

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Цифровая платформа для формирования
онлайн-пространства искусства и
взаимодействия художников с аудиторией**

Студент _____ Киселева Марина Евгеньевна
(подпись, дата) (ФИО полностью)

Руководитель _____ Балашова И.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Киселева Марина Евгеньевна гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Цифровая платформа для формирования онлайн-пространства
искусства и взаимодействия художников с аудиторией

Тема утверждена приказом ПГУ № 1583/о от "30" декабря 2025 г.

2. Руководитель работы _____

3. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Назначение: цифровая платформа для формирования онлайн пространства
Искусства и взаимодействия художников с аудиторией.

2) Среда выполнения: Windows 10.

Языковые средства разработки: C#, TypeScript, UML.

Инструментальные средства: Visual Studio 2022, Visual Studio Code,
Visual Paradigm 17.0, Docker, Kafka, MongoDB, Swagger, Draw.io, Ocelot.

Технологии разработки: ООП, микросервисная архитектура.

3) Основные функции приложения:

а) регистрация и авторизация;

б) поиск и просмотр картин;

в) добавление картины в избранное и в корзину;

г) оформление заказа;

д) просмотр профиля художника и возможность коммуникации с ним;

е) фильтрация и сортировка картин;

ж) публикация, редактирование и управление картинами.

4. Перечень подлежащих разработке вопросов

- 1) Анализ предметной области и постановка задачи;
- 2) Проектирование;
- 3) Реализация;
- 4) Контроль качества программного обеспечения;
- 5) Оформление пояснительной записки.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 120 листов, 51 рисунок, 21 таблицу, 26 использованных источников и 4 приложения.

РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ, ИЗОБРАЗИТЕЛЬНОЕ ИСКУССТВО, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, КОНТЕЙНЕРИЗАЦИЯ МИКРОСЕРВИСОВ, ДОКУМЕНТО-ОРИЕНТИРОВАННАЯ БД, TYPESCRIPT, REACT.

Объект исследования – процесс монетизации результатов творческой деятельности.

Цель выпускной квалификационной работы – проектирование и разработка цифровой платформы для формирования онлайн-пространства искусства и взаимодействия художников с аудиторией.

Технологии разработки – разработка серверной части выполнялась в среде Microsoft Visual Studio 2022 на языке C# с использованием платформы ASP.NET Core на базе микросервисной архитектуры, с использованием брокера Apache Kafka, NoSQL базы данных MongoDB, API-шлюза Ocelot. Серверная часть контейнеризирована при помощи Docker. Клиентская часть реализована на TypeScript и React. Моделирование бизнес-процессов выполнено с использованием UML.

Среда исполнения – Windows 10, веб-браузер.

Результаты работы – спроектирована и разработана цифровая платформа для формирования онлайн-пространства искусства и взаимодействия художников с аудиторией.

					ПГУ 09.03.04 – 08BP222.16 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Киселева М.Е.			Цифровая платформа для формирования онлайн-пространства искусства и взаимодействия художников с аудиторией. Пояснительная записка	Лит.	Лист	Листов
Пров.		Балашова И.Ю.					4	120
Реценз.						Группа 22ВП2		
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку цифровой платформы для формирования онлайн-пространства искусства и взаимодействия художников с аудиторией	9
1.1 Анализ предметной области и постановка задачи	9
1.2 Анализ программного обеспечения аналогичного назначения	10
1.2.1 Веб-сайт «Artnet»	10
1.2.2 Веб-приложение «Uprise Art»	12
1.2.3 Веб-приложение «Bizar».....	13
1.2.4 Веб-приложение «Saatchi art»	15
1.2.5 Сравнение программных средств аналогичного назначения	16
1.3 Анализ функциональных требований	20
1.4 Анализ нефункциональных требований	23
1.5 Планирование разработки и оценка бюджета	25
2 Проектирование цифровой платформы для формирования онлайн пространства искусства и взаимодействия художников с аудиторией	27
2.1 Архитектура приложения	27
2.2 Проектирование структур данных	30
2.3 Проектирование классов программных средств	33
2.4 Проектирование пользовательского интерфейса	36
3 Реализация цифровой платформы для формирования онлайн пространства искусства и взаимодействия художников с аудиторией	39
3.1 Реализация базы данных	39
3.2 Реализация серверной части приложения	46
3.3 Реализация клиентской части приложения.....	50
4 Контроль качества программного обеспечения	64
4.1 Выбор режима тестирования и метрик кода.....	64

4.2	Тестирование программных средств формирования онлайн пространства искусства и взаимодействия художников с аудиторией.....	66
4.2.1	Тестирование пользовательской части цифровой платформы.....	66
4.2.2	Тестирование серверной части цифровой платформы.....	73
4.3	Оценка качества программного кода.....	75
4.3.1	Метрика SLOC.....	75
4.3.2	Метрика Чепина	76
4.3.3	Объектно-ориентированная метрика WMC	78
	Заключение	80
	Список использованных источников	82
	Приложение А (справочное) Глоссарий	85
	Приложение Б (обязательное) Листинг приложения	88
	Приложение В (рекомендуемое) Графическая часть	101
	Приложение Г (рекомендуемое) Результаты тестирования	109

Введение

В современном мире цифровые технологии развиваются с большой скоростью. Их влияние сказывается на сфере искусства, и с каждым годом оно лишь увеличивается. С одной стороны, с помощью современных технологий, в том числе искусственного интеллекта, человек без художественных навыков может сам создать и отредактировать картину под любой свой запрос, что негативно влияет на сферу искусства, поскольку у художника станет меньше возможностей для формирования собственной аудитории. С другой стороны, технологии открывают новые возможности для роста и развития творческих людей, включая представление, распространение и продвижение своих работ, а также поиск покупателей.

Тем не менее, несмотря на стремительное развитие цифровых технологий, взаимодействие между художником и его потенциальной аудиторией в настоящий момент осложнено в связи с целым рядом факторов, главным из которых является отсутствие удобной и надежной площадки. Из-за этого художники часто сталкиваются с трудностями при продвижении своих работ и поиске новых клиентов. А пользователи, в свою очередь, не знают, где можно посмотреть работы разных художников, найти картину под свой запрос, а также безопасно совершить сделку по покупке картины.

Существующие платформы не решают всего перечня задач, являются малоизвестными и непопулярными вследствие устаревшего дизайна и примитивного функционала. Часть таких платформ направлена исключительно на продажу картин, что сводит к минимуму взаимодействие художника с его аудиторией. Оставшиеся платформы предоставляют возможность для общения автора с аудиторией, но не имеют инструментов для монетизации художниками своих работ. Многие из существующих платформ являются зарубежными, отечественные аналоги составляют малую часть от подобных приложений.

Целью выпускной квалификационной работы является разработка единой цифровой платформы формирования онлайн-пространства искусства и

взаимодействия художников с аудиторией, которая сочетает в себе функционал, позволяющий художникам публиковать свои работы, продвигать их, набирать постоянную аудиторию и монетизировать свое творчество, а пользователям – искать подходящие картины, приобретать их, формировать персональные рекомендации, а также взаимодействовать с художником напрямую для уточнения информации или заказа картины.

Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать предметную область и выделить основные требования к разрабатываемой системе;
- проанализировать существующее программное обеспечение аналогичного назначения;
- выявить функциональные и нефункциональные требования к разрабатываемой платформе;
- спроектировать удобное и надежное хранение данных, учитывая требования безопасности и скорости;
- спроектировать архитектуру серверной части, обеспечивающей масштабируемость;
- спроектировать клиентское приложение с удобным интерфейсом;
- разработать систему, учитывая выделенные требования;
- осуществить контроль качества разработанной платформы.

При решении поставленных задач цель выпускной квалификационной работы, заключающаяся в разработке единой цифровой платформы формирования онлайн-пространства искусства и взаимодействия художников с аудиторией, будет успешно достигнута.

1 Анализ предметной области и постановка задачи на разработку цифровой платформы для формирования онлайн-пространства искусства и взаимодействия художников с аудиторией

1.1 Анализ предметной области и постановка задачи

Сфера изобразительного искусства постоянно расширяется, включая в себя новые направления, возникающие благодаря технологиям, а также новым художественным открытиям. Сам термин изобразительное искусство можно определить, как один из видов художественного творчества, который непосредственно визуально воспроизводит предметы, формы и явления окружающего мира или ассоциативные напоминания о них [1]. К изобразительному искусству относятся живопись, скульптура, графика, фотоискусство.

