутентифікацією було використано вбудовану в ASP.NET Core систему аутентифікації та авторизації на основі cookie. Взаємодія відбувається через об’єкт, який відповідає інтерфейсу IPrincipal – HttpContext.User. За допомогою цього ми можемо перевіряти в якій ролі знаходиться користувач та взаємодіяти зі списком його Claims, які ми задаємо при його авторизації.

Метод аутентифікації виглядає наступним чином Рисунок 0.1

```
1  private async Task Authenticate(Users users)
2          {
3              var claims = new List<Claim>
4              {
5                  new(ClaimsIdentity.DefaultNameClaimType, users.Login),
6                  new(ClaimsIdentity.DefaultRoleClaimType, users.Role.Name),
7                  new("id", users.Id.ToString())
8              };
9              ClaimsIdentity id = new ClaimsIdentity(claims, "ApplicationCookie",
   ClaimsIdentity.DefaultNameClaimType,
10                 ClaimsIdentity.DefaultRoleClaimType);
11             await
   HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new
   ClaimsPrincipal(id));
12         }
```

Рисунок 0.1

3.4.2 Розробка взаємодії з колекцією

Взаємодія з колекцією відбувається в контролері CollectionController.

Для прискорення процесу підрахунку кількості монет в класах сімейства CollectionCount було використано мультипотоковий метод. Для кожної окремої категорії створюється окремий потік, який записує результуючі дані в масив, метод Count() повертає результуючу модель даних для представлення.

Взаємодію при запиті колекції ми можемо представити таким чином Схема 0.1

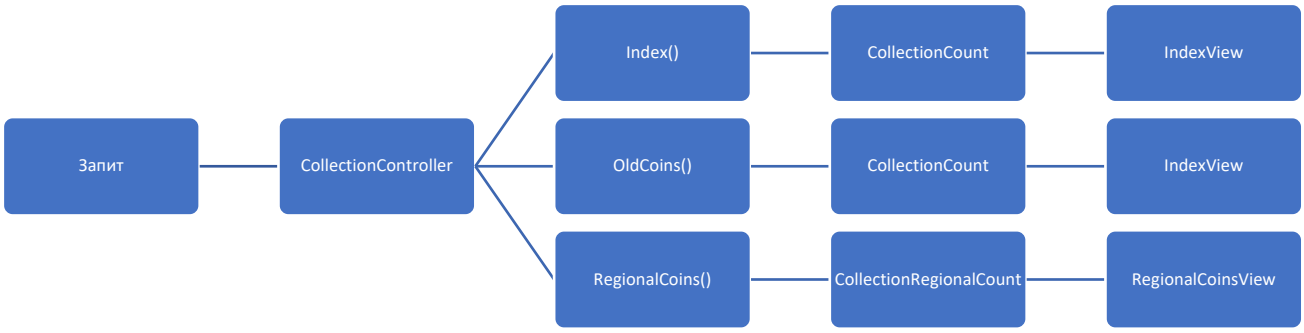

Схема 0.1

Також користувач має можливість переглянути конкретні монети з певних категорій, для цього він може натиснути на кількість монет в представленні або на назву категорії. Схему взаємодії можемо представити наступним чином Схема 0.2

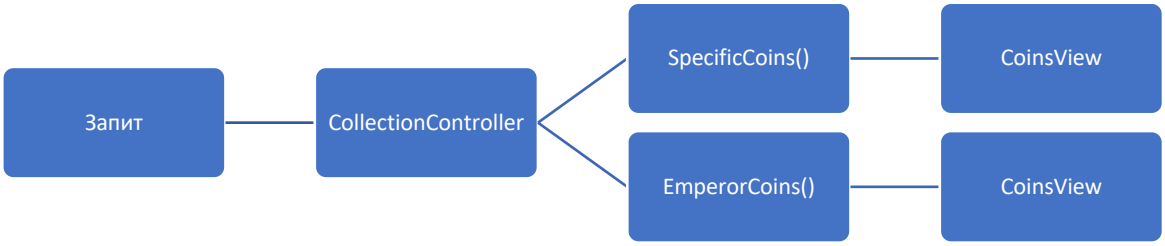

Схема 0.2

Реалізацію класу підрахунку монет можна побачити в Додаток А.

3.4.3 Розробка взаємодії з аукціоном

Аукціон складається з лотів, у кожного з яких можуть бути коментарі та ставки. Логіка реалізована в відповідному контролері AuctionController. Вся взаємодія на сторінці лоту реалізована без перезавантаження сторінки та динамічною зміною контенту за допомогою .ajax запитів.

Будь-який користувач може залишити коментар на сторінці лоту та оцінити інші коментарі. Система рейтингів та додавання коментаря реалізовані через окремий стовпець в таблиці коментарів та зміну його за допомогою API функції Рисунок 0.2.

```
1        [HttpPost("add")]
2        public ActionResult AddComment([FromBody] CommentModel model)
3        {
4            if (_db.Lots.FirstOrDefault(l => l.Id == model.LotId) != null)
5            {
6                LotComment comment = new LotComment();
7                comment.LotId = model.LotId;
8                comment.Rating = 0;
9                comment.UserId = model.UserId;
10               comment.Text = model.Text;
11               comment.DateTime = DateTime.Now;
12               _db.LotComments.Add(comment);
13               _db.SaveChanges();
14               return Ok();
15           }
16
17           return BadRequest();
18       }
19
20       [HttpPost("rating/add/{commentId}")]
21       public ActionResult AddRating(int commentId)
22       {
23           LotComment comment = _db.LotComments.FirstOrDefault(c => c.Id ==
commentId);
24           if (comment != null)
25           {
26               comment.Rating = comment.Rating + 1;
27               _db.Update(comment);
28               _db.SaveChanges();
29               return Ok();
30           }
31
32           return BadRequest();
33       }
```

