Министерство просвещения ПМР ГОУ СПО

«Тираспольский техникум информатики и права»

**КУРСОВАЯ РАБОТА**

По дисциплине: «Разработка программных модулей»

На тему: «Разработка программной системы для управления художественной галереей»

Выполнил обучающийся Шеленков Кирилл Вячеславович

специальность 09.02.07 «Информационные системы и программирование»

Руководитель преподаватель

Подсекина Татьяна Сергеевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись руководителя)

Тирасполь 2025

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc198752481)

[ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ОБЩАЯ ПОСТАНОВКА ЗАДАЧ 5](#_Toc198752482)

[1.1. Актуальность и значение информационных систем в культуре 5](#_Toc198752483)

[1.2. Анализ форматов и технологий хранения данных 6](#_Toc198752484)

[1.3. Обзор аналогов и сравнительный анализ 9](#_Toc198752485)

[1.4. Постановка задачи и функциональные требования 11](#_Toc198752486)

[ГЛАВА 2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ УПРАВЛЕНИЯ ХУДОЖЕСТВЕННОЙ ГАЛЕРЕЕЙ 12](#_Toc198752487)

[2.1. Структура программного продукта 12](#_Toc198752488)

[2.2. Архитектура и структура JSON-хранилища 13](#_Toc198752489)

[2.3. Реализация интерфейса и ключевых операций 17](#_Toc198752490)

[2.4. Тестирование и примеры работы 19](#_Toc198752491)

[ЗАКЛЮЧЕНИЕ 22](#_Toc198752492)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](#_Toc198752493)

[ПРИЛОЖЕНИЕ 1 24](#_Toc198752494)

# ВВЕДЕНИЕ

Актуальность выбранной темы обусловлена необходимостью цифровизации процессов учета и управления культурной инфраструктурой. В современных условиях художественные галереи сталкиваются с ростом количества экспонатов, посетителей и разнообразием форм работы (продажа, аренда, временные выставки). Отсутствие единой автоматизированной системы ведет к увеличению ошибок в документообороте, трудоемкости операций и потере данных.



Целью курсовой работы является разработка консольной программной системы, обеспечивающей полный цикл управления художественной галереей: учет экспонатов, управление художниками, планирование и проведение выставок, бронирование билетов, оформление продаж и аренды.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Исследовать существующие форматы хранения и технологии для автоматизированного учета в культуре.
2. Сформулировать требования и бизнес‑правила для системы управления галереей.
3. Спроектировать предметную область и структуру JSON‑файлов как основного хранилища.
4. Реализовать проект на языке C# разделив сервисную логику и консольный интерфейс.
5. Провести тестирование ключевых функций и продемонстрировать сценарии использования.

Предметом исследования является консольная программная система, реализующая процессы управления художественной галереей с хранением данных в JSON. Объектом исследования являются процессы учета и управления художественной галереей.

Для решения поставленных задач использовались методы: анализ (при описании предметной области); синтез (при реализации консольной программной системы); формализации (при проектировании консольной программной системы).

Разработанное консольная программная система для художественной галереи позволит улучшить качество обрабатываемой информации, повысит производительность труда специалиста, организует хранение информации.

# ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ОБЩАЯ ПОСТАНОВКА ЗАДАЧ

## 1.1. Актуальность и значение информационных систем в культуре

В последние десятилетия культурная сфера претерпевает существенные изменения под воздействием цифровых технологий. Художественные галереи, традиционно оперирующие бумажными картотеками, реестрами и отчетами, сталкиваются с растущим потоком информации: новые поступления произведений, временные выставки, аренда картин частными лицами и организациями, онлайн-бронирование билетов. Это многомерное пространство данных требует единого централизованного инструмента для ведения учёта, чтобы избежать дублирования записей и потерь информации.

Во-первых, автоматизация учёта экспонатов позволяет гарантировать сохранность уникальных объектов. Классический пример – перемещение картины на временную выставку в другом городе: без полноценной системы легко ошибиться в дате отправки, месте хранения или договоре аренды, что чревато штрафными санкциями и риском для объекта искусства. Информационная система фиксирует каждое действие над экспонатом, хранит историю реставраций и перемещений, что исключает «человеческий фактор» и облегчает работу кураторов и реставраторов.

Во-вторых, информационная система сокращает время обработки запросов на бронирование и продажу билетов. В условиях высокой проходимости галерея ежедневно регистрирует сотни транзакций: каждый билет должен быть пронумерован, привязан к конкретной дате и посетителю, а отчётность по выручке – формироваться в режиме реального времени. Использование базы данных с поддержкой транзакций и бизнес-логики гарантирует целостность финансовых потоков и прозрачность работы касс.

В-третьих, аналитические возможности информационной системы открывают новые горизонты развития. Системы отчётности способны строить графики динамики посещаемости, выявлять пиковые часы и наиболее востребованные выставки, анализировать демографию посетителей. Эти данные лежат в основе маркетинговых кампаний и стратегического планирования: галерея может заранее подготовить дополнительные экспозиции для семейной аудитории или предложить специальные тарифы для студентов.

Наконец, наличие современной информационной системы повышает уровень сервиса и взаимодействия с публикой. Онлайн-бронь, мобильные приложения с виртуальными турами, персонализированные рассылки – всё это элементы единой экосистемы, которая удерживает интерес зрителя, превращая разовый визит в лояльность к бренду галереи.

## 1.2. Анализ форматов и технологий хранения данных

Выбор технологии хранения определяет архитектуру системы «снизу», влияет на скорость работы, сложность поддержки и возможность расширения функционала. Рассмотрим четыре основных подхода.

Реляционные СУБД

Реляционные базы данных (например, PostgreSQL, MySQL, SQL Server) базируются на строгой схеме: таблицы с заранее определёнными полями и связями реализуют модель «сущность–связь» [9, 8].

Преимущества:

1. Целостность и непротиворечивость данных благодаря внешним ключам и транзакциям [9].
2. Мощный язык запросов SQL для выборок, агрегаций и отчётов [8].
3. Широкая экосистема инструментов – от графических администраторов до зрелых ORM-фреймворков.

Недостатки:

1. Низкая гибкость: изменение структуры таблицы (добавление или удаление поля) требует миграции схемы и остановки части сервиса [9].
2. Избыточность: для простых текстовых метаданных может потребоваться несколько объединённых таблиц и «джоин», что снижает производительность при больших объёмах запросов [8].

Документо-ориентированные хранилища

NoSQL-решения (MongoDB, CouchDB) хранят «документы» в формате JSON или BSON без жёсткой схемы. Каждый документ может иметь свой набор полей, а вложенные структуры легко отражают иерархию данных (например, в документе «выставка» сразу хранится список экспонатов с их атрибутами) [7, 2].

Преимущества:

1. Гибкость: добавление новых атрибутов не требует изменения схемы, достаточно сохранить дополнительные поля в документах.
2. Шардинг и горизонтальное масштабирование: легко распределить коллекции по нескольким узлам.
3. Простота интеграции с Web-фреймворками: прямое чтение/запись JSON без преобразований.

Недостатки:

1. Отсутствие транзакций на уровне нескольких документов (MongoDB поддерживает транзакции лишь с версии 4+ и при определённых условиях) [7].
2. Сложность сложных выборок и объединений «многие-ко-многим» без внедрения дополнительного слоя.

Файловое хранение в JSON

Все данные разбиваются по сущностям, и каждая сущность хранится в отдельном JSON-файле или в каталоге файлов (для картин – файл artworks.json, для художников – artists.json и т. д.) [1].

Преимущества:

1. Минимальные требования к развёртыванию: достаточно прав на чтение/запись в файловой системе.
2. Полная прозрачность: пользователь может открыть файл и сразу увидеть его структуру.
3. Отсутствие лицензионных и инфраструктурных затрат.

Недостатки:

1. Перебор всех файлов для поиска («линейный» поиск) замедляется при превышении нескольких сотен записей.
2. Нет поддержки параллельных транзакций: при одновременном доступе возможны «состояния гонки» без явной блокировки файлов.

CRM-системы и их роль

CRM (Customer Relationship Management) – системы, предназначенные для управления взаимодействием с клиентами: учёт контактов, история коммуникаций, сегментация, маркетинговые кампании и поддержка продаж [6, 10].

В контексте галерей CRM-системы применяются для:

1. Ведения базы посетителей и спонсоров, включая категории (VIP, корпоративные клиенты).
2. Автоматизации рассылок анонсов и напоминаний о предстоящих выставках.
3. Анализа эффективности маркетинговых активностей (реакция на приглашения, конверсия в посещения).

Однако CRM не приспособлена для учёта физических объектов искусства:

1. Нет встроенных модулей для фиксации перемещений и состояния экспонатов.
2. Сложно организовать учёт аренды и продаж картин без написания специализированных расширений.

CRM-движок отлично дополняет основную систему, отвечая за маркетинг и работу с аудиторией, но не заменяет полноценный «фоновой» учётный модуль.