В современных реалиях художнику недостаточно заниматься творчеством и развиваться в этом ремесле, ему необходимо также показывать свои работы людям и рассказывать о себе и своем творческом пути, набирать собственную аудиторию, которая принесет ему потенциальных клиентов. Без постоянного продвижения своих работ на онлайн-площадках художник не сможет развиваться и монетизировать результаты своей творческой деятельности, получать признание и популярность. Пользователи, в свою очередь, не знают популярных и удобных, а главное надежных площадок, предоставляющих удобный и простой функционал для поиска произведений искусства, подбора их под свой бюджет, изучения работ различных авторов и взаимодействия с художниками напрямую.

Также, существует большой разрыв между художником и его потенциальной аудиторией. Платформы, которая направлена на создание пространства, где художник сможет взаимодействовать с аудиторией напрямую, и которая включает в себя такие процессы, как: публикация художественных работ для широкой аудитории, продвижение автора работ, формирование интереса у аудитории к творчеству, взаимодействие с потенциальными

покупателями и поиск подходящих художественных работ, пока не существует. Создание такой платформы в онлайн-среде позволит художнику увеличить охваты пользователей, то есть выйти за пределы своего локального круга клиентов и представить свои работы более широкой аудитории. Уменьшение разрыва между художниками и аудиторией и налаживание тесной связи между ними являются необходимыми в современной цифровой среде, где взаимодействие между авторами работ и клиентами напрямую влияет на творческий процесс.

Исходя из всего описанного выше, основной задачей является создание программных средств, которые будут удобны как художникам, так и простым пользователям, заинтересованным в художественных работах, и которые наладят связь между авторами и клиентами. Такое программное средство позволит художнику набрать свою постоянную аудиторию и клиентуру, что положительно скажется на развитии творческого потенциала автора, общаться с аудиторией напрямую, что подскажет автору о желаниях его клиентов. Пользователи, благодаря такой платформе, получают удобный и безопасный способ для покупки предметов изобразительного искусства, а также для связи с понравившимся художником с целью сделать заказ на изготовление картины.

1.2 Анализ программного обеспечения аналогичного назначения

1.2.1 Веб-сайт «Artnet»

Веб-сайт «Artnet» представляет собой международную онлайн-платформу для торговли предметами искусства. Проект был основан в 2008 году, однако идея о создании подобной площадки и первые варианты реализации программных средств появились в 1989 году. Основной направленностью данного сайта являются услуги по предоставлению пользователям информации об онлайн-аукционах. То есть предметы изобразительного искусства нельзя купить так, как совершают покупки в интернет-магазинах: чтобы приобрести понравившееся произведение, необходимо посетить онлайн-аукцион и предложить цену выше конкурентов.

Интерфейс приложения представлен на рисунке 1.

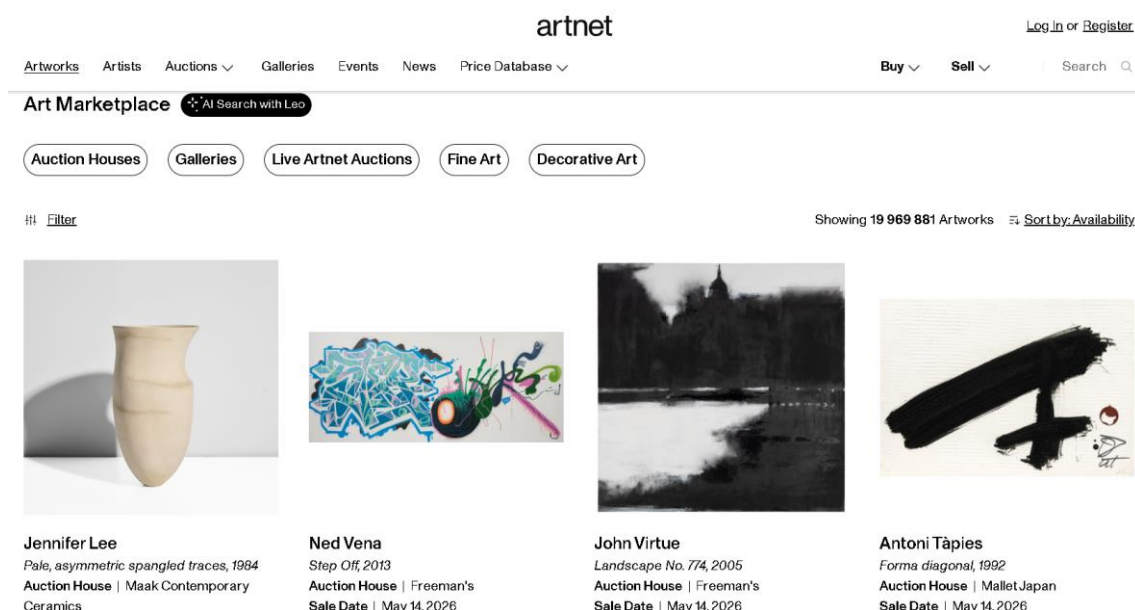


Рисунок 1 – Интерфейс приложения «Artnet»

У данного приложения есть несколько преимуществ, среди которых:

- доступ к большой базе работ как начинающих художников, так и известных мастеров;
- наличие собственного новостного портала, обновляющего информацию в сфере искусства ежедневно;
- широкая база данных, содержащая записи о продажах произведений искусств.

Стоит также отметить, что веб-приложение достаточно популярно, что, безусловно, влияет на потенциальную аудиторию художника.

Но у данного сайта имеется ряд недостатков, к которым относятся:

- доступ к самой площадке является платным;
- доступ к расширенному функционалу для художников, включающего аналитику и статистику проведенных аукционов, является дорогостоящим;
- интерфейс веб-приложения, несмотря на его минимализм, неудобный и не интуитивный, особенно для новых пользователей, из-за запутанности навигации на сайте.

Резюмируя, можно сказать, что площадка «Artnet» вследствие своей универсальности, а именно наличия различных инструментов для работы художников, включая сервис для онлайн-аукционов, новостной журнал о развитии сферы искусства, онлайн-галереи, является очень популярной во всем мире. Пользователей привлекает как прозрачность платформы – наличие базы данных о всех совершенных продажах предметов искусства, так и широкий выбор различных работ и авторов.

1.2.2 Веб-приложение «Uprise Art»

«Uprise Art» – это современная платформа, нацеленная на продажу уникальных предметов изобразительного искусства. Платформа помогает художникам представлять свои работы, продавать их, а также дает обратную связь и консультации по вопросам выстраивания карьеры. Приложение очень популярно среди начинающих художников, так как его политика в отношении авторов лояльна – платформа не сосредоточена на продвижение известных и популярных художников, а дает шанс каждому автору найти свою аудиторию.

Интерфейс приложения представлен на рисунке 2.

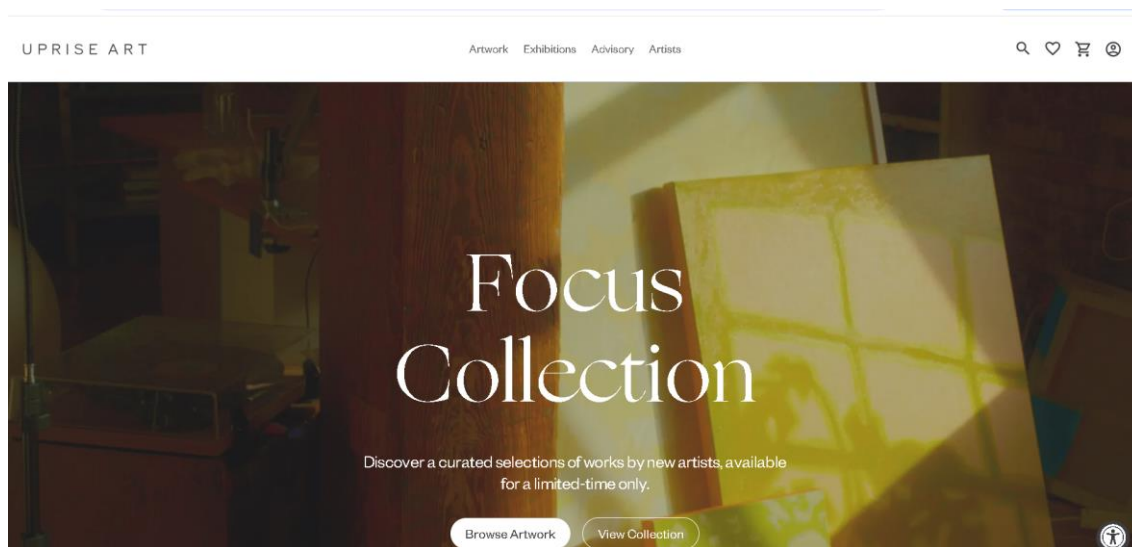


Рисунок 2 – Интерфейс приложения «Uprise Art»

Стоит также отметить, что на данной платформе отсутствует сортировка и распределение по категориям предметов искусства, так как целью

создательницы данной площадки было приближение онлайн-платформы, представляющей собой галерею, к обычной, традиционной галерее, где все работы выставлены вместе. Когда пользователь заходит на сайт, он видит большое разнообразие различных работ, не отсортированных по какому-либо критерию. Эта особенность отличает «Uprise Art» от остальных онлайн-галерей.