Рисунок 0.2

Ставку може зробити будь-який користувач, за виключенням власнику лоту та користувача, який зробив останню ставку. Також реалізовано за допомогою взаємодії з API серверної частини та клієнтських функцій JQuery Рисунок 0.3.

```
1         [HttpPost("add")]
2         public ActionResult AddBid([FromBody] BidModel model)
3         {
4             AuctionLot lot = _db.Lots.FirstOrDefault(b => b.Id == model.LotId);
5             if (lot != null)
6             {
7                 if (model.Price <= lot.CurrentPrice || DateTime.Now > lot.EndTime)
8                 {
9                     return Problem();
10                }
11
12                AuctionBid bid = new AuctionBid {Price = model.Price, LotId =
    model.LotId,
13                                                 UserId = model.UserId, DateTime =
    DateTime.Now};
14                lot.Bids.Add(bid);
15                lot.CurrentPrice = model.Price;
16                _db.SaveChanges();
17
18                return Ok();
19            }
20
21            return BadRequest();
22        }
```

Рисунок 0.3

На сторінці лоту також реалізований таймер для відслідковування кількості часу, що залишилося до кінця Рисунок 0.4.

```
1 let timeLeft = $("#time-left")
2                 if(timeLeft.text() != "Аукціон завершено"){
3                         let date = new Date("@Model.Lot.EndTime.ToString("yyyy-MM-
  ddThh:mm:ssZ")")
4                         let x = setInterval(function() {
5
6                             let now = new Date().getTime();
7
8                             let distance = date - now;
9
10                            let days = Math.floor(distance / (1000 * 60 * 60 * 24));
11                            let hours = Math.floor((distance % (1000 * 60 * 60 * 24)) /
  (1000 * 60 * 60));
12                            let minutes = Math.floor((distance % (1000 * 60 * 60)) /
  (1000 * 60));
13                            let seconds = Math.floor((distance % (1000 * 60)) / 1000);
14                            let dayName
15                            let mod10Days = days % 10;
16                            if (mod10Days == 1){
17                                dayName = " день"
18                                }
19                            else if (mod10Days == 2 || mod10Days == 3 || mod10Days ==
  4){
20                                 dayName = " дні"
21                                }
22                            else{
23                                dayName = " днів"
24                                }
25
26                            timeLeft.text(days + dayName + " " + hours + ":"
27                                                                  + minutes + ":" +
  seconds);
28
29                            if (distance < 0) {
30                                clearInterval(x);
31                                timeLeft.text("Аукціон завершено");
32                            }
33                        }, 1000);
34                    }
```

Рисунок 0.4

Користувач має можливість переглядати всі активні лоти та шукати потрібні по ключовим словам. Також всі лоти в яких користувач бере участь він може

переглянути в розділах особистого кабінету. За відображення всіх лотів відповідає метод контролеру Index(), а за відображення лотів користувача – UserLots(). Обидва методи повертають сторінку Index з моделлю представлення LotsViewModel. За отримання інформації про лоти користувача в яких він бере участь відповідає метод UserBids(), який повертає представлення з трьома різними типами лотів користувача: закінчені виграшні, закінчені програшні та активні.

Для перевірки активності лотів в паралельному потоці постійно перевіряються чи пройшов час закінчення активних лотів Рисунок 0.5.

```
1  private void CheckOverDueLots()
2          {
3              if (_actualLots == null)
4              {
5                  return;
6              }
7              while (true)
8              {
9                  lock (_actualLots)
10                 {
11                     if (_actualLots.Count != 0)
12                     {
13                         List<int> lotsIdsToRemove = new List<int>();
14                         foreach (var lot in _actualLots)
15                         {
16                             if (lot.EndTime <= DateTime.Now)
17                             {
18                                 lot.IsEnded = true;
19                                 lock (_locker)
20                                 {
21                                     using (var db = new AuctionContext())
22                                     {
23                                         db.Update(lot);
24                                         db.SaveChanges();
25                                     }
26                                 }
27                                 lotsIdsToRemove.Add(lot.Id);
28                             }
29                         }
30
31                         _actualLots.RemoveAll(l => lotsIdsToRemove.Contains(l.Id));
32                     }
33                 }
34
35                 Thread.Sleep(100000);
36             }
37         }
```