Учитывая цели курсовой работы – простоту развёртывания и отсутствие внешних зависимостей – оптимальным решением становится файловая JSON-база в сочетании с консольным интерфейсом.

## 1.3. Обзор аналогов и сравнительный анализ

В современной практике управления музейными и галерейными коллекциями используются как специализированные коммерческие системы, так и решения с открытым исходным кодом, а также корпоративные CRM-платформы, расширенные под нужды культурной сферы. Ниже изложен анализ трех основных категорий таких продуктов.

Коммерческие системы

The Museum System (TMS) – «коробочное» решение с модулем учёта коллекций, GIS-возможностями и встроенной BI-аналитикой [11]. Внедрение TMS обеспечивает «под ключ» весь цикл работы с коллекцией, но требует значительных инвестиций и долгосрочной контрактной поддержки.

GallerySoft ориентирована на автоматизацию коммерческих процессов – продажу репродукций и аренду картин через встроенный интернет-магазин [4]. Несмотря на быстрый запуск, система ограничена преднастроенными бизнес-правилами, что снижает гибкость кастомизации.

Open‑source проекты

CollectionSpace является примером гибкой платформы с модульной архитектурой: хранилище данных, управление выставками, инструменты консервирования и REST‑API для интеграции с внешними приложениями [3]. Отсутствие лицензионной платы делает его привлекательным для организаций с ограниченным бюджетом, однако требует собственных усилий по установке, настройке и сопровождению.

Проект openMAINT изначально реализован как система управления активами (CMMS), но способен адаптироваться для галерейных нужд: учёта логистики и состояния объектов [12]. Тем не менее, его конфигурация затруднена, когда речь идёт о специфических процессах, присущих художественным коллекциям.

CRM‑платформы с расширениями

Корпоративные решения, такие как Salesforce Nonprofit Cloud и Microsoft Dynamics 365, изначально предназначены для управления взаимоотношениями с донорами и посетителями [10][6]. Функционал CRM легко расширяется кастомными объекты «Экспонат» или «Аренда», а встроенные инструменты маркетинга и аналитики позволяют организовать рассылки, отчёты по вовлечённости и сегментацию аудитории. При этом основные ограничения связаны с необходимостью разработки и поддержки дополнительных модулей, так как стандартный CRM‑интерфейс не учитывает все нюансы работы с физическими экспонатами.

Сравнительная оценка и рекомендации

Сравнивая представленные решения, можно выделить следующие критерии выбора:

1. Степень готовности «из коробки» и скорость внедрения.
2. Возможности настройки бизнес‑процессов и гибкость расширения.
3. Стоимость владения: лицензионные платежи и ресурсы на поддержку.
4. Масштабируемость и требования к инфраструктуре.

Для крупных государственных и коммерческих музеев оптимальным выбором становится TMS [11] или аналогичные коммерческие продукты, обеспечивающие полный цикл управления коллекцией и аналитикой при наличии бюджета.

Организации, готовые вкладываться в собственную ИТ‑команду и конфигурацию, могут обратить внимание на CollectionSpace [3] или openMAINT [12].

Малые галереи и учебные проекты выигрывают от использования лёгких файловых систем на базе JSON с открытым кодом, комбинируя их при необходимости с CRM‑модулем для работы с посетителями и маркетингом.

## 1.4. Постановка задачи и функциональные требования

В рамках курсового проекта разработана консольная система, предназначенная для учёта и управления ключевыми процессами художественной галереи.

Основная цель практической части – перевести наработанные теоретические модели в рабочую программу на языке C#, использовать файловое хранилище на базе JSON и организовать интуитивно понятный пользовательский интерфейс.

Перед началом разработки были сформулированы следующие требования:

1. Все данные должны храниться в отдельных JSON-файлах в каталоге (Data\). Каждая сущность представлена своим файлом, что обеспечивает модульность и упрощает резервное копирование.
2. Реализовать ключевые бизнес-сценарии: бронирование билета, оформление продажи картины, передача её в аренду, генерацию отчётов по статистике наличия, выручке и популярности авторов.
3. Обеспечить консольное меню с разделами для управления каждой сущностью и навигацией между разделами.
4. После каждой операции изменения данных выполнять запись в JSON, чтобы гарантировать сохранность при неожиданном завершении программы.
5. Поддерживать фильтрацию списков: отображение всех доступных картин, выставок ближайшего месяца, топ‑3 художников по проданным работам, список картин в аренде.
6. Реализация в среде .NET 6+, с использованием встроенных библиотек System.Text.Json для сериализации и лёгкого чтения/записи [5].
7. Выгружать данные в кодировке UTF-8 для возможности читать и изменять файлы вне программы и обмениваться ими с другими пользователями.

# ГЛАВА 2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ УПРАВЛЕНИЯ ХУДОЖЕСТВЕННОЙ ГАЛЕРЕЕЙ

## 2.1. Структура программного продукта

Для уточнения связей между сущностями была составлена ER‑диаграмма, отражающая основные отношения:

1. Каждое произведение искусства имеет своего автора (связь «многие к одному»).
2. Художник может иметь множество работ (обратная связь, «один ко многим»).
3. Выставка объединяет произвольный набор картин (отношение «многие ко многим»), решение реализовано через хранение списка идентификаторов произведений.
4. Посетитель может приобретать билеты на множество выставок (связь «один ко многим»), а также совершать покупки и аренды (дополнительные связи «один ко многим» с сущностями Sale и Rental).

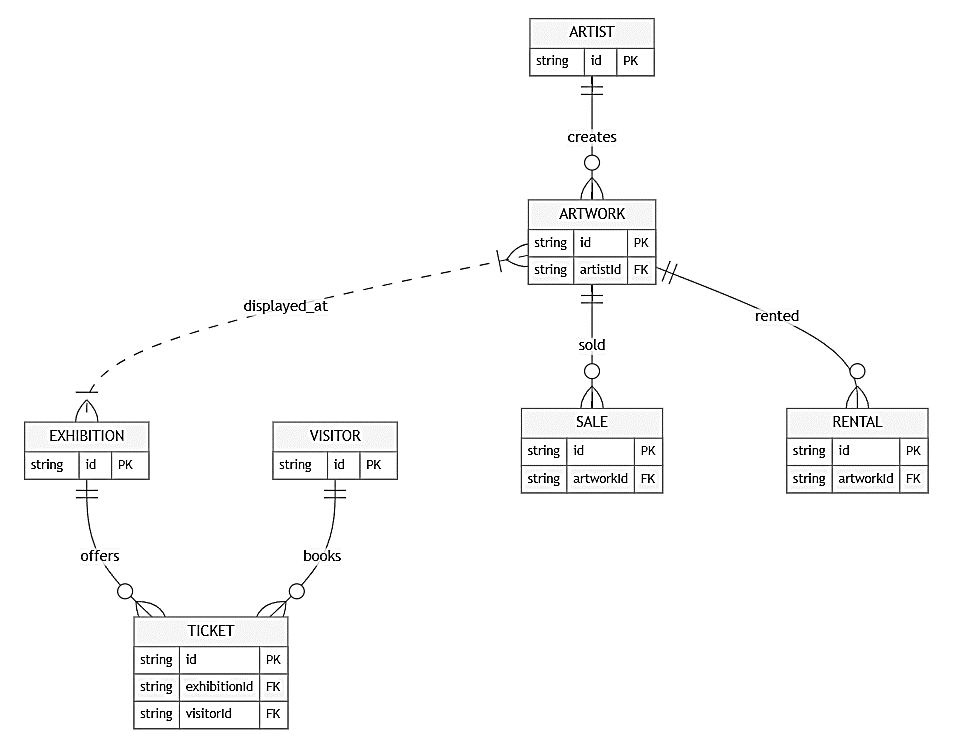


Рис. 2.1. Блок-схема связей в проекте

На практике связи заданы двумя способами:

1. В моделях C# для простоты навигации используются коллекции идентификаторов (List<Guid>). При загрузке данных сервис разрешает эти идентификаторы в объекты и строит полную картину.
2. JSON‑файлы не содержат дублированных вложенных структур, только ссылки-идентификаторы, что исключает избыточность.

## 2.2. Архитектура и структура JSON-хранилища

Система организована в три слоя:

1. Модель данных – набор классов Artwork, Artist, Exhibition, Visitor, Ticket, Sale, Rental с обязанными свойствами и атрибутом [JsonPropertyName] для контроля имён полей.
2. Репозитории – обобщённый класс JsonRepository<T> с методами GetAll(), GetById(Guid id), Add(T item), Update(T item), Remove(Guid id). Репозиторий при инициализации читает файл в List<T> и при изменениях перезаписывает его.
3. Сервисный слой – класс GalleryService инкапсулирует логику бизнес-операций: бронирование билета, оформление продажи, начисление статусов. Сервис оперирует ядром данных через репозитории.
4. Пользовательский интерфейс – класс Program с методом Main(), реализующим меню через цикл while (true) и switch по выбору пользователя.