Среди преимуществ рассматриваемого веб-приложения, можно выделить:

- наличие уникальных произведения изобразительного искусства, существующих в единственном экземпляре;
- удобная фильтрация представленных картин по ценовому диапазону, что упрощает поиск картины;
- минималистичный и привлекающий внимание дизайн;
- возможность купить картину через сайт без участия в аукционе.

Кроме того, в приложении имеется возможность связаться с консультантом, который поможет подобрать необходимую картину, ответит на вопросы пользователя касательно площадки, автора или работы, а также даст рекомендации по покупке картины с учетом интерьера помещения.

У данной платформы также есть ряд недостатков, среди которых:

- цены на произведения искусства довольно высокие, вследствие чего платформа менее популярна, чем конкуренты;
- на площадке не предусмотрен возврат купленных произведений.

В результате приложение привлекает пользователей благодаря удобному и понятному интерфейсу, привлекательному дизайну, отличию от конкурентов, возможности быстро приобрести понравившуюся и уникальную картину.

1.2.3 Веб-приложение «Bizar»

Веб-приложение «Bizar» разработано в 2021 году в России и представляет собой маркетплейс современного искусства и дизайна, который помогает работам художников найти своих владельцев [2]. Интерфейс приложения представлен на рисунке 3.

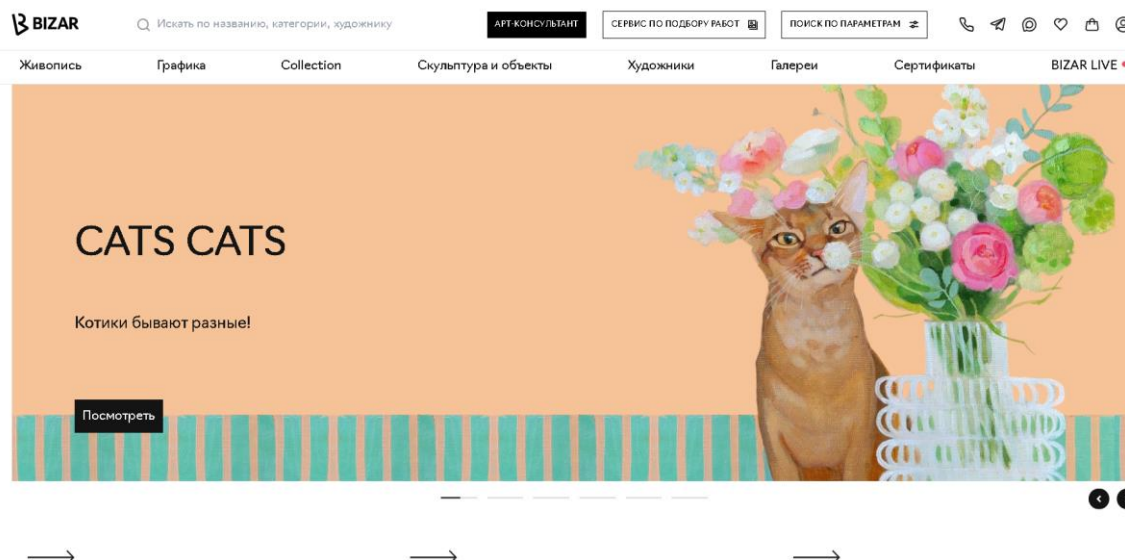


Рисунок 3 – Интерфейс приложения «Bizar»

Данная платформа обладает целым рядом достоинств, среди которых:

- привлекающий внимание дизайн;
- большой выбор картин, разделенных по разным категориям;
- современные экраны страниц;
- удобный поиск картин с большим количеством фильтров.

Также в сервисе присутствует возможность связаться с одним из дизайнеров, курирующих платформу, который бесплатно поможет найти картину, подходящую под все указанные параметры, проконсультирует покупателя, при помощи графических редакторов, наложит картину на фотографии интерьера покупателя, чтобы пользователи смогли более быстро подобрать подходящую под интерьер картину.

Веб-сайт не лишен и недостатков, главным из которых, для конечного пользователя, является неудовлетворительная работа сайта, а именно:

- медленная загрузка сайта;
- невозможность открыть некоторые страницы;
- медленный отклик на действия пользователя;
- неудобные ссылки – переходы на другие страницы.

Эти недостатки являются очень существенными, так как конечный пользователь не захочет оставаться на платформе, которая работает очень

медленно и не позволяет перейти на некоторые из своих страниц. На впечатление пользователя о сайте влияет и скорость отклика системы на любые действия пользователя. «Bizar» оставляет негативное впечатление от работы из-за своей нестабильности. Потенциальные покупатели могут отказаться совершать покупку через данную онлайн-галерею вследствие ее низкой надежности.

1.2.4 Веб-приложение «Saatchi art»

Веб-приложение «Saatchi art» представляет собой ведущую зарубежную онлайн-галерею, которая была основана двадцать лет назад. Эта платформа обладает широким функционалом для продажи оригинальных произведений искусства и репродукций. Площадка очень популярна среди иностранных художников. Выбор художников и их работ отличается большим разнообразием.

Особенностью платформы является отсутствие необходимости в регистрации для оформления заказа и осуществления покупки. Это делает процесс покупки быстрее, но с другой стороны, такая возможность может привести к нежелательным последствиям в случае ошибки транзакции или иных финансовых проблем.

Интерфейс приложения представлен на рисунке 4.

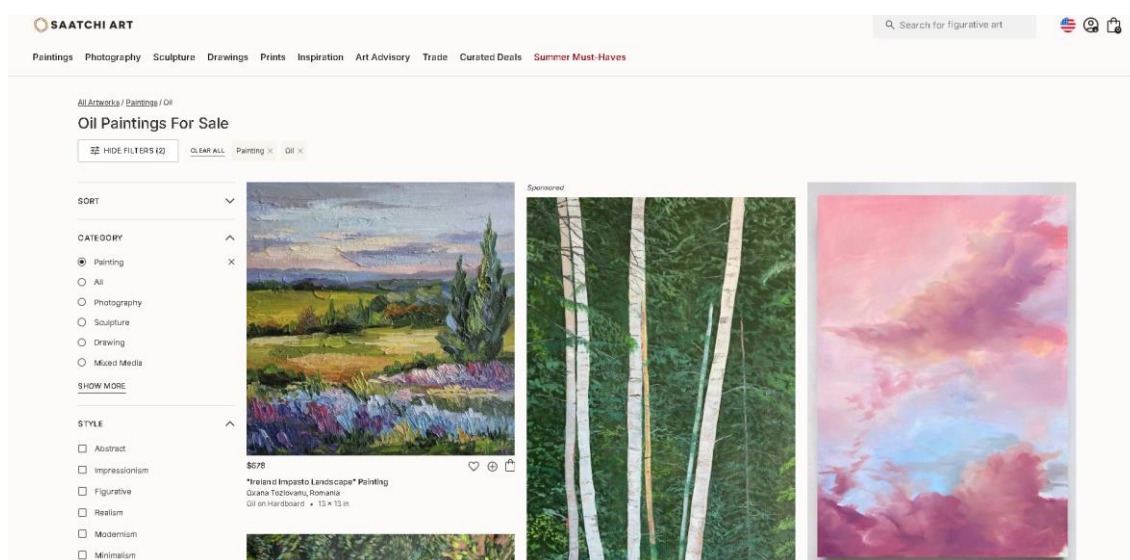


Рисунок 4 – Интерфейс приложения «Saatchi art»

Стоит отметить, что в приложении присутствует услуга консультирования по предметам искусств. Пользователь заполняет и отправляет анкету, дизайнеры в течение недели подбирают подходящие работы, подходящие под запросы клиента и присылают их.

Веб-приложение обладает следующими преимуществами:

- большая база различных работ и авторов;
- бесплатные консультации с дизайнерами для подбора подходящей работы;
- популярность среди зарубежной аудитории и художников.

Несмотря на все свои преимущества, «Saatchi art» не предоставляет пользователям и авторам инструментов для прямого взаимодействия и формирования комфортного и удобного онлайн-пространства.