Рисунок 0.5

3.5 Опис файлів даних та інтерфейсу програми

3.5.1 База даних

В реалізації проекту використана така структура бази даних Схема 0.3

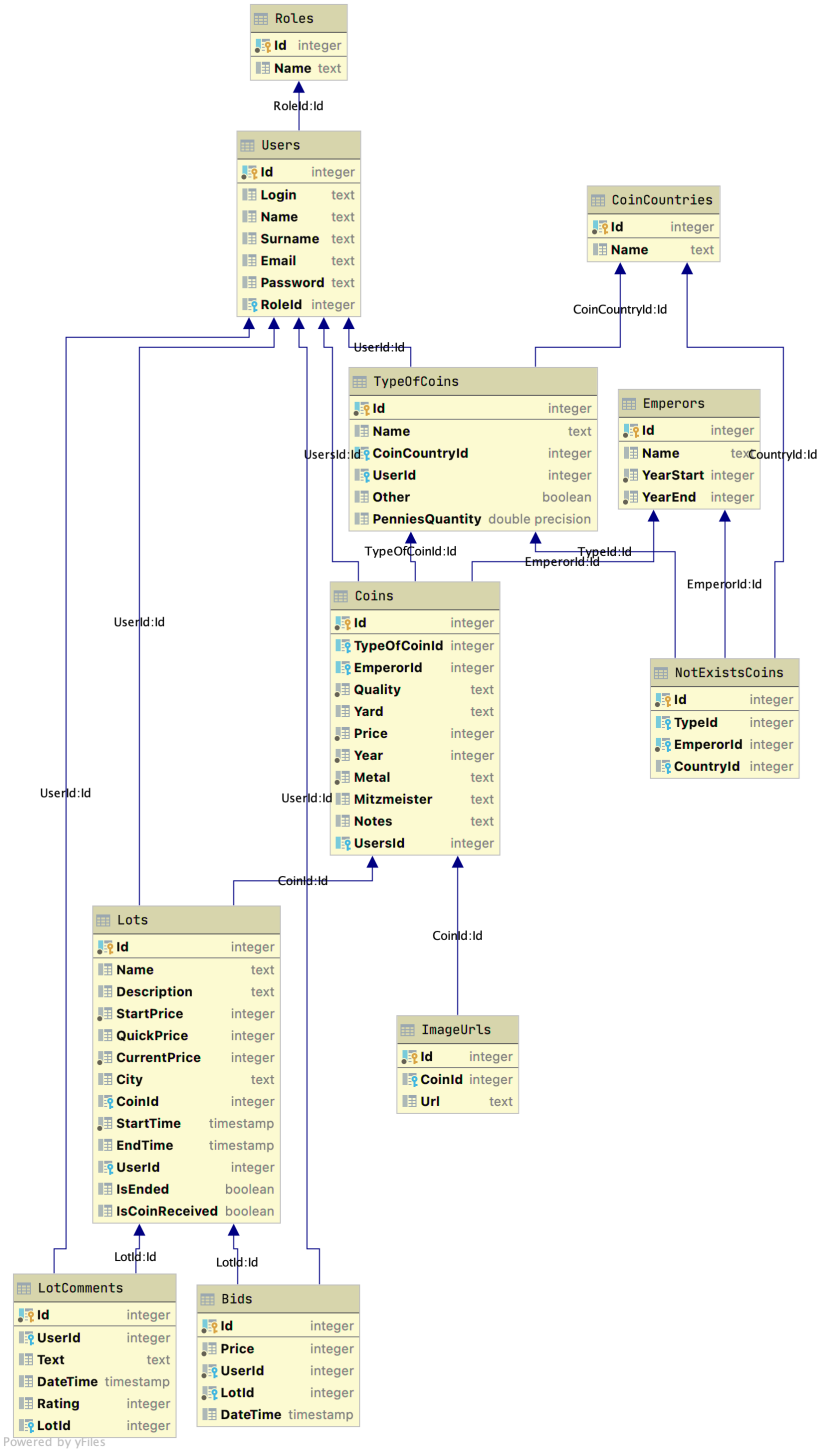

Схема 0.3

Таблиця Users відповідає за зареєстрованих користувачів системи, також використовується для ідентифікації монет, номіналів, лотів, ставок, коментарів.

Таблиця Roles відповідає за наявні ролі для користувачів.

Таблиця Coins відповідає за наявні в системі монети, має різні характеристики, включаючи номінал, царя та користувача. Також використовується в таблицях лотів та зображень монет.

Таблиця TypeOfCoins відповідає за номінали в системі, можуть бути як за замовчуванням, так і користувацькими.

Таблиця CoinCountries відповідає за наявні в системі країни регіонального випуску монет.

Таблиця Emperors відповідає за наявних царів в системі.

Таблиця NotExistsCoins відповідає за неіснуючі типи монет по номіналу та царю або по країні на царю.

Таблиця ImageUrls відповідає за збереження інформації про посилання на зображення монети в сервісі Cloudinary.

Таблиця Lots відповідає за інформацію про лоти.

Таблиця Bids та LotComments зберігають в собі інформацію про ставки та коментарі відповідно, не можуть існувати без лоту.

3.5.2 Користувацький інтерфейс

При відкритті даного додатку користувач потрапляє на головну сторінку додатку для обліку колекції, що вітає та запрошує ознайомитися з головними функціями продукту. Спершу додаток перевірить наявність акаунту та у разі його відсутності запропонує зареєструвати нового користувача(Рисунок 0.6).