Каталог проекта выглядит следующим образом:

1. Program.cs – точка входа, меню
2. Models/ – классы-сущности (Artwork, ArtworkStatus, Artist, Exhibition, Visitor, Ticket, TicketStatus, Sale, Rental, BaseEntity)
3. Repositories/ – IRepository.cs, JsonRepository.cs
4. Services/ – GalleryService.cs
5. Data/ – artworks.json, artists.json, exhibitions.json, visitors.json, tickets.json, sales.json, rentals.json

В проекте имеется модели Artwork, Artist, Exhibition, Visitor, Ticket, Sale, Rental и для каждой из них есть отдельный JSON файл со своим определенным форматом.

Далее будут предоставлены формат каждой сущности:

1. Формат JSON для «Произведения искусства».

Каждое произведение хранится в artworks.json как отдельный объект:

{

"id": "b3f1c2d4-5e6f-7a8b-9c0d-e1f2a3b4c5d6",

"title": "Звёздная ночь",

"artistId": "a1b2c3d4-5e6f-7a8b-9c0d-e1f2a3b4c5d6",

"year": 1889,

"genre": " Пейзаж",

"description": " Теплый весенний пейзаж",

"value": 150000,

"status": " Sold"

}

Это обеспечивает простое отображение полей и быструю фильтрацию, например, по полю status.

1. Формат JSON для «Художника»

В artists.json каждая запись включает базовую биографию и список работ:

{

"id": "a1b2c3d4-5e6f-7a8b-9c0d-e1f2a3b4c5d6",

"fullName": "Винсент ван Гог",

"country": "Нидерланды",

"lifeYears": 1950-2000,

"style": "Постимпрессионизм",

"artworkIds": ["b3f1c2d4-5e6f-7a8b-9c0d-e1f2a3b4c5d6", ...]

}

Поле artworkIds позволяет при загрузке собрать связанные объекты Artwork.

1. Формат JSON для «Выставки»

В файле exhibitions.json:

{

"id": "c1d2e3f4-5a6b-7c8d-9e0f-a1b2c3d4e5f6",

"title": "Импрессионизм во Франции",

"startDate": "2025-06-01",

"endDate": "2025-08-31",

"location": "Главный зал ТТИиП",

"artworkIds": ["b3f1c2d4-5e6f-7a8b-9c0d-e1f2a3b4c5d6", ...],

"ticketPrice": 500.00

}

Система рассчитывает доступные даты для бронирования на основе текущей даты и диапазона выставки.

1. Формат JSON для «Посетителя»

В visitors.json хранится профиль пользователя и история:

{

"id": "d1e2f3a4-5b6c-7d8e-9f0a-b1c2d3e4f5a6",

"fullName": "Иван Иванов",

"contactInfo": "+375291234567",

"visitedExhibitionIds": ["c1d2e3f4-5a6b-7c8d-9e0f-a1b2c3d4e5f6"],

"purchaseIds": ["e1f2a3b4-5c6d-7e8f-9a0b-c1d2e3f4a5b6"],

"rentalIds": []

}

Исторические списки позволяют формировать отчёты по лояльности.

1. Формат JSON для «Билета»

{

"ExhibitionId": "5aa8313d-e519-4a43-9708-5591b53f5ef7",

"VisitorId": "302564d8-42a2-4d40-b7ef-070310be6d54",

"VisitDate": "2025-05-13T00:00:00+03:00",

"Price": 500,

"Status": "Reserved",

"Id": "0b117e77-a822-4487-913c-ffd8524bbe57"

}

В tickets.json содержит ссылки на выставку и посетителя, дату визита, цену билета и статус (Reserved, Used, Cancelled).

1. Формат JSON для «Продажи»

{

"ArtworkId": "2b518d27-28c4-4a29-bb8a-6a6eaffd75f2",

"BuyerId": "302564d8-42a2-4d40-b7ef-070310be6d54",

"Date": "2025-05-11T13:15:04.0500248+03:00",

"Amount": 150000,

"Id": "e8a53155-093e-4184-8753-50fe17330af5"

}

В sales.json фиксирует проданные произведения: ссылка на картину, ссылка на покупателя, сумма и дата.

1. Формат JSON для «Аренды»

{

"ArtworkId": "c7e1727e-ef4d-4aed-90d8-774776a0aa61",

"RenterId": "302564d8-42a2-4d40-b7ef-070310be6d54",

"StartDate": "2025-05-11T00:00:00+03:00",

"EndDate": "2025-05-18T00:00:00+03:00",

"Cost": 10000,

"Id": "e184e227-8c10-4ea0-9fe2-f1fa050acd74"

}

В rentals.json аналогично хранит информацию о арендованных работах: даты начала/окончания и размер платы.

## 2.3. Реализация интерфейса и ключевых операций

Практическая часть сосредоточена на текстовом (консольном) интерфейсе, разбитом на отдельные меню, каждое из которых соответствует одной из бизнес‑сценариев галереи. Ниже описан функционал, доступный в проекте, на элементах меню:

При старте приложения выводится главное меню с разделами «Произведения искусства», «Художники», «Выставки», «Посетители», «Управление билетами», «Продажи и аренда» и «Отчёты». Пользователь выбирает раздел вводом номера пункта, после чего попадает в локальное меню.

Меню «Произведения искусства» позволяет выполнять три действия. Первое – просмотр всего списка экспонатов: система загружает все объекты из файла artworks.json, связывает их с авторами и выводит на экран идентификатор, название, данные художника, год, жанр, статус и оценочную стоимость. Второе действие – отображение только доступных работ: фильтрация по полю Status == InGallery. Третье – добавление нового экспоната: пользователь вводит название, идентификатор художника (из списка из artists.json), год создания, жанр, описание и оценочную стоимость. Введённые данные валидируются сервисом и сохраняются в репозиторий.

Меню «Художники» предоставляет пользователю два варианта: просмотр всех авторов и вывод самых популярных по количеству проданных работ. При первом действии загружаются записи из artists.json, выполняется подсчёт числа работ у каждого автора и выводится биография и число картин. Во втором – сервис собирает данные о продажах из sales.json, сопоставляет их с художниками и формирует рейтинг.

Меню «Выставки» включает функции просмотра всех выставок и выбор предстоящих. В первом режиме программа читает exhibitions.json, сопоставляет ссылки на картины и выводит период, место, количество экспонатов и цену билета. Во втором – дополнительно фильтрует по дате: от текущего дня до месяца вперед.

Меню «Посетители» позволяет отобразить список зарегистрированных гостей: читаются данные из visitors.json, выводятся ФИО, контактные данные, число посещений и совершённых покупок.

В меню «Управление билетами» реализованы просмотр всех бронирований и создание нового. Для брони система предлагает перечень предстоящих выставок, затем – список посетителей. После ввода идентификаторов выставки и клиента, а также даты визита, сервис проверяет корректность периода и сохраняет новый объект Ticket со статусом Reserved.

Раздел «Продажи и аренда» объединяет четыре действия. Просмотр существующих продаж и аренд считывает файлы sales.json и rentals.json, сопоставляет их с сущностями Artwork и Visitor, выводит подробности сделки. Операции оформления сделки – продажа или аренда – проводятся через методы SellArtwork и RentArtwork: проверяется доступность картины, вводятся данные клиента и суммы, затем обновляется статус экспоната и фиксируется запись о продаже или аренде.

Меню «Отчёты» предоставляет сводные данные по ключевым показателям: список доступных экспонатов, ближайшие выставки, рейтинг художников по продажам, общая выручка (билеты и картины) и текущие арендные соглашения. Логика формирования отчётов реализована в методах GetAvailableArtworks, GetUpcomingExhibitions, GetPopularArtistsBySales, GetTotalRevenue и GetRentedArtworks.

Все операции над данными («создание» и «чтение») реализованы при помощи сервиса ArtGalleryService и обобщённого репозитория JsonRepository<T>.

## 2.4. Тестирование и примеры работы

В данном разделе представлены ключевые сценарии использования системы в реальных условиях и наглядные примеры её работы.

На рис. 2.2 показано главное меню проекта, в котором пользователь может выбрать один из модулей системы: «Произведения искусства», «Художники», «Выставки», «Посетители», «Управление билетами», «Продажи и аренда» и «Отчёты». Такой интерфейс обеспечивает быстрый доступ ко всем возможностям приложения.

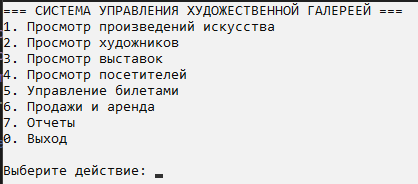


Рис. 2.2 Главное меню проекта.

Рис. 2.3 иллюстрирует вывод полного списка произведений искусства. Система загружает данные из artworks.json, сопоставляет их с информацией об авторах и отображает идентификаторы, названия, авторов, годы создания и статус каждого экспоната. Это позволяет оператору мгновенно оценивать текущую коллекцию галереи.