Приложение также обладает рядом недостатков, среди которых:

- простой дизайн, не привлекающий внимания;
- сложные, неинтуитивные переходы между страницами;
- приложение нацелено только на зарубежную аудиторию и не адаптировано для работы в России.

Таким образом, хотя «Saatchi art» и обладает рядом существенных достоинств и выделяется среди своих конкурентов, но из-за перечисленных недостатков не может стать главным приложением на рынке. Тем не менее платформа остается одной из ведущих онлайн-галерей.

1.2.5 Сравнение программных средств аналогичного назначения

Для проведения сравнительного анализа существующих на рынке приложений было выявлено несколько критериев для их оценки, а именно: наличие механизма для прямых продаж, инструменты для продвижения автора работ, инструменты взаимодействия авторов и пользователей, удобство поиска и фильтрации работ, наглядность представления работ на странице, привлекательность интерфейса, высокая скорость и стабильность работы платформы, локализация и адаптация для рынка России.

Критерий «Наличие механизма для прямых продаж» подразумевает реализацию в программном средстве возможности для пользователя приобрести предмет искусства через сайт без участия в аукционе.

Критерий «Инструменты для продвижения автора работ» представляет собой наличие таких функций в приложении, как: портфолио автора, рекомендации работ автора на главной странице, а также рекомендации работ автора в качестве похожих к просматриваемой пользователем работы.

Критерий «Инструменты взаимодействия авторов и пользователей» – это наличие в приложении функционала, позволяющего: пользователю связаться с художником напрямую, без сторонних сервисов; пользователю посетить страницу автора и просмотреть его портфолио и информацию о нем; пользователю и художнику ставить отметки «нравится» работам других художников.

Смысл критерия «Удобство поиска и фильтрации работ» заключается в наличии удобной системы распределения работ по разным категориям и возможности поиска картин в выбранной категории с учетом различных, заданных пользователем фильтров. Также критерий подразумевает возможность поиска по ключевым словам – по названию картины, по описанию картины, по автору картины.

Критерий «Наглядность представления работ на странице» состоит из реализации удобной и наглядной карточки картины, отражающей ключевые ее черты, а также из понятной и удобной структуры самой страницы, содержащей карточки с информацией о картинах. Этот критерий определяет удобство использования сайта пользователем.

Критерий «Привлекательность интерфейса» отражает эстетическую составляющую интерфейса. Важно, чтобы внешний вид веб-приложения для формирования онлайн-пространства искусства был привлекательным для пользователя, эстетически приятным.

Критерий «Высокая скорость и стабильность работы платформы» состоит из двух факторов: скорость отклика системы после любого действия

пользователя; способность системы корректно функционировать без сбоев, зависаний, потери данных, а также обрабатывать различные ошибки [3]. Скорость отклика системы в норме составляет до трех секунд. При времени отклика менее одной секунды – скорость работы сайта отличная. При времени отклика более трех секунд пользователь уже ощущает задержку.

Критерий «Локализация и адаптация для рынка России» – это возможность системы корректно работать на разных рынках, в том числе на рынке России, с возможностью использования местной валюты.

Для каждого из критериев используется двухбалльная шкала, где:

- 0 баллов – «полностью не соответствует критерию»;
- 1 балл – «полностью соответствует критерию».

Результаты сравнительного анализа представлены в таблице 1.

Таблица 1 – Сравнительный анализ аналогичных программных средств

Критерии	Artnet	Uprise Art	Bizar	Saatchi art
Наличие механизма для прямых продаж	0	1	1	1
Инструменты для продвижения автора работ	0	1	1	1
Инструменты взаимодействия авторов и пользователей	0	1	1	0
Удобство поиска и фильтрации работ	0	0	1	1
Наглядность представления работ на странице	1	1	1	1
Привлекательность интерфейса	0	1	1	0
Высокая скорость и стабильность работы платформы	1	1	0	1
Локализация и адаптация для рынка России	0	0	1	0

Проанализировав полученные результаты, можно сделать вывод, что ни одно из рассмотренных приложений не соответствует всем перечисленным критериям.

Лидером среди рассмотренных приложений является веб-приложение «Bizar». Стоит отметить, что «Bizar», хоть и содержит в себе разнообразный функционал, позволяющий художникам публиковать и продвигать свои работы, а пользователям находить интересные произведения искусства и покупать их, но из-за нестабильности работы системы, проявляющейся в невозможности перейти на некоторые страницы, очень медленном отклике и периодических ошибках в работе системы, платформа не может занять первенство на рынке.

На втором месте находится зарубежное приложение «Uprise Art». Приложение достаточно популярно, при этом оно уникально и не похоже на своих конкурентов. Его особенности выделяются на фоне остального рынка, что одновременно является преимуществом, так как платформа привлекает внимание, и недостатком, так как не все привычные пользователю функции и сценарии доступны на платформе.

Третье место занимает приложение «Saatchi art». Площадка является популярной и содержит огромную базу различных работ и авторов. Однако приложение не реализует функционала для взаимодействия пользователя с художником, что негативно сказывается на впечатлении пользователей.

На последнем месте расположилось приложение «Artnet». Несмотря на то, что площадка обладает популярностью и известностью во всем мире, она не реализует некоторых ключевых функций, вследствие чего на данный момент теряет свою аудиторию.

В результате сравнения приложений аналогичного назначения приложения, реализующего все перечисленные критерии, не было выявлено. Было принято решение разработать собственные программные средства, которые будут реализовывать лучшие решения из сравниваемых приложений.

1.3 Анализ функциональных требований

После проведения исследования и анализа предметной области, поиска и детального рассмотрения существующих программных средств аналогичного назначения, а также их сравнения, были выявлены основные функциональные требования к разрабатываемой системе:

- регистрация и авторизация пользователей;
- редактирование данных для входа;
- редактирование данных своего аккаунта;
- просмотр профилей художников;
- удаление своего профиля;
- поиск пользователей по текстовому запросу;
- получение всех картин;
- получение всех картин одного художника;
- просмотр одной картины;
- получение списка похожих картин на выбранную;
- поиск и фильтрация картин;
- создание записи о новой картине;
- изменение существующей записи о картине;
- удаление картины;
- добавление картины в избранное;
- удаление картины из избранного;
- получение всех картин, находящихся в избранном;
- получение всех картин, находящихся в корзине;
- добавление новой картины в корзину.

В соответствии с выявленными функциональными требованиями была разработана диаграмма вариантов использования UML. Разработанная диаграмма вариантов использования для программных средств для формирования онлайн-пространства и взаимодействия художников с аудиторией представлена на рисунке 5.