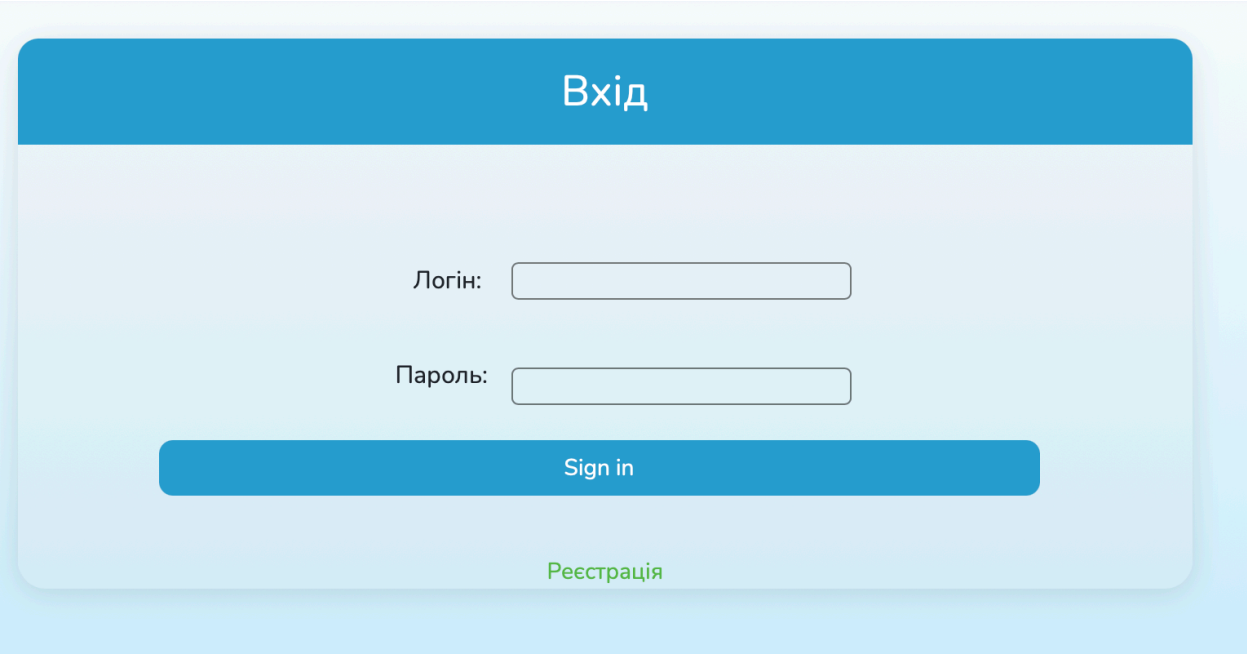

Рисунок 0.6

Інтерфейс виконано за допомогою використання спокійних кольорових поєднань через вертикальний градієнт фону білого та аквамарину кольорів. Через білий колір відображено лаконічність та мінімалізм, а даний відтінок голубого кольору використано через його властивості спокою та впевненості, що використовуються в маркетингу для залучення користувачів. Оскільки під опис портрету споживача підпадають люди середнього віку із середнім та вище рівнем матеріального достатку, разом із візуальним сприйняттям таких відтінків кольорів вони відчуватимуть впевненість у веденні колекції та довірятимуть продукту [].

Для зручності владинка “Особистий кабінет”(Рисунок 0.7) розташована зверху справа та інтуїтивно її легко можна знайти. В даному розділі користувач, який ще не ввійшов в систему має змогу це зробити, а діючий

користувач, може переглядати свої лоти, виставлені на аукціон, ставки, зроблені на ньому та вийти з системи.

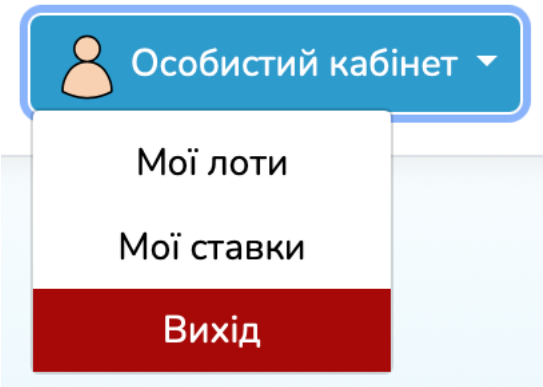

Рисунок 0.7

Додавання колекціонером нових монети до колекції відбувається за допомогою створеної форми, що складається з розділів(Рисунок 0.8):

- Номінал монети (за допомогою випадаючого списку можна обрати тип монети);
- Царь (за час правління якого було створено та випущено монету в обіг, також можна вибрати за допомогою випадаючого списку);
- Рік (виготовлення та випуску в обіг монети; з усіх можливих років додаток підбере та запропонує роки, що відповідають часам правління відповідно обраного Царя);
- Монетний двір;
- Мінцмейстер;
- Якість (за допомогою випадаючого списку можна обрати якість за системою оцінювання якості монет);
- Метал (за допомогою випадаючого списку можна вибрати матеріал виготовлення монети);
- Ціна (колекціонер вказує ціну покупки одиниці колекції, щоб мати можливість оцінити загальну вартість всієї колекції та знати, за якою ціною була виконана угода покупки);

- Замітки (в даному полі користувач може додати певні нотатки стосовно монети, які детальніше опишуть унікальність монети за певними якостями та властивостями монети);
- Фото (можна завантажити фотокартку монети, що буде відображатися у детальній інформації про неї та під час виставлення монети на аукціон).