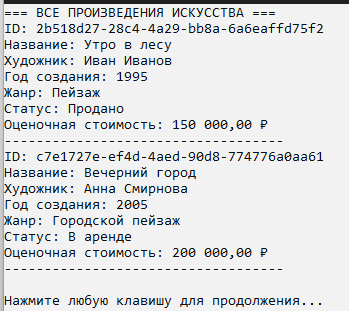


Рис. 2.3 Все произведения искусства

На рис. 2.4 демонстрируется процесс добавления нового произведения: ввод данных о картине, выбор художника из существующего списка и сохранение записи. Интерфейс подсказывает порядок ввода полей и проверяет корректность формата (например, год создания не может быть больше текущего).

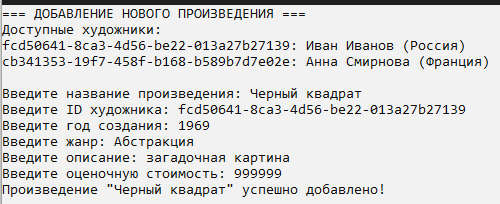


Рис. 2.4 Демонстрация добавления произведения

Рис. 2.5 показывает подсчёт общей выручки. Пользователь выбирает раздел «Отчёты», после чего система агрегирует суммы продаж и билетов, выдавая итоговую цифру. Такой отчёт помогает быстро получить финансовую картину работы галереи.

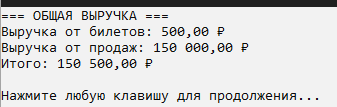


Рис. 2.5 Демонстрация подсчета выручки

На рис. 2.6 приведена сводка об арендованных картинах: выводятся идентификаторы работ, имена арендаторов и даты начала/окончания договора. Это упрощает контроль за срочностью возврата арендованных экспонатов.

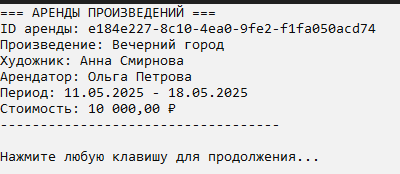


Рис. 2.6 Сводка об арендованных картинах

Рис. 2.7 иллюстрирует механизм проверки вводимых данных: при попытке ввести некорректный формат даты или несуществующий идентификатор система выводит понятное сообщение об ошибке и предлагает повторить ввод. Такая защита позволяет избежать нарушений целостности данных.

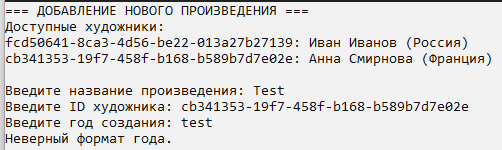


Рис. 2.7 Демонстрация проверок вводимых данных

Проведённый анализ основных экранов и пользовательских сценариев подтвердил корректность работы ключевых функций системы. Интерфейс продемонстрировал достаточную стабильность и удобство работы в консольном режиме, что делает его пригодным для эксплуатации в малых и средних художественных галереях.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были последовательно решены задачи исследования предметной области и практической реализации системы управления художественной галереей. В первом разделе углублённо рассмотрены ключевые аспекты автоматизации в сфере культуры: процессы учёта экспонатов, значение достоверных данных для планирования выставок и аналитики, а также возможности интеграции с внешними сервисами.

В процессе разработки консольной программы на C# были достигнуты следующие результаты:

1. Проведён всесторонний анализ технологий хранения, что позволило выработать чёткие критерии выбора формата данных и принципов архитектуры.
2. Спроектирована предметная область: перечислены основные сущности, установлены их взаимосвязи и разработаны схемы для безопасного хранения ссылочных данных.
3. Реализованы критически важные сценарии работы галереи: учёт экспонатов, управление художниками, организация выставок, бронирование билетов, оформление продаж и аренды.
4. Изучены особенности сериализации/десериализации JSON: от обработки ошибок формата до влияния структуры на скорость загрузки данных.

Внедрение разработанного решения автоматизирует основные операции художественной галереи, снижает трудозатраты сотрудников и минимизирует риски потери информации. Полученная система служит надёжным фундаментом для расширения и может быть дополнена интеграцией веб-интерфейса, поддержкой распределённых баз данных и мощным модулем отчётности, что обеспечит долгосрочное развитие проекта и адаптацию к потребностям современных культурных учреждений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Брей Т. The JavaScript Object Notation (JSON) Data Interchange Format: RFC 8259. IETF, 2017.
2. Apache CouchDB [Электронный ресурс]. – Режим доступа: https://couchdb.apache.org – Дата обращения: 11.05.2025.
3. CollectionSpace: Official Documentation [Электронный ресурс]. – Режим доступа: https://www.collectionspace.org – Дата обращения: 11.05.2025.
4. GallerySoft: Official Website [Электронный ресурс]. – Режим доступа: https://www.gallerysoft.com – Дата обращения: 11.05.2025.
5. Microsoft. System.Text.Json Namespace. Microsoft Docs [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/– Дата обращения: 11.05.2025.
6. Microsoft Dynamics 365 Documentation [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dynamics365/ – Дата обращения: 11.05.2025.
7. MongoDB: Official Website [Электронный ресурс]. – Режим доступа: https://www.mongodb.com – Дата обращения: 11.05.2025.
8. MySQL: Official Website [Электронный ресурс]. – Режим доступа: https://www.mysql.com – Дата обращения: 11.05.2025.
9. PostgreSQL: Official Website [Электронный ресурс]. – Режим доступа: https://www.postgresql.org – Дата обращения: 11.05.2025.
10. Salesforce Nonprofit Cloud Documentation [Электронный ресурс]. – Режим доступа: https://www.salesforce.com/products/nonprofit-cloud/overview – Дата обращения: 11.05.2025.
11. The Museum System (TMS): Official Website [Электронный ресурс]. – Режим доступа: https://www.gallerysystems.com/products/tms – Дата обращения: 11.05.2025.
12. openMAINT: Official Website [Электронный ресурс]. – Режим доступа: https://www.openmaint.org – Дата обращения: 11.05.2025.

# ПРИЛОЖЕНИЕ 1

namespace ArtGallery

{

using ArtGallery.Models;

using ArtGallery.Services;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

public class Program

{

private static ArtGalleryService \_service;

static void Main(string[] args)

{

// Console.BackgroundColor = ConsoleColor.White;

// Console.ForegroundColor = ConsoleColor.Black;

Console.OutputEncoding = System.Text.Encoding.UTF8;

Console.InputEncoding = System.Text.Encoding.UTF8;

CultureInfo.CurrentCulture = new CultureInfo("ru-RU");

\_service = new ArtGalleryService();

// Инициализация примерных данных для демонстрации

InitializeSampleData();

bool exit = false;

while (!exit)

{

Console.Clear();

Console.WriteLine("=== СИСТЕМА УПРАВЛЕНИЯ ХУДОЖЕСТВЕННОЙ ГАЛЕРЕЕЙ ===");

Console.WriteLine("1. Просмотр произведений искусства");

Console.WriteLine("2. Просмотр художников");

Console.WriteLine("3. Просмотр выставок");

Console.WriteLine("4. Просмотр посетителей");

Console.WriteLine("5. Управление билетами");

Console.WriteLine("6. Продажи и аренда");

Console.WriteLine("7. Отчеты");

Console.WriteLine("0. Выход");

Console.Write("\nВыберите действие: ");

if (int.TryParse(Console.ReadLine(), out int choice))

{

Console.Clear();

switch (choice)

{

case 1:

ArtworksMenu();

break;

case 2:

ArtistsMenu();

break;

case 3:

ExhibitionsMenu();

break;

case 4:

VisitorsMenu();

break;

case 5:

TicketsMenu();

break;

case 6:

SalesAndRentalsMenu();

break;

case 7:

ReportsMenu();

break;

case 0:

exit = true;

break;

default:

Console.WriteLine("Неверный выбор. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

break;

}

}

else

{

Console.WriteLine("Неверный ввод. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

}

}

}

// === МЕНЕДЖМЕНТ ПРОИЗВЕДЕНИЙ ИСКУССТВА ===

private static void ArtworksMenu()

{

bool back = false;

while (!back)

{

Console.Clear();

Console.WriteLine("=== ПРОИЗВЕДЕНИЯ ИСКУССТВА ===");

Console.WriteLine("1. Просмотр всех произведений");

Console.WriteLine("2. Просмотр доступных произведений");

Console.WriteLine("3. Добавить новое произведение");

Console.WriteLine("0. Назад");

Console.Write("\nВыберите действие: ");

if (int.TryParse(Console.ReadLine(), out int choice))

{

Console.Clear();

switch (choice)

{

case 1:

ShowAllArtworks();

break;

case 2:

ShowAvailableArtworks();

break;

case 3:

AddNewArtwork();

break;

case 0:

back = true;

break;

default:

Console.WriteLine("Неверный выбор. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

break;

}

}

else

{

Console.WriteLine("Неверный ввод. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

}

}

}

private static void ShowAllArtworks()

{

\_service.LoadReferences();

var artworks = \_service.GetAllArtworks();

Console.WriteLine("=== ВСЕ ПРОИЗВЕДЕНИЯ ИСКУССТВА ===");

if (artworks.Count == 0)