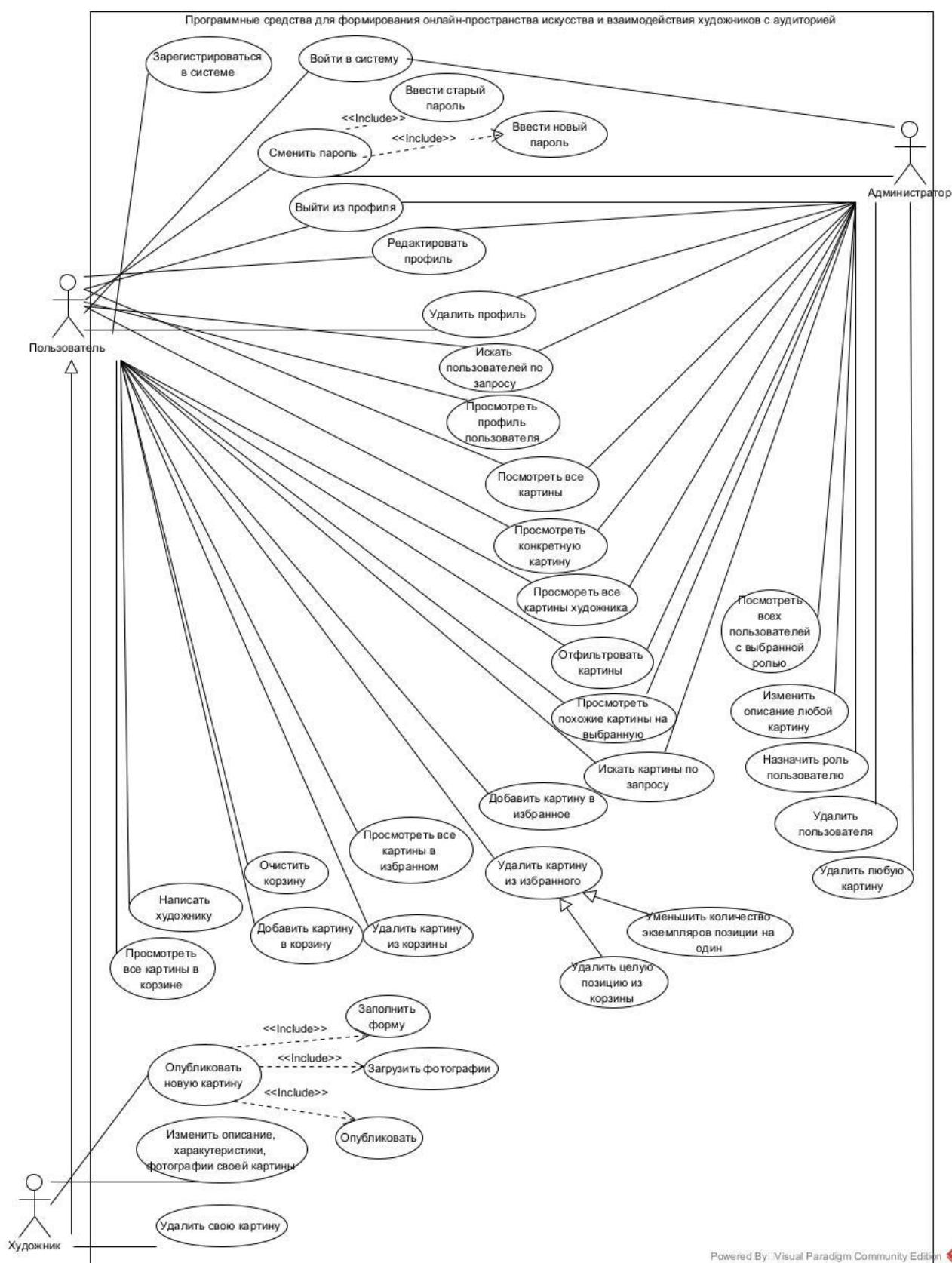


Рисунок 5 – Диаграмма вариантов использования для разрабатываемых программных средств

Диаграмма вариантов использования позволяет четко определить границы системы и понять потребности пользователей, фокусируясь на том, что система должна делать, а не на деталях ее внутренней реализации или архитектуры [4].

На разработанной диаграмме представлено несколько актеров, в числе которых: пользователь, который имеет базовый уровень доступа к функциям системы, художник – пользователь, имеющий доступ к функциям системы базового уровня, а также свой собственный функционал, администратор – пользователь, управляющий системой и базами данных. Стоит отметить, что художник состоит в отношении обобщения с пользователем.

На диаграмме также используются несколько видов связей: отношение ассоциации, обозначающее связь между актером и отдельным вариантом использования; отношение включения, позволяющее включить поведение одного прецедента в поток другого, и отношение обобщения. Обобщение прецедентов выносит поведение, общее для одного или более прецедентов, в родительский прецедент [5].

В таблице 2 приведен сценарий для варианта использования «Войти в систему».

Таблица 2 – Сценарий выполнения варианта использования «Войти в систему»

Наименование	Войти в систему
ID	1
Краткое описание	Пользователь аутентифицируется в системе
Действующие лица	Пользователь
Предусловие	Пользователь не был аутентифицирован
Основной поток	1. Пользователь переходит на страницу входа в приложении; 2. Пользователь вводит данные для входа в свой аккаунт (логин и пароль); 3. Пользователь нажимает на кнопку «Войти»; 4. Пользователь получает доступ к функционалу системы в соответствии со своей ролью.
Постусловие	Пользователь вошел в свой аккаунт

В таблице 3 приведен сценарий для варианта использования «Редактировать профиль».

Таблица 3 – Сценарий выполнения варианта использования «Редактировать профиль»

Наименование	Редактировать профиль
ID	2
Краткое описание	Пользователь редактирует личную информацию на странице профиля
Действующие лица	Аутентифицированный пользователь
Предусловие	Пользователь вошел в систему
Основной поток	1. Пользователь переходит на страницу своего профиля; 2. Пользователь нажимает на «Редактировать»; 3. Пользователь заполняет все обязательные поля формы; 4. Пользователь сохраняет данные при помощи кнопки «Сохранить».
Постусловие	Данные аккаунта отредактированы

В таблице 4 приведен сценарий для варианта использования «Назначить роль пользователю».

Таблица 4 – Сценарий выполнения варианта использования «Назначить роль пользователю»

Наименование	Назначить роль пользователю
ID	3
Краткое описание	Администратор назначает роль выбранному пользователю
Действующие лица	Администратор
Предусловие	Администратор вошел в свой аккаунт
Основной поток	1. Администратор переходит на страницу админ-панели; 2.Администратор выбирает пользователя, которому необходимо изменить роль; 3. Администратор выбирает роль; 4. Администратор подтверждает изменение.
Постусловие	Роль изменена

В результате анализа функциональных требований для разрабатываемой системы были выделены варианты использования, которые будут реализованы в ходе разработки приложения. Для части вариантов использования были составлены спецификации, подробно описывающие выполнение конкретного варианта использования.

1.4 Анализ нефункциональных требований

Нефункциональные требования, они же свойства архитектуры, играют решающую роль в успехе приложения [3]. Следовательно, разрабатываемая

система должна удовлетворять ряду нефункциональных требований, для успешной работы:

- масштабируемость;
- сопровождаемость;
- авторизуемость;
- удобство и простота использования.

Масштабируемость – это работоспособность системы при росте числа пользователей или ужесточении уровня технических потребностей.

Сопровождаемость – это легкость внесения в систему изменений и улучшений.

Авторизуемость – это требования мер безопасности, позволяющие открывать пользователям доступ только к строго определенным функциям приложения.

Удобство и простота использования – это уровень подготовки, необходимый пользователям для достижения намеченных ими целей с помощью приложения.

Реализация всех перечисленных нефункциональных требований обеспечит стабильную и надежную работу системы.

Разработка серверной части цифровой платформы производится на языке C# в среде разработки Microsoft Visual Studio 2022 на платформе ASP.NET Core. Для хранения данных используется документо-ориентированная база данных MongoDB. В качестве API-шлюза выступает Ocelot. Для развертывания серверной части используется Docker.

Для разработки клиентского приложения используется язык Typescript, библиотека React, а также сервер для локальной разработки Vite. Взаимодействие клиентской и серверной части осуществляется при помощи REST API.

1.5 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета являются критически важными этапами для успешной реализации программного продукта [6]. Для управления проектом использовалась программа Microsoft Office Project 2007, которая позволяет планировать разработку, отслеживать выполнение поставленных задач и контролировать ресурсы на разработку.

После внесения в программу всех необходимых данных о проекте была составлена диаграмма Ганта, которая отображает ключевые этапы разработки программного обеспечения. Диаграмма Ганта – это график работ по проекту, который строится в виде таблицы с этапами. Диаграмма состоит из двух осей – вертикальной, со списком задач, и горизонтальной, со сроками [7].

Полученная диаграмма Ганта представлена на рисунке 6.