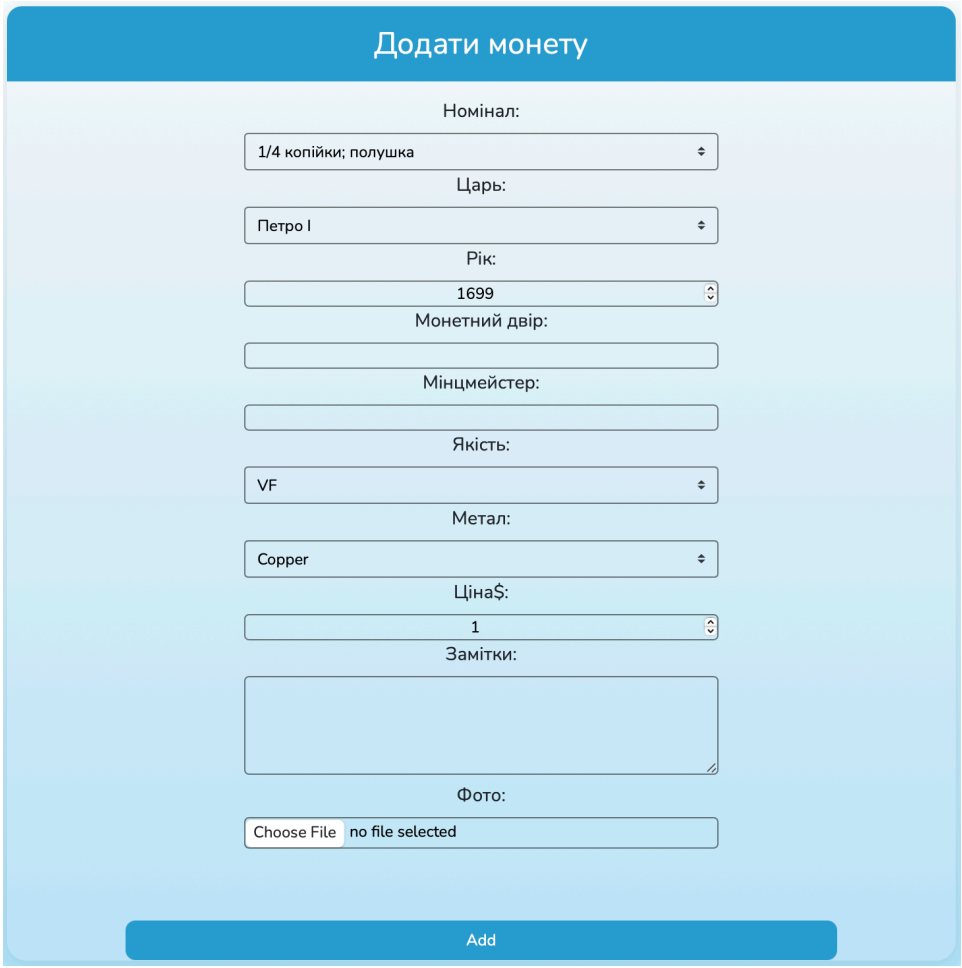

Рисунок 0.8

За допомогою навігаційного меню, що розташоване зверху, можна шляхом обрання кнопки “Моя колекція” перейти до таблиці, де представлені усі можливі монети. Кнопка “Моя колекція” має три розділи: “Регулярний чекан”, “Нерегулярний чекан” та “Регіональний чекан”, що за своєю суттю представляють три різні колекції за наповненням та направленням Рисунок 0.9.

Рисунок 0.9

При переході до перегляду колекції, як вже зазначалося, з’являється таблиця колекції з усіма можливими видами монет. Зверху над нею велика добре проглядаєма кнопка додавання монет, функціонал якої був описаний вище. В колонтитулі таблиці (шапці) перелічені усі номінали монет, а в боковику таблиці вказані усі царі з їхніми іменами та роками правління у дужках. Такі головні елементи виділені темним кольором для контрасту та зазначення “шапки” таблиці. В умовах переглядання даної таблиці виконано закріплення колонтитулів та боковику таблиці, щоб було легше переглядати таблицю та ознайомлюватися з нею. Для зручного перегляду об’ємної таблиці при наведенні на прографку горизонтальною та вертикальною чорними лініями виділяються номінал монети, цар, за часи правління якого вона була емісійована, та власне сама монета та її кількість у колекції власника. У прографках вказано кількість монет одного виду. За допомогою кольорів зручніше орієнтуватися у таблиці. Червоний колір вказує на відсутність монет у колекції власника, та грунтуючись на цих даних, колекціонер якісніше збере всю колекцію, адже знатиме, чого не вистачає. Зелений колір

вказує на присутність видів монет у колекції та вказана їхня кількість. Комірки, що не мають ніякого забарвлення окрім фону самого інтерфейсу, означають, що даного виду монети не існувало. Останні стовпчик та рядок таблиці рахують загальну кількість монет за їхнім номіналом та царем з роками емісії відповідно. Таблиця підбивається останньою строкою, що рахує загальну вартість та грошову цінність колекції шляхом сумування всіх введених цін купівлі монет. Рисунок 0.10

Додати монету

Царь/Номінал	1/4 копійки; полушка	1/2 копійки; деньга; денежка	1 копійка	2 копійки	3 копійки; алтин	4 копійки	5 копійок	10 копійок; гривенник	15 копійок	20 копійок	25 копійок; полуполтина	50 копійок; полтина	1 рубль	Інші	Кількість
Петро I (1699 - 1725)	8	6	6		1		1	0			0	0	0	0	22
Катерина I (1725 - 1727)			0				2	0				0	0	0	2
Петро II (1727 - 1730)			2				0					0	0	0	2
Анна (1730 - 1740)	5	10					1	0			0	0	1	0	17
Іоан (1740 - 1741)	0	1						0				0	0	0	1
Єлизавета (1741 - 1762)	0	7	4	9			7	2			1	0	1	0	31
Петро III (1762 - 1762)		0	0	1		2		0				0	0	0	3
Катерина II (1762 - 1796)	3	2	3	6		0	11	2	1	1	0	0	2	0	31
Павло I (1796 - 1801)	2	1	2	5			1	0			0	0	1	0	12
Олександр I (1801 - 1825)	0	3	3	8			3	3		2	0	1	4	0	27
Микола I (1826 - 1855)	3	4	7	4	2		7	4		1	3	3	3	0	41
Олександр II (1855 - 1881)	2	2	3	1	3		6	1	1	1	2	2	1	0	25
Олександр III (1881 - 1894)	0	2	2	1	1		1	1	1	1	1	1	3	0	15
Микола II (1894 - 1917)	1	2	2	2	3		5	2	1	4	2	1	2	3	30
Кількість	24	40	34	37	10	2	45	15	4	10	9	8	18	3	259

Загальна вартість: 1241₽

Рисунок 0.10

З мобільного пристрою, щоб таблиця виглядала зручною, було вирішено зменшити кількість даних одночасно, тому перехід до конкретного списку монет проходить в кілька етапів. Перший – відображення царів та наявна кількість монет кожного. Другий – при натисканні на ім’я царя при наявності монет відкривається вікно з існуючими номіналами. Рисунок 0.11

Рисунок 0.11

При нажатті на відповідний вид монети можна переглянути кількість та характеристики існуючих монет у новому вікні. На цій сторінці відображатимуться дані у вигляді таблиці за певними видами інформації, що були заповнені власником під час додавання нових елементів колекції. Назва таблиці містить інформацію про номінал монети та царя з його роками правління, коли дана монета могла бути випущена в обіг. В даній таблиці є 3 круглі кнопки, виконані в однаковому стилі, які вносять зміни до цієї таблиці та власне в основну таблицю колекції. Рисунок 0.12