{

Console.WriteLine("Произведения отсутствуют.");

}

else

{

foreach (var artwork in artworks)

{

Console.WriteLine($"ID: {artwork.Id}");

Console.WriteLine($"Название: {artwork.Title}");

Console.WriteLine($"Художник: {artwork.Artist?.FullName ?? "Неизвестно"}");

Console.WriteLine($"Год создания: {artwork.Year}");

Console.WriteLine($"Жанр: {artwork.Genre}");

Console.WriteLine($"Статус: {GetArtworkStatusName(artwork.Status)}");

Console.WriteLine($"Оценочная стоимость: {artwork.EstimatedValue:C}");

Console.WriteLine("-----------------------------------");

}

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

private static void ShowAvailableArtworks()

{

var artworks = \_service.GetAvailableArtworks();

Console.WriteLine("=== ДОСТУПНЫЕ ПРОИЗВЕДЕНИЯ ИСКУССТВА ===");

if (artworks.Count == 0)

{

Console.WriteLine("Доступные произведения отсутствуют.");

}

else

{

foreach (var artwork in artworks)

{

Console.WriteLine($"ID: {artwork.Id}");

Console.WriteLine($"Название: {artwork.Title}");

Console.WriteLine($"Художник: {artwork.Artist?.FullName ?? "Неизвестно"}");

Console.WriteLine($"Год создания: {artwork.Year}");

Console.WriteLine($"Жанр: {artwork.Genre}");

Console.WriteLine($"Оценочная стоимость: {artwork.EstimatedValue:C}");

Console.WriteLine("-----------------------------------");

}

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

private static void AddNewArtwork()

{

Console.WriteLine("=== ДОБАВЛЕНИЕ НОВОГО ПРОИЗВЕДЕНИЯ ===");

\_service.LoadReferences();

// Показываем список художников для выбора

var artists = \_service.GetAllArtists();

Console.WriteLine("Доступные художники:");

foreach (var artist in artists)

{

Console.WriteLine($"{artist.Id}: {artist.FullName} ({artist.Country})");

}

Console.Write("\nВведите название произведения: ");

string title = Console.ReadLine();

Console.Write("Введите ID художника: ");

string artistId = Console.ReadLine();

Console.Write("Введите год создания: ");

if (!int.TryParse(Console.ReadLine(), out int year))

{

Console.WriteLine("Неверный формат года.");

Console.ReadKey();

return;

}

Console.Write("Введите жанр: ");

string genre = Console.ReadLine();

Console.Write("Введите описание: ");

string description = Console.ReadLine();

Console.Write("Введите оценочную стоимость: ");

if (!decimal.TryParse(Console.ReadLine(), out decimal value))

{

Console.WriteLine("Неверный формат стоимости.");

Console.ReadKey();

return;

}

try

{

var artwork = \_service.AddArtwork(title, artistId, year, genre, description, value);

Console.WriteLine($"Произведение \"{artwork.Title}\" успешно добавлено!");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

// === МЕНЕДЖМЕНТ ХУДОЖНИКОВ ===

private static void ArtistsMenu()

{

bool back = false;

while (!back)

{

Console.Clear();

Console.WriteLine("=== ХУДОЖНИКИ ===");

Console.WriteLine("1. Просмотр всех художников");

Console.WriteLine("2. Популярные художники по продажам");

Console.WriteLine("0. Назад");

Console.Write("\nВыберите действие: ");

if (int.TryParse(Console.ReadLine(), out int choice))

{

Console.Clear();

switch (choice)

{

case 1:

ShowAllArtists();

break;

case 2:

ShowPopularArtists();

break;

case 0:

back = true;

break;

default:

Console.WriteLine("Неверный выбор. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

break;

}

}

else

{

Console.WriteLine("Неверный ввод. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

}

}

}

private static void ShowAllArtists()

{

\_service.LoadReferences();

var artists = \_service.GetAllArtists();

Console.WriteLine("=== ВСЕ ХУДОЖНИКИ ===");

if (artists.Count == 0)

{

Console.WriteLine("Художники отсутствуют.");

}

else

{

foreach (var artist in artists)

{

Console.WriteLine($"ID: {artist.Id}");

Console.WriteLine($"ФИО: {artist.FullName}");

Console.WriteLine($"Страна: {artist.Country}");

Console.WriteLine($"Годы жизни: {artist.LifeYears}");

Console.WriteLine($"Стиль: {artist.Style}");

Console.WriteLine($"Количество работ: {artist.ArtworkIds.Count}");

Console.WriteLine("-----------------------------------");

}

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

private static void ShowPopularArtists()

{

var popularArtists = \_service.GetPopularArtistsBySales();

Console.WriteLine("=== ПОПУЛЯРНЫЕ ХУДОЖНИКИ ПО ПРОДАЖАМ ===");

if (popularArtists.Count == 0)

{

Console.WriteLine("Данные о продажах отсутствуют.");

}

else

{

int rank = 1;

foreach (var item in popularArtists)

{

Console.WriteLine($"{rank}. {item.Artist.FullName} - {item.SoldCount} проданных работ");

rank++;

}

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

// === МЕНЕДЖМЕНТ ВЫСТАВОК ===

private static void ExhibitionsMenu()

{

bool back = false;

while (!back)

{

Console.Clear();

Console.WriteLine("=== ВЫСТАВКИ ===");

Console.WriteLine("1. Просмотр всех выставок");

Console.WriteLine("2. Предстоящие выставки");

Console.WriteLine("0. Назад");

Console.Write("\nВыберите действие: ");

if (int.TryParse(Console.ReadLine(), out int choice))

{

Console.Clear();

switch (choice)

{

case 1:

ShowAllExhibitions();

break;

case 2:

ShowUpcomingExhibitions();

break;

case 0:

back = true;

break;

default:

Console.WriteLine("Неверный выбор. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

break;

}

}

else

{

Console.WriteLine("Неверный ввод. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

}

}

}

private static void ShowAllExhibitions()

{

\_service.LoadReferences();

var exhibitions = \_service.GetAllExhibitions();

Console.WriteLine("=== ВСЕ ВЫСТАВКИ ===");

if (exhibitions.Count == 0)

{

Console.WriteLine("Выставки отсутствуют.");

}

else

{

foreach (var exhibition in exhibitions)

{

Console.WriteLine($"ID: {exhibition.Id}");

Console.WriteLine($"Название: {exhibition.Title}");

Console.WriteLine(

$"Период: {exhibition.StartDate.ToShortDateString()} - {exhibition.EndDate.ToShortDateString()}");

Console.WriteLine($"Место: {exhibition.Location}");

Console.WriteLine($"Количество экспонатов: {exhibition.ArtworkIds.Count}");

Console.WriteLine($"Цена билета: {exhibition.TicketPrice:C}");

Console.WriteLine("-----------------------------------");

}

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

private static void ShowUpcomingExhibitions()

{

var exhibitions = \_service.GetUpcomingExhibitions();

Console.WriteLine("=== ПРЕДСТОЯЩИЕ ВЫСТАВКИ ===");

if (exhibitions.Count == 0)

{

Console.WriteLine("Предстоящие выставки отсутствуют.");

}

else

{

foreach (var exhibition in exhibitions)

{

string status = DateTime.Today < exhibition.StartDate ? "Скоро" : "Действует";

Console.WriteLine($"ID: {exhibition.Id}");

Console.WriteLine($"Название: {exhibition.Title} ({status})");

Console.WriteLine(

$"Период: {exhibition.StartDate.ToShortDateString()} - {exhibition.EndDate.ToShortDateString()}");

Console.WriteLine($"Место: {exhibition.Location}");

Console.WriteLine($"Цена билета: {exhibition.TicketPrice:C}");

Console.WriteLine("-----------------------------------");

}

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

// === МЕНЕДЖМЕНТ ПОСЕТИТЕЛЕЙ ===

private static void VisitorsMenu()

{

bool back = false;

while (!back)

{

Console.Clear();

Console.WriteLine("=== ПОСЕТИТЕЛИ ===");

Console.WriteLine("1. Просмотр всех посетителей");

Console.WriteLine("0. Назад");

Console.Write("\nВыберите действие: ");

if (int.TryParse(Console.ReadLine(), out int choice))

{

Console.Clear();

switch (choice)

{

case 1:

ShowAllVisitors();

break;

case 0:

back = true;

break;

default:

Console.WriteLine("Неверный выбор. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

break;

}

}

else

{

Console.WriteLine("Неверный ввод. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

}

}

}

private static void ShowAllVisitors()

{

\_service.LoadReferences();

var visitors = \_service.GetAllVisitors();

Console.WriteLine("=== ВСЕ ПОСЕТИТЕЛИ ===");

if (visitors.Count == 0)

{

Console.WriteLine("Посетители отсутствуют.");

}

else

{

foreach (var visitor in visitors)

{

Console.WriteLine($"ID: {visitor.Id}");

Console.WriteLine($"ФИО: {visitor.FullName}");