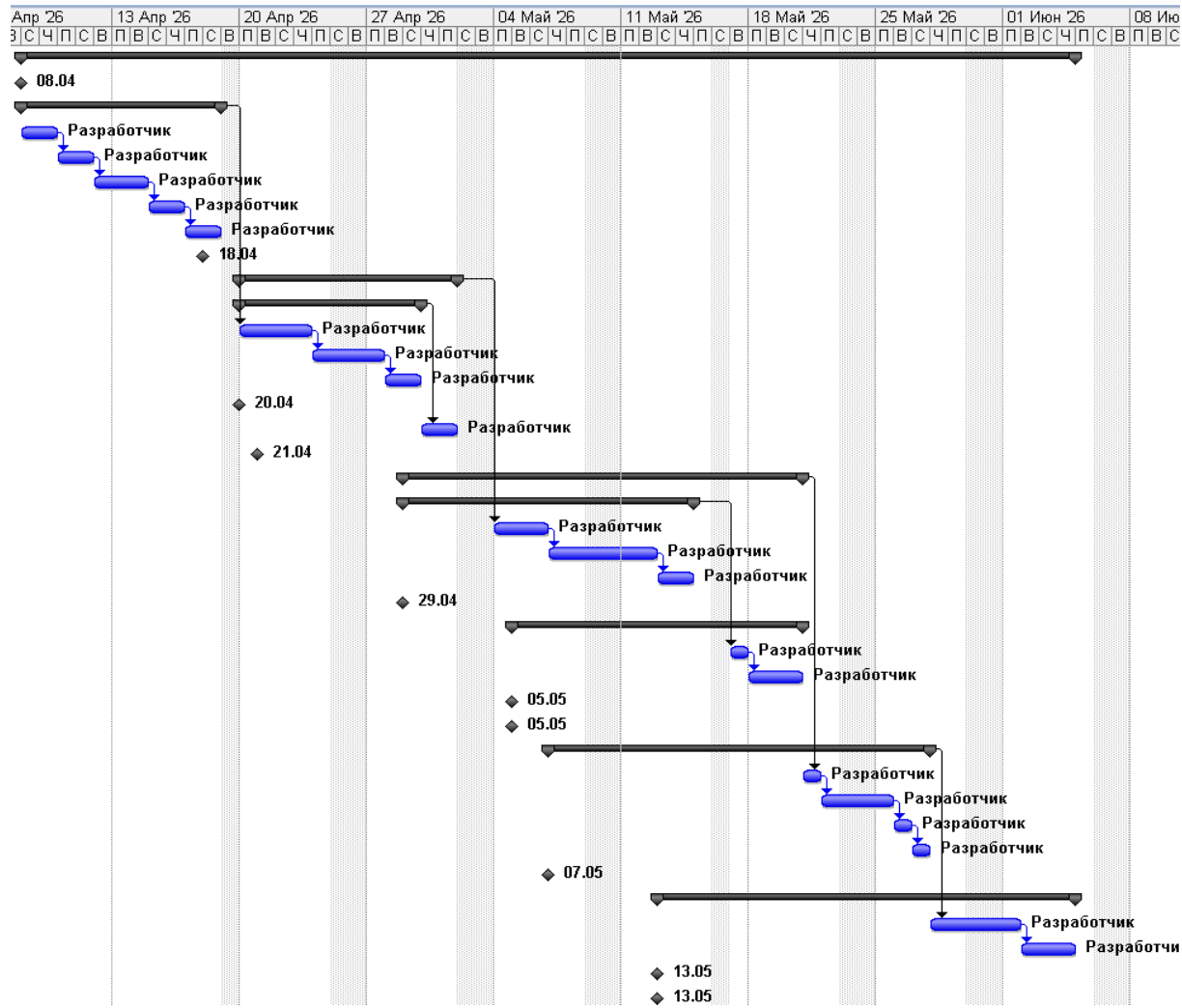


Рисунок 6 – Диаграмма Ганта для разрабатываемого программного обеспечения

На диаграмме показаны основные этапы разработки, а также планируемые даты их выполнения и трудовой ресурс, занимающийся выполнением всех задач.

В ходе планирования работ над проектом было выявлено двадцать задач. Датой начала работы над проектом, в соответствии с заданием, является восьмое апреля, а датой завершения проекта, по полученной диаграмме, – четвертое июня. Список поставленных задач представлен на рисунке 7.

	Название задачи	Длительность	Начало	Окончание
1	Разработка приложения для формирования онлайн-пространства и взаимодействия художников с аудиторией	46 дней?	Ср 08.04.26	Чт 04.06.26
2	Начало проекта	0 дней	Ср 08.04.26	Ср 08.04.26
3	Анализ программных средств для формирования онлайн-пространства искусства и взаимодействия художников с аудиторией	11 дней?	Ср 08.04.26	Сб 18.04.26
4	Анализ предметной области разрабатываемых программных средств и постановка задачи	2 дней	Ср 08.04.26	Чт 09.04.26
5	Анализ и сравнение программного обеспечения аналогичного назначения	2 дней	Пт 10.04.26	Сб 11.04.26
6	Анализ функциональных требований к разрабатываемой системе	3 дней	Вс 12.04.26	Вт 14.04.26
7	Анализ нефункциональных требований	2 дней?	Ср 15.04.26	Чт 16.04.26
8	Осуществление планирования разработки и оценки бюджета	2 дней?	Пт 17.04.26	Сб 18.04.26
9	Конец анализа программных средств	0 дней	Сб 18.04.26	Сб 18.04.26
10	Проектирование приложения для формирования онлайн-пространства и взаимодействия художников с аудиторией	10 дней?	Пн 20.04.26	Пт 01.05.26
11	Проектирование серверной части	8 дней?	Пн 20.04.26	Ср 29.04.26
12	Проектирование архитектуры серверного приложения	4 дней?	Пн 20.04.26	Чт 23.04.26
13	Проектирование структур данных	2 дней?	Пт 24.04.26	Пн 27.04.26
14	Проектирование программных средств для формирования онлайн-пространства и взаимодействия художников с аудиторией	2 дней?	Вт 28.04.26	Ср 29.04.26
15	Конец проектирования серверной части	0 дней	Пн 20.04.26	Пн 20.04.26
16	Проектирование клиентской части	2 дней	Чт 30.04.26	Пт 01.05.26
17	Конец проектирования приложения	0 дней	Вт 21.04.26	Вт 21.04.26
18	Разработка приложения для формирования онлайн-пространства и взаимодействия художников с аудиторией	17 дней?	Ср 29.04.26	Ср 20.05.26
19	Разработка серверного приложения	12 дней?	Ср 29.04.26	Чт 14.05.26
20	Разработка архитектуры	3 дней?	Пн 04.05.26	Ср 06.05.26
21	Реализация микросервисов серверного приложения	4 дней?	Чт 07.05.26	Вт 12.05.26
22	Контейнеризация микросервисов серверного приложения	2 дней?	Ср 13.05.26	Чт 14.05.26
23	Конец разработки серверного приложения	0 дней	Ср 29.04.26	Ср 29.04.26
24	Разработка клиентского приложения	13 дней?	Вт 05.05.26	Ср 20.05.26
25	Разработка архитектуры клиентского приложения	1 день?	Вс 17.05.26	Вс 17.05.26
26	Реализация клиентского приложения	3 дней?	Пн 18.05.26	Ср 20.05.26
27	Конец разработки клиентского приложения	0 дней	Вт 05.05.26	Вт 05.05.26
28	Конец разработки приложения	0 дней	Вт 05.05.26	Вт 05.05.26
29	Контроль качества программного обеспечения	16 дней?	Чт 07.05.26	Ср 27.05.26
30	Выбор метрик программного кода	1 день?	Чт 21.05.26	Чт 21.05.26
31	Проведение функционального тестирования	2 дней?	Пт 22.05.26	Пн 25.05.26
32	Проведение модульного тестирования	1 день	Вт 26.05.26	Вт 26.05.26
33	Вычисление метрик программного кода	1 день?	Ср 27.05.26	Ср 27.05.26
34	Конец контроля качества программного обеспечения	0 дней	Чт 07.05.26	Чт 07.05.26
35	Составление отчета	18 дней?	Ср 13.05.26	Чт 04.06.26
36	Оформление пояснительной записки	3 дней?	Чт 28.05.26	Пн 01.06.26
37	Составление глоссария и приложений	3 дней?	Вт 02.06.26	Чт 04.06.26
38	Конец составления отчета	0 дней	Ср 13.05.26	Ср 13.05.26
39	Конец проекта	0 дней	Ср 13.05.26	Ср 13.05.26

Рисунок 7 – Список поставленных задач

Для оценки финансовых затрат на реализацию приложения для трудового ресурса, разработчика, была назначена ставка, сумма которой пятьсот рублей за час работы.

Подводя итог, можно сделать вывод, что для разработки программных средств для формирования онлайн-пространства искусства и взаимодействия художников с аудиторией необходимо 180000 рублей, а общая занятость трудовых ресурсов составляет 360 часов, что соответствует 45 рабочим дням.

2 Проектирование цифровой платформы для формирования онлайн-пространства искусства и взаимодействия художников с аудиторией

2.1 Архитектура приложения

После проведения анализа функциональных и нефункциональных требований к разрабатываемым программным средствам в качестве архитектуры серверного приложения была выбрана микросервисная архитектура.

Архитектурный стиль микросервисов – это подход, при котором единое приложение строится, как набор небольших сервисов, каждый из которых работает в собственном процессе и общается с остальными при помощи механизмов API.

Микросервисная архитектура обладает такими отличительными чертами, как [3]:

- распределенность – каждый микросервис запускается в своем собственном процессе;
- ограниченный контекст – в каждом микросервисе моделируется предметная область или рабочий процесс, то есть каждый микросервис включает в себя все необходимо для работы в составе приложения;
- наличие API-уровня, представляющего из себя компонент, стоящий между потребителями системы и самой системой.

При такой архитектуре каждый микросервис имеет свою зону ответственности, то есть выполняет задачи в пределах своей предметной области. Каждый из проектируемых микросервисов должен выполнять только одну функцию, то есть микросервисы должны удовлетворять принципу единой ответственности.