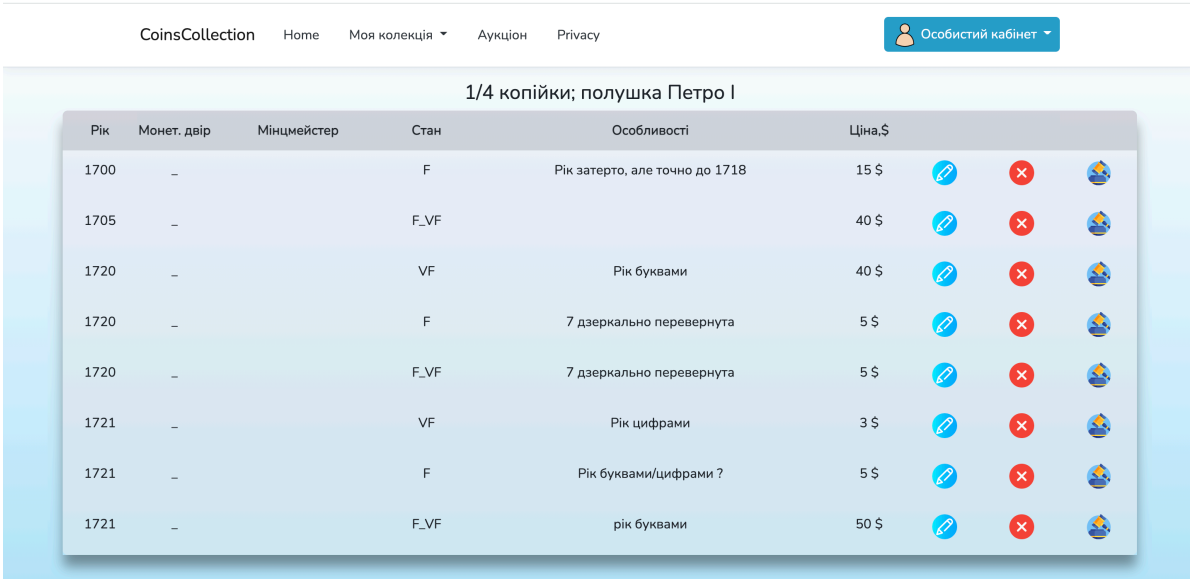

Рисунок 0.12

За допомогою кнопки виконаної у блакитному кольорі із зображеним на ній олівцем, нескладно інтуїтивно здогадатись, що її функція редагувати дані про монету. При натисканні кнопки з’являється новий рядок таблиці в якому знаходиться інформація для редагування. Рисунок 0.13

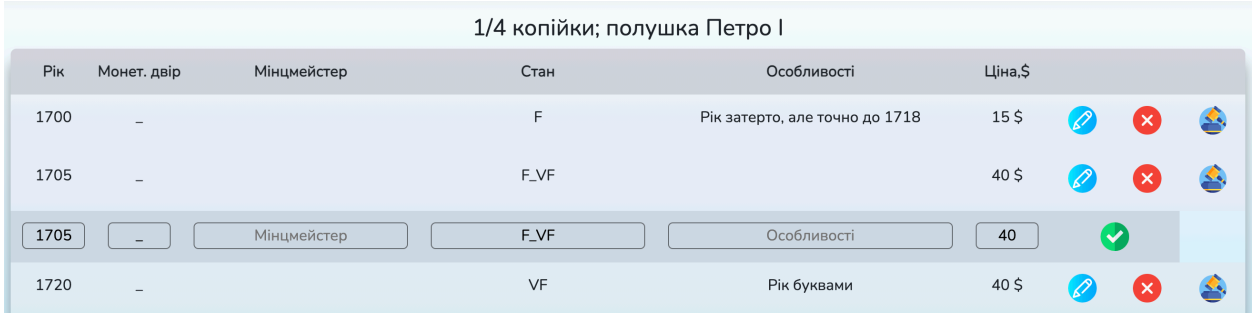

Рисунок 0.13

За допомогою червоної кнопки, в всередині якої зображений білий хрестик, можна видалити певну монету з колекції. За допомогою кнопки, на якій зображений молоток для аукціону, можна винести певну монету на аукціон одразу із своєї колекції. При нажатті на дану кнопку з’являється форма заповнення інформацію про монету із вже заповненими графами, що вже заповнювались під час додавання монет у колекцію. Також присутні вже нові графи, що потребують заповнення, такі як:

- Початкова ціна (треба заповнити ціну, з якої почнеться аукціон);

- Місцезнаходження (заповнити місце щоб учасники аукціону розуміли, де відбувається продаж);
- Дата закінчення (налаштовує ініціатор аукціону для закінчення проведення торгів). Рисунок 0.14**Error! Reference source not found.**