Console.WriteLine($"Контактная информация: {visitor.ContactInfo}");

Console.WriteLine($"Количество посещений: {visitor.VisitHistory.Count}");

Console.WriteLine($"Количество покупок: {visitor.PurchaseIds.Count}");

Console.WriteLine("-----------------------------------");

}

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

// === МЕНЕДЖМЕНТ БИЛЕТОВ ===

private static void TicketsMenu()

{

bool back = false;

while (!back)

{

Console.Clear();

Console.WriteLine("=== БИЛЕТЫ ===");

Console.WriteLine("1. Просмотр всех билетов");

Console.WriteLine("2. Забронировать билет");

Console.WriteLine("0. Назад");

Console.Write("\nВыберите действие: ");

if (int.TryParse(Console.ReadLine(), out int choice))

{

Console.Clear();

switch (choice)

{

case 1:

ShowAllTickets();

break;

case 2:

BookNewTicket();

break;

case 0:

back = true;

break;

default:

Console.WriteLine("Неверный выбор. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

break;

}

}

else

{

Console.WriteLine("Неверный ввод. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

}

}

}

private static void ShowAllTickets()

{

\_service.LoadReferences();

var tickets = \_service.GetAllTickets();

Console.WriteLine("=== ВСЕ БИЛЕТЫ ===");

if (tickets.Count == 0)

{

Console.WriteLine("Билеты отсутствуют.");

}

else

{

foreach (var ticket in tickets)

{

Console.WriteLine($"ID: {ticket.Id}");

Console.WriteLine($"Выставка: {ticket.Exhibition?.Title ?? "Неизвестно"}");

Console.WriteLine($"Посетитель: {ticket.Visitor?.FullName ?? "Неизвестно"}");

Console.WriteLine($"Дата посещения: {ticket.VisitDate.ToShortDateString()}");

Console.WriteLine($"Цена: {ticket.Price:C}");

Console.WriteLine($"Статус: {GetTicketStatusName(ticket.Status)}");

Console.WriteLine("-----------------------------------");

}

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

private static void BookNewTicket()

{

Console.WriteLine("=== БРОНИРОВАНИЕ БИЛЕТА ===");

// Показываем список выставок

var exhibitions = \_service.GetUpcomingExhibitions();

Console.WriteLine("Доступные выставки:");

foreach (var exhibition in exhibitions)

{

Console.WriteLine($"{exhibition.Id}: {exhibition.Title} " +

$"({exhibition.StartDate.ToShortDateString()} - {exhibition.EndDate.ToShortDateString()})");

}

Console.Write("\nВведите ID выставки: ");

string exhibitionId = Console.ReadLine();

\_service.LoadReferences();

// Показываем список посетителей

var visitors = \_service.GetAllVisitors();

Console.WriteLine("\nЗарегистрированные посетители:");

foreach (var visitor in visitors)

{

Console.WriteLine($"{visitor.Id}: {visitor.FullName}");

}

Console.Write("\nВведите ID посетителя: ");

string visitorId = Console.ReadLine();

Console.Write("Введите дату посещения (дд.мм.гггг): ");

if (!DateTime.TryParse(Console.ReadLine(), out DateTime visitDate))

{

Console.WriteLine("Неверный формат даты.");

Console.ReadKey();

return;

}

try

{

var ticket = \_service.BookTicket(exhibitionId, visitorId, visitDate);

Console.WriteLine($"Билет успешно забронирован! ID билета: {ticket.Id}");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

// === ПРОДАЖИ И АРЕНДА ===

private static void SalesAndRentalsMenu()

{

bool back = false;

while (!back)

{

Console.Clear();

Console.WriteLine("=== ПРОДАЖИ И АРЕНДА ===");

Console.WriteLine("1. Просмотр всех продаж");

Console.WriteLine("2. Просмотр всех аренд");

Console.WriteLine("3. Продать произведение");

Console.WriteLine("4. Сдать произведение в аренду");

Console.WriteLine("0. Назад");

Console.Write("\nВыберите действие: ");

if (int.TryParse(Console.ReadLine(), out int choice))

{

Console.Clear();

switch (choice)

{

case 1:

ShowAllSales();

break;

case 2:

ShowAllRentals();

break;

case 3:

SellArtwork();

break;

case 4:

RentArtwork();

break;

case 0:

back = true;

break;

default:

Console.WriteLine("Неверный выбор. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

break;

}

}

else

{

Console.WriteLine("Неверный ввод. Нажмите любую клавишу для продолжения...");

Console.ReadKey();

}

}

}

private static void ShowAllSales()

{

\_service.LoadReferences();

var sales = \_service.GetAllSales();

Console.WriteLine("=== ВСЕ ПРОДАЖИ ===");

if (sales.Count == 0)

{

Console.WriteLine("Продажи отсутствуют.");

}

else

{

foreach (var sale in sales)

{

Console.WriteLine($"ID: {sale.Id}");

Console.WriteLine($"Произведение: {sale.Artwork?.Title ?? "Неизвестно"}");

Console.WriteLine($"Художник: {sale.Artwork?.Artist?.FullName ?? "Неизвестно"}");

Console.WriteLine($"Покупатель: {sale.Buyer?.FullName ?? "Неизвестно"}");

Console.WriteLine($"Дата: {sale.Date.ToShortDateString()}");

Console.WriteLine($"Сумма: {sale.Amount:C}");

Console.WriteLine("-----------------------------------");

}

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

private static void ShowAllRentals()

{

var rentedArtworks = \_service.GetRentedArtworks();

var visitors = \_service.GetAllVisitors();

Console.WriteLine("=== АРЕНДЫ ПРОИЗВЕДЕНИЙ ===");

if (rentedArtworks.Count == 0)

{

Console.WriteLine("Активные аренды отсутствуют.");

}

else

{

foreach (var item in rentedArtworks)

{

var visitor = visitors.FirstOrDefault(x => x.Id == item.Rental.RenterId);

Console.WriteLine($"ID аренды: {item.Rental.Id}");

Console.WriteLine($"Произведение: {item.Artwork?.Title ?? "Неизвестно"}");

Console.WriteLine($"Художник: {item.Artwork?.Artist?.FullName ?? "Неизвестно"}");

Console.WriteLine($"Арендатор: {visitor?.FullName ?? item.Rental.RenterId}");

Console.WriteLine(

$"Период: {item.Rental.StartDate.ToShortDateString()} - {item.Rental.EndDate.ToShortDateString()}");

Console.WriteLine($"Стоимость: {item.Rental.Cost:C}");

Console.WriteLine("-----------------------------------");

}

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

private static void SellArtwork()

{

Console.WriteLine("=== ПРОДАЖА ПРОИЗВЕДЕНИЯ ===");

// Показываем список доступных произведений

var availableArtworks = \_service.GetAvailableArtworks();

Console.WriteLine("Доступные произведения:");

foreach (var artwork in availableArtworks)

{

Console.WriteLine($"{artwork.Id}: {artwork.Title} - {artwork.Artist?.FullName ?? "Неизвестно"} " +

$"(Оценочная стоимость: {artwork.EstimatedValue:C})");

}

Console.Write("\nВведите ID произведения: ");

string artworkId = Console.ReadLine();

\_service.LoadReferences();

// Показываем список посетителей

var visitors = \_service.GetAllVisitors();

Console.WriteLine("\nПосетители (потенциальные покупатели):");

foreach (var visitor in visitors)

{

Console.WriteLine($"{visitor.Id}: {visitor.FullName}");

}

Console.Write("\nВведите ID покупателя: ");

string buyerId = Console.ReadLine();

Console.Write("Введите сумму продажи: ");

if (!decimal.TryParse(Console.ReadLine(), out decimal amount))

{

Console.WriteLine("Неверный формат суммы.");

Console.ReadKey();

return;

}

try

{

var sale = \_service.SellArtwork(artworkId, buyerId, amount);

Console.WriteLine($"Произведение успешно продано за {sale.Amount:C}!");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

private static void RentArtwork()

{

Console.WriteLine("=== АРЕНДА ПРОИЗВЕДЕНИЯ ===");

// Показываем список доступных произведений

var availableArtworks = \_service.GetAvailableArtworks();

Console.WriteLine("Доступные произведения:");

foreach (var artwork in availableArtworks)

{

Console.WriteLine($"{artwork.Id}: {artwork.Title} - {artwork.Artist?.FullName ?? "Неизвестно"}");

}

Console.Write("\nВведите ID произведения: ");

string artworkId = Console.ReadLine();

\_service.LoadReferences();

// Показываем список посетителей

var visitors = \_service.GetAllVisitors();

Console.WriteLine("\nПосетители (потенциальные арендаторы):");

foreach (var visitor in visitors)

{

Console.WriteLine($"{visitor.Id}: {visitor.FullName}");

}

Console.Write("\nВведите ID арендатора: ");

string renterId = Console.ReadLine();

Console.Write("Введите дату начала аренды (дд.мм.гггг): ");

if (!DateTime.TryParse(Console.ReadLine(), out DateTime startDate))