Микросервисная архитектура позволяет легко масштабировать систему, также она повышает отказоустойчивость системы. В случае перебоев в работе одного микросервиса, работа остальных микросервисов не будет нарушена, так как микросервисы работают независимо друг от друга. Кроме того, архитектура

микросервисов позволяет легко сопровождать систему, то есть вносить изменения в существующий код или добавлять новые функции.

Клиентское приложение, в свою очередь, обеспечивает отображение данных и служит основным интерфейсом для получения команд от пользователей, которые впоследствии отправляются на сервер посредством REST API [8].

Для наглядного представления архитектуры разрабатываемой системы была составлена диаграмма компонентов UML.

Диаграмма компонентов отражает особенности физического представления программного средства, другими словами, она определяет структуру приложения при помощи установки связей между отдельными его компонентами, в роли которых могут быть как сервисы, так и базы данных [9].

Диаграмма компонентов для разрабатываемых программных средств представлена на рисунке В.1 в приложении В.

На разработанной диаграмме компонентов рассмотрены все компоненты системы, а именно:

- компонент «Клиентское приложение»;
- компонент «ApiGateway (Api-шлюз)»;
- компонент «IdentityService (микросервис авторизации)»;
- компонент «UserService (микросервис пользователей)»;
- компонент «RoleService (микросервис ролей)»;
- компонент «ArtService (микросервис картин)»;
- компонент «CartService (микросервис корзины)»;
- компонент «FavoritesService (микросервис избранного)»;
- компонент «ArtPlatform_Identity_DB (MongoDB)»;
- компонент «ArtPlatform_Users_DB (MongoDB)»;
- компонент «ArtPlatform_Roles_DB (MongoDB)»;
- компонент «ArtPlatform_Art_DB (MongoDB)»;
- компонент «ArtPlatform_Cart_DB (MongoDB)»;

- компонент «ArtPlatform_Favorites_DB (MongoDB)»;
- компонент «Kafka».

Компонент «Клиентское приложение» представляет собой клиентскую часть системы, отвечающую за пользовательский интерфейс.

Компонент «ApiGateway (Api-шлюз)» служит единой точкой входа в приложение, то есть API-шлюз принимает запросы от клиентского приложения и перенаправляет их в соответствующие микросервисы, проверив при этом токен.

Взаимодействие между клиентской частью и серверной осуществляется при помощи REST API.

Компонент «IdentityService (микросервис авторизации)» представляет собой микросервис авторизации, который отвечает за хранение и обработку учетных данных пользователей – логина и пароля. Микросервис должен реализовывать функционал регистрации, авторизации, смены пароля и выдачи Bearer-токена. Структура токена должна включать в себя поле «Роль» для того, чтобы система понимала, пользователь с каким уровнем доступа к функционалу взаимодействует с платформой, и показывала только тот функционал, который доступен текущему пользователю.

Компонент «UserService (микросервис пользователей)» – это микросервис, отвечающий за профили пользователей и обеспечивающий функционал для работы с данными профиля, а также реализует функционал для управления самим профилем – удалением или созданием.

Компонент «RoleService (микросервис ролей)» представляет собой микросервис, отвечающий за роли пользователей. Обеспечивает функционал по установке, смене и удалению ролей у пользователей.

Компонент «ArtService (микросервис картин)» содержит в себе микросервис по работе с картинками, предоставляющий инструменты для создания записи новой картины, изменению, удалению, а также поиску и фильтрации картин.

Компонент «CartService (микросервис корзины)» – это микросервис, который предоставляет функционал для работы с корзиной – добавление в корзину, удаление из корзины, а также просмотр содержимого корзины.

Компонент «FavoritesService (микросервис избранного)» представляет микросервис, позволяющий добавлять картины в избранное, удалять из избранного и просматривать свое избранное.

Компоненты «ArtPlatform_Identity_DB (MongoDB)», «ArtPlatform_Users_DB (MongoDB)», «ArtPlatform_Roles_DB (MongoDB)», «ArtPlatform_Art_DB (MongoDB)», «ArtPlatform_Cart_DB (MongoDB)», «ArtPlatform_Favorites_DB (MongoDB)» представляют собой шесть баз данных, каждая из которых взаимодействует с определенным микросервисом. Микросервисы используют протокол Mongo wire protocol для взаимодействия с базами данных.

Компонент «Kafka» – это брокер сообщений Apache Kafka, который используется для организации выполнения асинхронных операций. Брокер содержит в себе разделы, которые называются топиками. Микросервисы подписываются на нужные топики и прослушивают их. Когда в топик попадает новое сообщение, микросервисы, которые подписаны на этот топик, смогут получить данное сообщение в удобный для них момент и обработать информацию из этого сообщения, параллельно с выполнением остальных задач.

Взаимодействие между всеми компонентами осуществляется по протоколу HTTP, за исключением взаимодействия микросервисов с базами данных и брокером Kafka. Кроме того, клиентское приложение, API-шлюз и микросервисы взаимодействуют через REST API, который подразумевает вызов заранее определенных точек доступа.

2.2 Проектирование структур данных

Проектирование структуры базы данных представляет собой важный этап разработки, определяющий эффективность и удобство работы с данными.

Необходимо разграничивать этапы проектирования структур данных и их реализации и не объединять их в один [10].

Важность данного этапа определяется тем, что именно архитектура и хранение данных в базе данных определяет скорость поиска и обработки информации, а также удобство хранения ее. Проектирование структур данных позволяет заранее избежать возможной избыточности данных, что упрощает работу с программными средствами и делает проще их сопровождение. Так как архитектура серверной части представляет собой микросервисную архитектуру, то проектирование баз данных имеет большое значение в реализации системы, поскольку важно заранее разграничить зоны ответственности каждой базы данных и выявить, как они связаны друг с другом.

В качестве базы данных для спроектированной микросервисной архитектуры будет использоваться NoSQL база данных, а именно документо-ориентированная система `mongoDB`, которая была выбрана по следующим причинам:

- простое масштабирование, благодаря отсутствию сложных логических связей между документами;
- высокая скорость чтения и обновления данных;
- хорошая производительность.

Основным преимуществом использования `noSQL` базы данных является ее эффективность в распределенной среде.

Особенностью использования `mongoDB` является заранее не регламентированная структура данных, то есть, если в реляционных базах данных четко определены схемы таблиц, то в `noSQL` базах данных структура заранее не определяется, данные могут храниться в том виде, в котором они обрабатываются в приложении.

Концептуальная модель базы данных описывает бизнес-процессы, работающие над данными, без указания технических особенностей реализации. Такая модель представления данных разрабатывается для отображения структуры хранения данных в общем виде [11].

Для более наглядного представления распределенной между микросервисами базы данных была построена диаграмма пакетов в нотации UML, представленная на рисунке 9. Пакет, в нотации UML – это общий механизм организации элементов в группы [12].