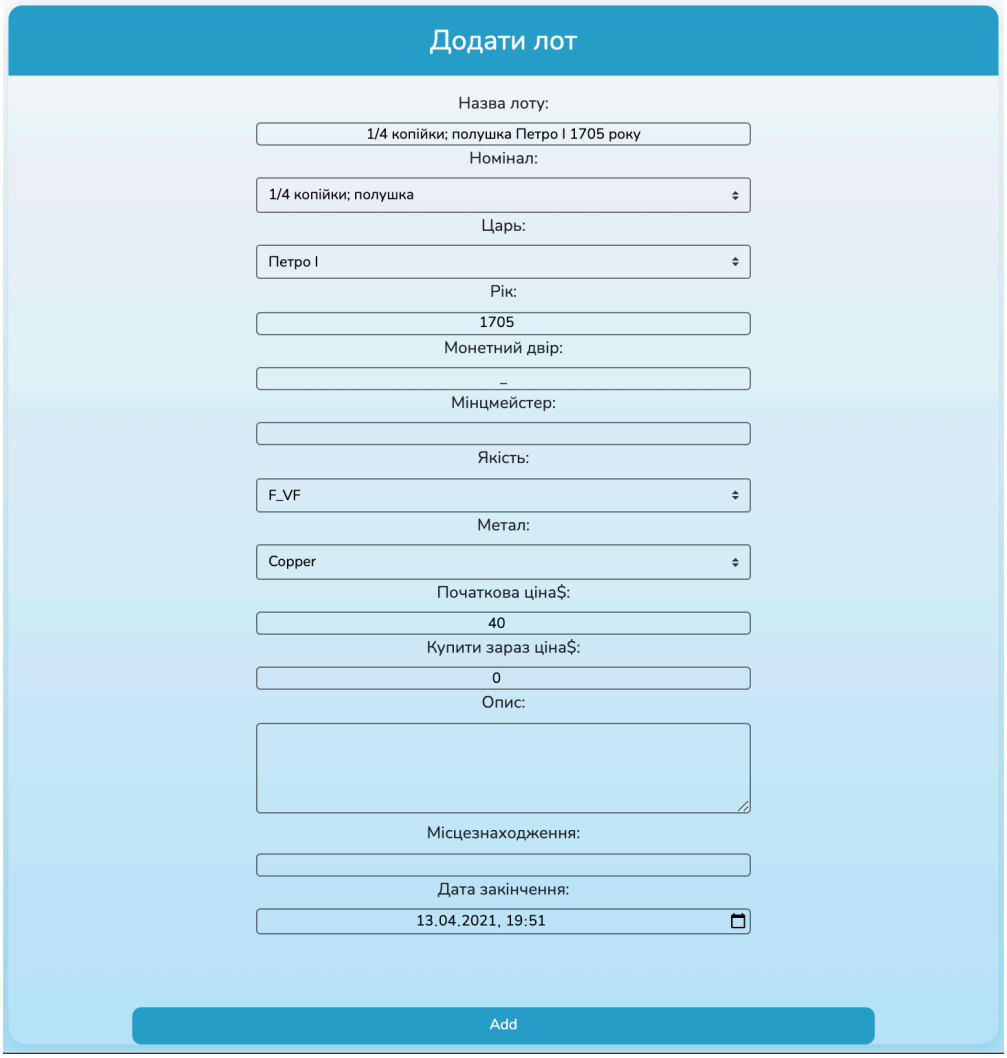

Рисунок 0.14

Через головне меню за допомогою кнопки «Аукціон» або «Мої лоти» можна потрапити до перегляду лотів. За допомогою пошукового поля можна швидко знайти монету за ключовими словами, що цікавить більш всього. За допомогою активації кнопки “Лише активні” можна прибрати лоти, продажі яких завершені по закінченню сроку торгів. На сторінці аукціону відображені лоти з їхньою фотокаркою, номіналом монети, рік та цар,

початкова ціна та розмір останньої ставки. У разі відсутності зображення лоту відображається картинка із зображенням монети. Рисунок 0.15

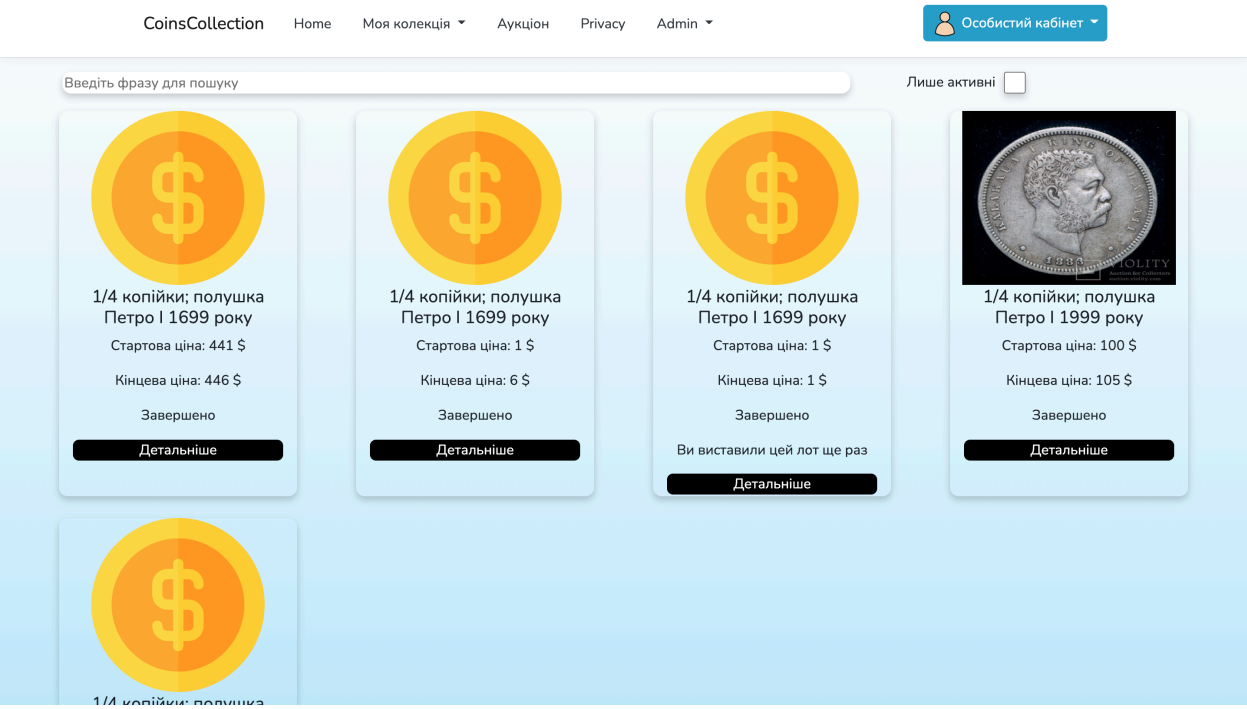

Рисунок 0.15

Висновок

Під час збору інформації, її обробки та вивчення було прийнято рішення розробити зручний додаток для колекціонерів-нумізматів для простого ведення обліку власних колекції з метою постійного доступу до неї та можливості переглядати її, змінювати інформацію про окремі одиниці колекції, видаляти та додавати нові елементи, а також приймати участь в аукціонах з метою продажу та/або купівлі елементів колекцій.