{

Console.WriteLine("Неверный формат даты.");

Console.ReadKey();

return;

}

Console.Write("Введите дату окончания аренды (дд.мм.гггг): ");

if (!DateTime.TryParse(Console.ReadLine(), out DateTime endDate))

{

Console.WriteLine("Неверный формат даты.");

Console.ReadKey();

return;

}

Console.Write("Введите стоимость аренды: ");

if (!decimal.TryParse(Console.ReadLine(), out decimal cost))

{

Console.WriteLine("Неверный формат стоимости.");

Console.ReadKey();

return;

}

try

{

var rental = \_service.RentArtwork(artworkId, renterId, startDate, endDate, cost);

Console.WriteLine($"Произведение успешно сдано в аренду до {rental.EndDate.ToShortDateString()}!");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

// Меню отчетов

private static void ReportsMenu()

{

bool back = false;

while (!back)

{

Console.Clear();

Console.WriteLine("=== ОТЧЕТЫ ===");

Console.WriteLine("1. Список доступных произведений");

Console.WriteLine("2. Выставки ближайшего месяца");

Console.WriteLine("3. Популярные художники по продажам");

Console.WriteLine("4. Общая выручка");

Console.WriteLine("5. Арендованные произведения");

Console.WriteLine("0. Назад");

Console.Write("\nВыберите действие: ");

if (!int.TryParse(Console.ReadLine(), out int choice))

{

Console.WriteLine("Неверный ввод.");

Console.ReadKey();

continue;

}

Console.Clear();

switch (choice)

{

case 1:

ShowAvailableArtworks();

break;

case 2:

ShowUpcomingExhibitions();

break;

case 3:

ShowPopularArtists();

break;

case 4:

ShowTotalRevenue();

break;

case 5:

ShowAllRentals();

break;

case 0:

back = true;

continue;

default:

Console.WriteLine("Неверный выбор.");

break;

}

}

}

private static void ShowTotalRevenue()

{

\_service.LoadReferences();

var (ticketRev, salesRev, total) = \_service.GetTotalRevenue();

Console.WriteLine("=== ОБЩАЯ ВЫРУЧКА ===");

Console.WriteLine($"Выручка от билетов: {ticketRev:C}");

Console.WriteLine($"Выручка от продаж: {salesRev:C}");

Console.WriteLine($"Итого: {total:C}");

Console.WriteLine("\nНажмите любую клавишу для продолжения...");

Console.ReadKey();

}

// Инициализация демонстрационных данных

private static void InitializeSampleData()

{

\_service.LoadReferences();

// Если данные уже загружены, пропускаем повторное добавление

if (\_service.GetAllArtists().Any()) return;

// Добавляем художников

var artist1 = \_service.AddArtist("Иван Иванов", "Россия", "1950-2000", "Реализм");

var artist2 = \_service.AddArtist("Анна Смирнова", "Франция", "1965-наст", "Импрессионизм");

// Добавляем произведения искусства

var art1 = \_service.AddArtwork(

"Утро в лесу",

artist1.Id,

1995,

"Пейзаж",

"Теплый весенний пейзаж",

150000m

);

var art2 = \_service.AddArtwork(

"Вечерний город",

artist2.Id,

2005,

"Городской пейзаж",

"Огни ночного мегаполиса",

200000m

);

// Добавляем выставку

var exhibition = \_service.AddExhibition(

"Весенняя коллекция",

DateTime.Today.AddDays(1),

DateTime.Today.AddDays(30),

"Главный зал",

new List<string> { art1.Id, art2.Id },

500m

);

// Добавляем посетителя

var visitor = \_service.AddVisitor("Ольга Петрова", "olga.pet@example.com");

// Бронирование билета

\_service.BookTicket(exhibition.Id, visitor.Id, DateTime.Today.AddDays(2));

// Продажа и аренда произведений

\_service.SellArtwork(art1.Id, visitor.Id, art1.EstimatedValue);

\_service.RentArtwork(art2.Id, visitor.Id, DateTime.Today, DateTime.Today.AddDays(7), 10000m);

}

// Помощники для отображения статусов

private static string GetArtworkStatusName(ArtworkStatus status)

{

return status switch

{

ArtworkStatus.InGallery => "В галерее",

ArtworkStatus.Sold => "Продано",

ArtworkStatus.Rented => "В аренде",

\_ => status.ToString()

};

}

private static string GetTicketStatusName(TicketStatus status)

{

return status switch

{

TicketStatus.Reserved => "Забронирован",

TicketStatus.Used => "Использован",

TicketStatus.Cancelled => "Отменен",

\_ => status.ToString()

};

}

}

}

using ArtGallery.Models;

using ArtGallery.Repository;

namespace ArtGallery.Services

{

public class ArtGalleryService

{

private readonly IRepository<Artwork> \_artworkRepo;

private readonly IRepository<Artist> \_artistRepo;

private readonly IRepository<Exhibition> \_exhibitionRepo;

private readonly IRepository<Visitor> \_visitorRepo;

private readonly IRepository<Ticket> \_ticketRepo;

private readonly IRepository<Sale> \_saleRepo;

private readonly IRepository<Rental> \_rentalRepo;

public ArtGalleryService()

{

\_artworkRepo = new JsonRepository<Artwork>("artworks.json");

\_artistRepo = new JsonRepository<Artist>("artists.json");

\_exhibitionRepo = new JsonRepository<Exhibition>("exhibitions.json");

\_visitorRepo = new JsonRepository<Visitor>("visitors.json");

\_ticketRepo = new JsonRepository<Ticket>("tickets.json");

\_saleRepo = new JsonRepository<Sale>("sales.json");

\_rentalRepo = new JsonRepository<Rental>("rentals.json");

}

// Загрузка и установление связей между сущностями

public void LoadReferences()

{

var artworks = \_artworkRepo.GetAll();

var artists = \_artistRepo.GetAll();

var exhibitions = \_exhibitionRepo.GetAll();

var visitors = \_visitorRepo.GetAll();

var tickets = \_ticketRepo.GetAll();

var sales = \_saleRepo.GetAll();

var rentals = \_rentalRepo.GetAll();

// Связываем художников и произведения искусства

foreach (var artwork in artworks)

{

artwork.Artist = artists.FirstOrDefault(a => a.Id == artwork.ArtistId);

}

foreach (var artist in artists)

{

artist.Artworks = artworks.Where(a => a.ArtistId == artist.Id).ToList();

}

// Связываем выставки и произведения искусства

foreach (var exhibition in exhibitions)

{

exhibition.Artworks = artworks

.Where(a => exhibition.ArtworkIds.Contains(a.Id))

.ToList();

}

// Связываем посетителей и покупки

foreach (var visitor in visitors)

{

visitor.Purchases = sales

.Where(s => s.BuyerId == visitor.Id)

.ToList();

}

// Связываем билеты с выставками и посетителями

foreach (var ticket in tickets)

{

ticket.Exhibition = exhibitions.FirstOrDefault(e => e.Id == ticket.ExhibitionId);

ticket.Visitor = visitors.FirstOrDefault(v => v.Id == ticket.VisitorId);

}

// Связываем продажи с произведениями и покупателями

foreach (var sale in sales)

{

sale.Artwork = artworks.FirstOrDefault(a => a.Id == sale.ArtworkId);

sale.Buyer = visitors.FirstOrDefault(v => v.Id == sale.BuyerId);

}

// Связываем аренды с произведениями и арендаторами

foreach (var rental in rentals)

{

rental.Artwork = artworks.FirstOrDefault(a => a.Id == rental.ArtworkId);

rental.Renter = visitors.FirstOrDefault(v => v.Id == rental.RenterId);

}

}

// === СПИСКИ ===

public List<Artwork> GetAllArtworks() => \_artworkRepo.GetAll();

public List<Artist> GetAllArtists() => \_artistRepo.GetAll();

public List<Exhibition> GetAllExhibitions() => \_exhibitionRepo.GetAll();

public List<Visitor> GetAllVisitors() => \_visitorRepo.GetAll();

public List<Ticket> GetAllTickets() => \_ticketRepo.GetAll();

public List<Sale> GetAllSales() => \_saleRepo.GetAll();

public List<Rental> GetAllRentals() => \_rentalRepo.GetAll();

// === ЗАПРОСЫ ===

// Список произведений искусства в наличии

public List<Artwork> GetAvailableArtworks()

{

LoadReferences();

return \_artworkRepo.GetAll()

.Where(a => a.Status == ArtworkStatus.InGallery)

.ToList();

}

// Выставки, запланированные на ближайший месяц

public List<Exhibition> GetUpcomingExhibitions()

{

LoadReferences();

DateTime today = DateTime.Today;

DateTime oneMonthAhead = today.AddMonths(1);

return \_exhibitionRepo.GetAll()

.Where(e => e.StartDate <= oneMonthAhead && e.EndDate >= today)

.ToList();

}

// Самые популярные художники по количеству проданных работ

public List<(Artist Artist, int SoldCount)> GetPopularArtistsBySales()