В результаті роботи мета була досягнута. Було детально розглянуто технологію ASP.NET Core для розробки веб-застосунків. Також був реалізований лаконічний, зручний та інтуїтивно зрозумілий користувацький інтерфейс для демонстративного вигляду даного додатку та зручності користування ним.

Список використаних джерел

1. Офіційний сайт Microsoft, .NET. [Електронний ресурс]. Режим доступу: https://dotnet.microsoft.com
2. [Електронний ресурс]. Режим доступу: https://trends.builtwith.com/framework/traffic/Entire-Internet
3. Енциклопедія історії України. [Електронний ресурс]. Режим доступу: http://resource.history.org.ua/cgi-bin/eiu/history.exe?&I21DBN=EIU&P21DBN=EIU&S21STN=1&S21REF=10&S21FMT=eiu_all&C21COM=S&S21CNR=20&S21P01=0&S21P02=0&S21P03=TRN=&S21COLORTERMS=0&S21STR=Numizmatyka
4. Офіційний сайт Аукціону «Violity». Електронний ресурс]. Режим доступу: https://auction.violity.com
5. Офіційний сайт Маркетплейсу «Violity». Електронний ресурс]. Режим доступу: https://violity.com
6. Phil Barden, Decoded: The Science Behind Why We Buy (First edition). John Wiley & Sons, 2013, 288pp. (ISBN-10: 1118345606, ISBN-13: 978-1118345603)

Додаток А. Клас CollectionCount

```
public class CollectionCount
{
    protected readonly List<Emperor> _emperors;
    protected readonly List<TypeOfCoin> _types;
    protected readonly List<Coin> _userCoins;
    protected readonly List<NotExistsCoin> _notExistsCoins;
    protected int[][] _counts;

    public CollectionCount(List<Emperor> emperors, List<TypeOfCoin> types,
List<Coin> userCoins,
        List<NotExistsCoin> notExistsCoins)
    {
        _emperors = emperors;
        _types = types;
        _userCoins = userCoins;
        _notExistsCoins = notExistsCoins;
    }

    public CollectionModel Count(string deviceType)
    {
        CollectionModel viewModel = new CollectionModel
            {Emperors = _emperors, Types = _types.Where(t => !t.Other).ToList(),
DeviceType = deviceType};
        if (_userCoins.Any())
        {
            _counts = new int[_emperors.Count + 1][];
            int i = 0;
            List<Thread> threads = new List<Thread>();

            foreach (var e in _emperors)
            {
                var coins = _userCoins
                    .Where(c => c.EmperorId == e.Id).ToList();
                var tmp = i;
                ThreadCountModel model = new ThreadCountModel
                    {I = tmp, Coins = coins, Types = _types, EmperorId = e.Id};
                Thread thread = new Thread(new
ParameterizedThreadStart(ThreadCount));
                thread.Name = $"Thread-{i}";
                threads.Add(thread);
                thread.Start(model);
```

```
            i++;
        }

        if (threads.Count != 0)
        {
            foreach (var t in threads)
            {
                t.Join();
            }
        }

        lock (_counts)
        {
            viewModel = new CollectionModel
            {
                Emperors = _emperors, Types = _types.Where(t => !t.Other).ToList(),
                Price = _userCoins.Sum(c => c.Price), DeviceType = deviceType
            };
            List<int> countTypes = new List<int>();
            int otherCounts = 0;
            foreach (var t in _types)
            {
                if (!t.Other)
                {
                    countTypes.Add(_userCoins.Count(c => c.TypeOfCoinId == t.Id));
                }
                else
                {
                    otherCounts += _userCoins.Count(c => c.TypeOfCoinId == t.Id);
                }
            }

            countTypes.Add(otherCounts);
            countTypes.Add(_userCoins.Count);

            _counts[_emperors.Count] = countTypes.ToArray();
            viewModel.Counts = _counts;
        }
    }
    else
    {
```

```
        _counts = new int[_emperors.Count + 1][];
        for (int i = 0; i <= _emperors.Count; i++)
        {
            int[] temp = new int[viewModel.Types.Count + 2];
            for (int j = 0; j <= viewModel.Types.Count + 1; j++)
            {
                temp[j] = 0;
            }

            _counts[i] = temp;
        }

        viewModel.Counts = _counts;
        viewModel.Price = 0;
    }

    return viewModel;
}

protected virtual void ThreadCount(object x)
{
    ThreadCountModel model = (ThreadCountModel) x;
    List<int> count = new List<int>();
    int otherCounts = 0;
    var emperorId = model.EmperorId;

    foreach (var type in model.Types)
    {
        bool exist = true;
        lock (_notExistsCoins)
        {
            exist = _notExistsCoins.Exists(c =>
                c.EmperorId == emperorId && c.TypeId == type.Id);
        }

        if (!exist)
        {
            if (!type.Other)
            {
                count.Add(model.Coins.Count(c => c.TypeOfCoinId == type.Id));
            }
            else
            {
```

```
                otherCounts += model.Coins.Count(c => c.TypeOfCoinId == type.Id);
            }
        }
        else
        {
            count.Add(-1);
        }
    }

    count.Add(otherCounts);
    count.Add(model.Coins.Count);
    lock (_counts)
    {
        _counts[model.I] = count.ToArray();
    }
  }
}
```