{

LoadReferences();

var artworks = \_artworkRepo.GetAll();

var sales = \_saleRepo.GetAll();

var artists = \_artistRepo.GetAll();

var soldArtworks = sales.Select(s => s.ArtworkId).ToList();

return artists

.Select(artist => (

Artist: artist,

SoldCount: soldArtworks.Count(id =>

artworks.FirstOrDefault(a => a.Id == id)?.ArtistId == artist.Id)

))

.OrderByDescending(item => item.SoldCount)

.ToList();

}

/// <summary>

/// Общая выручка от билетов и продаж картин

/// </summary>

/// <returns>Выручка с билетов, Выручка с картин, Общая выручка</returns>

public (decimal TicketRevenue, decimal SalesRevenue, decimal TotalRevenue) GetTotalRevenue()

{

LoadReferences();

decimal ticketRevenue = \_ticketRepo.GetAll()

.Where(t => t.Status != TicketStatus.Cancelled)

.Sum(t => t.Price);

decimal salesRevenue = \_saleRepo.GetAll()

.Sum(s => s.Amount);

return (ticketRevenue, salesRevenue, ticketRevenue + salesRevenue);

}

// Произведения, находящиеся в аренде

public List<(Artwork Artwork, Rental Rental)> GetRentedArtworks()

{

LoadReferences();

var rentals = \_rentalRepo.GetAll();

var artworks = \_artworkRepo.GetAll();

var currentRentals = rentals

.Where(r => r.EndDate >= DateTime.Today)

.ToList();

return currentRentals

.Select(rental => (

Artwork: artworks.FirstOrDefault(a => a.Id == rental.ArtworkId),

Rental: rental

))

.Where(tuple => tuple.Artwork != null)

.ToList()!;

}

// === ОПЕРАЦИИ ===

// Добавление нового произведения искусства

public Artwork AddArtwork(string title, string artistId, int year, string genre,

string description, decimal estimatedValue)

{

var artist = \_artistRepo.GetById(artistId);

if (artist == null)

{

throw new Exception("Художник не найден");

}

var artwork = new Artwork

{

Title = title,

ArtistId = artistId,

Year = year,

Genre = genre,

Description = description,

EstimatedValue = estimatedValue,

Status = ArtworkStatus.InGallery

};

\_artworkRepo.Add(artwork);

\_artworkRepo.SaveChanges();

// Обновляем список работ художника

artist.ArtworkIds.Add(artwork.Id);

\_artistRepo.Update(artist);

\_artistRepo.SaveChanges();

return artwork;

}

// Продажа произведения искусства

public Sale SellArtwork(string artworkId, string buyerId, decimal amount)

{

var artwork = \_artworkRepo.GetById(artworkId);

if (artwork == null)

{

throw new Exception("Произведение не найдено");

}

if (artwork.Status != ArtworkStatus.InGallery)

{

throw new Exception("Это произведение недоступно для продажи");

}

var buyer = \_visitorRepo.GetById(buyerId);

if (buyer == null)

{

throw new Exception("Покупатель не найден");

}

// Создаем запись о продаже

var sale = new Sale

{

ArtworkId = artworkId,

BuyerId = buyerId,

Date = DateTime.Now,

Amount = amount

};

// Обновляем статус произведения

artwork.Status = ArtworkStatus.Sold;

\_artworkRepo.Update(artwork);

\_artworkRepo.SaveChanges();

// Добавляем продажу в историю покупок посетителя

buyer.PurchaseIds.Add(sale.Id);

\_visitorRepo.Update(buyer);

\_visitorRepo.SaveChanges();

// Сохраняем информацию о продаже

\_saleRepo.Add(sale);

\_saleRepo.SaveChanges();

return sale;

}

// Передача произведения в аренду

public Rental RentArtwork(string artworkId, string renterId, DateTime startDate,

DateTime endDate, decimal cost)

{

var artwork = \_artworkRepo.GetById(artworkId);

if (artwork == null)

{

throw new Exception("Произведение не найдено");

}

if (artwork.Status != ArtworkStatus.InGallery)

{

throw new Exception("Это произведение недоступно для аренды");

}

var renter = \_visitorRepo.GetById(renterId);

if (renter == null)

{

throw new Exception("Арендатор не найден");

}

// Создаем запись об аренде

var rental = new Rental

{

ArtworkId = artworkId,

RenterId = renterId,

StartDate = startDate,

EndDate = endDate,

Cost = cost

};

// Обновляем статус произведения

artwork.Status = ArtworkStatus.Rented;

\_artworkRepo.Update(artwork);

\_artworkRepo.SaveChanges();

// Сохраняем информацию об аренде

\_rentalRepo.Add(rental);

\_rentalRepo.SaveChanges();

return rental;

}

// Бронирование билетов на выставку

public Ticket BookTicket(string exhibitionId, string visitorId, DateTime visitDate)

{

var exhibition = \_exhibitionRepo.GetById(exhibitionId);

if (exhibition == null)

{

throw new Exception("Выставка не найдена");

}

if (visitDate < exhibition.StartDate || visitDate > exhibition.EndDate)

{

throw new Exception("Дата посещения должна быть в период проведения выставки");

}

var visitor = \_visitorRepo.GetById(visitorId);

if (visitor == null)

{

throw new Exception("Посетитель не найден");

}

// Создаем билет

var ticket = new Ticket

{

ExhibitionId = exhibitionId,

VisitorId = visitorId,

VisitDate = visitDate,

Price = exhibition.TicketPrice,

Status = TicketStatus.Reserved

};

// Добавляем запись о посещении

visitor.VisitHistory.Add($"{exhibition.Title} ({visitDate.ToShortDateString()})");

\_visitorRepo.Update(visitor);

\_visitorRepo.SaveChanges();

// Сохраняем билет

\_ticketRepo.Add(ticket);

\_ticketRepo.SaveChanges();

return ticket;

}

// Добавление нового художника

public Artist AddArtist(string fullName, string country, string lifeYears, string style)

{

var artist = new Artist

{

FullName = fullName,

Country = country,

LifeYears = lifeYears,

Style = style

};

\_artistRepo.Add(artist);

\_artistRepo.SaveChanges();

return artist;

}

// Добавление новой выставки

public Exhibition AddExhibition(string title, DateTime startDate, DateTime endDate,

string location, List<string> artworkIds, decimal ticketPrice)

{

// Проверяем существование произведений

foreach (var artworkId in artworkIds)

{

if (\_artworkRepo.GetById(artworkId) == null)

{

throw new Exception($"Произведение с ID {artworkId} не найдено");

}

}

var exhibition = new Exhibition

{

Title = title,

StartDate = startDate,

EndDate = endDate,

Location = location,

ArtworkIds = artworkIds,

TicketPrice = ticketPrice

};

\_exhibitionRepo.Add(exhibition);

\_exhibitionRepo.SaveChanges();

return exhibition;

}

// Регистрация нового посетителя

public Visitor AddVisitor(string fullName, string contactInfo)

{

var visitor = new Visitor

{

FullName = fullName,

ContactInfo = contactInfo

};

\_visitorRepo.Add(visitor);

\_visitorRepo.SaveChanges();

return visitor;

}

}

}

using System.Text;

using System.Text.Encodings.Web;

using System.Text.Json;

using System.Text.Json.Serialization;

using System.Text.Unicode;

namespace ArtGallery.Repository

{

public class JsonRepository<T> : IRepository<T> where T : ArtGallery.Models.BaseEntity

{

private readonly string \_filePath;

private List<T> \_entities;

public JsonRepository(string fileName)

{

string directory = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Data");

if (!Directory.Exists(directory))

{

Directory.CreateDirectory(directory);

}

\_filePath = Path.Combine(directory, fileName);

\_entities = LoadFromFile();

}

private List<T> LoadFromFile()

{

if (!File.Exists(\_filePath))

{

return new List<T>();

}

string json = File.ReadAllText(\_filePath);

if (string.IsNullOrWhiteSpace(json))

{

return new List<T>();

}

var options = new JsonSerializerOptions

{

PropertyNameCaseInsensitive = true,

Converters = { new JsonStringEnumConverter() }

};

return JsonSerializer.Deserialize<List<T>>(json, options) ?? new List<T>();

}

public List<T> GetAll()

{

return \_entities;

}

public T? GetById(string id)

{

return \_entities.FirstOrDefault(e => e.Id == id);

}

public void Add(T entity)

{

\_entities.Add(entity);

}

public void Update(T entity)

{

int index = \_entities.FindIndex(e => e.Id == entity.Id);

if (index != -1)

{

\_entities[index] = entity;

}

}

public void Delete(string id)

{

var entity = GetById(id);

if (entity != null)

{

\_entities.Remove(entity);

}

}

public void SaveChanges()

{

var options = new JsonSerializerOptions

{

WriteIndented = true,

Converters = { new JsonStringEnumConverter() },

Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping

};

string json = JsonSerializer.Serialize(\_entities, options);

File.WriteAllText(\_filePath, json, Encoding.UTF8);

}

}

}