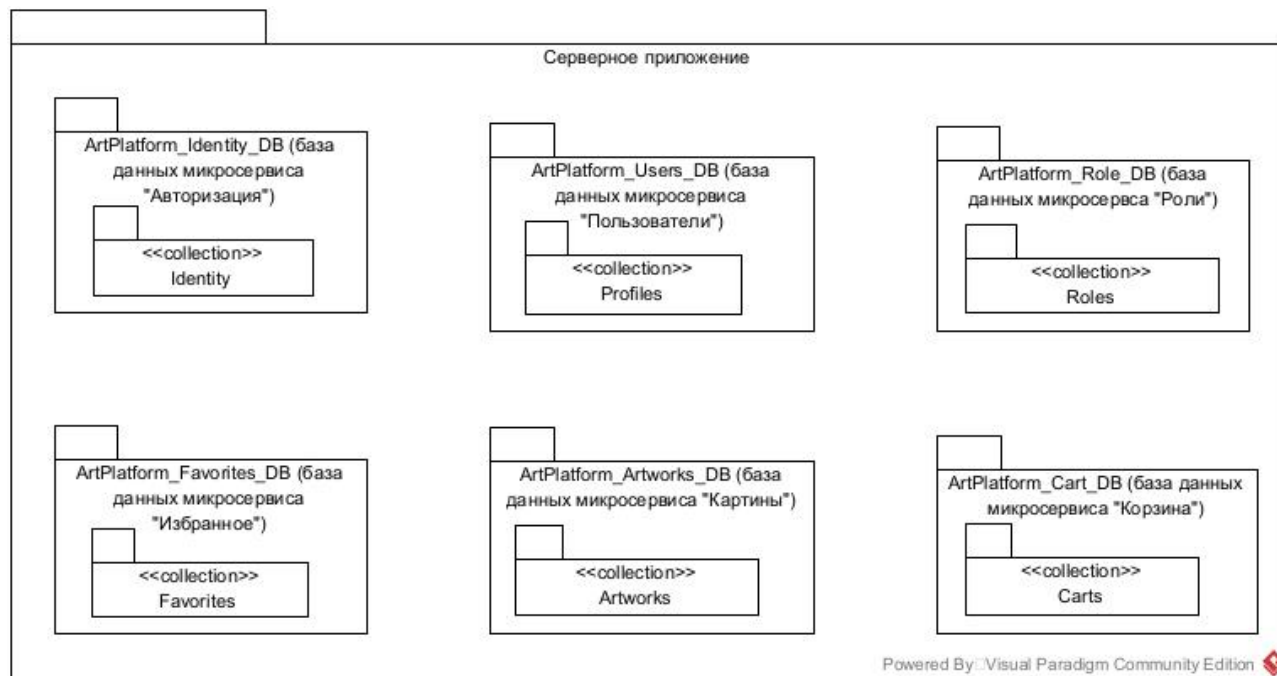


Рисунок 9 – Диаграмма пакетов для баз данных

Стоит отметить, что каждому из спроектированных микросервисов соответствует собственная база данных, которая не знает ничего о существовании остальных. Такая архитектура более гибкая и масштабируемая. Помимо этого, использование такой архитектуры позволяет вносить правки в код одного микросервиса независимо, то есть без влияния на остальные части системы.

У каждой из представленных баз данных есть своя коллекция, в которой будут храниться документы. Микросервисы общаются между собой при помощи асинхронных сообщений, благодаря которым, базы данных смогут обновлять свои документы, в случае необходимости. Помимо этого, два микросервиса, а именно микросервис «Избранное» и микросервис «Корзина» обращаются к микросервису «Картины» по HTTP для получения информации о картинах из

базы данных «ArtPlatform_Artworks_DB». Впоследствии эта информация сохраняется в базах данных микросервисов «Избранное» и «Корзина».

Для более детального представления проектируемых баз данных были разработаны диаграммы классов для каждого микросервиса. Диаграммы классов для микросервисов «Авторизация», «Пользователи», «Роли», «Картины», «Избранное», «Корзина» представлены в приложении В на рисунках В.2, В.3, В.4, В.5, В.6 и В.7 соответственно.

В каждом микросервисе выделен отдельный класс, который предназначен для подключения к базе данных и задания ее конфигурации. При помощи этого класса также происходит создание индексов, которые уменьшают время выполнения различных операций с данными.

Стоит отметить, что класс, предназначенный для доступа к данным, оперирует доменными сущностями, вокруг которых строится предметная область каждого микросервиса. То есть класс, необходимый для доступа к базе данных в качестве структуры хранимых данных использует сущности, то есть классы со стереотипом «Entity».

Данный класс, в свою очередь, используют сервисы, которые содержат в себе всю бизнес-логику и которые работают с данными.

Спроектированный концептуальный уровень структуры данных позволяет увидеть структуру хранения данных в общем виде. Разработанная архитектура базы данных позволит системе быстро оперировать данными и легко масштабироваться.

2.3 Проектирование классов программных средств

В ходе проектирования серверной части цифровой платформы для формирования онлайн-пространства искусства и взаимодействия художников с аудиторией были спроектированы диаграммы классов для каждого из микросервисов, представленные в приложении В.

Диаграмма классов – это один из видов диаграмм унифицированного языка моделирования, которая используется для визуального представления структуры

программной системы: состава классов, их атрибутов, методов и взаимосвязей между ними [13].

Для всех микросервисов был выбран единый стиль – многослойная архитектура, состоящая из таких элементов, как:

- слой представления – классы с постфиксом «Controller», обеспечивающие работу с API;
- слой сервисов – классы с постфиксом «Service», отвечающие за бизнес-логику;
- слой доступа к данным;
- слой базы данных;
- слой (асинхронного) обмена сообщениями.

Схема используемой во всех микросервисах архитектуры представлена на рисунке на рисунке 10.

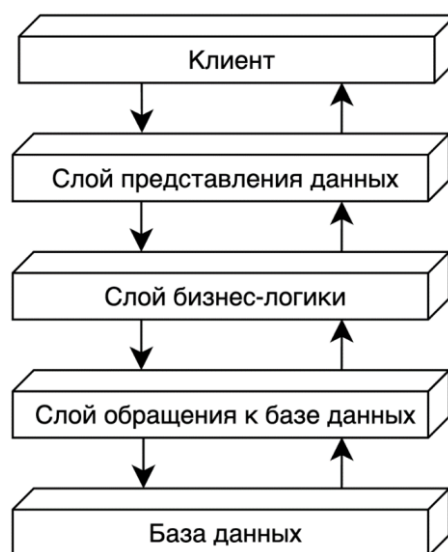


Рисунок 10 – Схема многослойной архитектуры микросервисов

При такой организации микросервисов запрос пользователя (Клиент), в первую очередь, направляется к слою представления данных (Controller), который занимается обработкой пользовательских запросов, проверкой переданных данных и перенаправлением их к следующему слою – слою бизнес-логики.

Слой бизнес-логики, который представляет из себя сервисы (Service), обрабатывающие данные, переданные слоем представления данных (Controller). На данном этапе определяются бизнес-правила, согласно которым будут обрабатываться различные запросы, другими словами, на данном слое определяется основная логика работы приложения. Данный слой может общаться со слоем обращения к базе данных для получения конкретных данных. Помимо этого, слой бизнес-логики оперирует сущностями (Entity) – то есть основными объектами, являющимися ядром предметной области микросервиса.

В слое обращения к базе данных определяются основные настройки для подключения к базе данных.

Помимо этого, микросервисы общаются между собой при помощи брокера сообщений. На диаграммах классов отображены классы с постфиксом «Event», которые представляют модели для событий, публикуемых в брокер. Также два сервисных класса «KafkaProducerService» и «KafkaConsumerService» представляют отправителя сообщений в брокер и получателя сообщений соответственно.

Для спроектированных классов микросервиса «Картины» в таблице 5 приведено описание функционального назначения.

Таблица 5 – Описание функционального назначения классов микросервиса «Картины»

Класс	Назначение
Artwork	Класс, представляющий доменную сущность, вокруг которой строится предметная область микросервиса «Картины»
ArtworksController	Класс, представляющий контроллер Web API, обеспечивающий взаимодействие клиента и микросервиса
IArtworkService	Интерфейс, определяющий контракты бизнес-логики микросервиса «Картины»
ArtworkService	Класс, реализующий интерфейс IArtworkService, содержащий основную бизнес-логику микросервиса
KafkaProducerService	Сервис, предназначенный для публикации сообщений в брокер
KafkaConsumerService	Сервис, предназначенный для получения и обработки сообщений из брокера

Продолжение таблицы 5

Класс	Назначение
MongoDBContext	Класс, обеспечивающий подключение к базе данных и оперирующий сущностями
CreateArtworkRequest	Класс, представляющий объект передачи данных (DTO), используемый при публикации новой картины
ArtworkFilterRequest	Класс, представляющий объект передачи данных (DTO), используемый при задания фильтров при фильтрации картин
ArtDeletedEvent	Класс, представляющий модель для создания события, публикуемого при удалении картины
UserDeletedEvent	Класс, представляющий модель для создания события, получаемого при удалении пользователя

Похожим функциональным назначением обладают классы других микросервисов.

Таким образом, была спроектирована архитектура разрабатываемого серверного приложения в целом, и каждого из микросервисов в частности.

2.4 Проектирование пользовательского интерфейса

На этапе проектирования пользовательского интерфейса закладывается основная структура интерфейса программных средств, с которым будут взаимодействовать пользователи. Важно, чтобы структура пользовательского интерфейса была понятная, простая и удобная. Помимо этого, дизайн интерфейса должен обладать эстетической привлекательностью и соответствовать представлениям пользователя о дизайне площадки, ориентированной на предметы изобразительного искусства. Дизайн пользовательского интерфейса – это этап разработки программного обеспечения, который реализуется после глубокого изучения целевой аудитории программных средств и особенностей предметной области [14].

В соответствии с результатами анализа предметной области, выявленными требованиями к разрабатываемому приложению, а также рассмотренными приложениями аналогичного назначения был спроектирован пользовательский интерфейс, который включает в себя такие страницы, как:

- страница авторизации;
- страница регистрации;

- страница аккаунта текущего пользователя;
- главная страница;
- страница каталога;
- страница картины;
- страница избранного;
- страница корзины;
- страница публичного профиля художника;
- страница публикации новой картины;
- страница редактирования картины;
- страница админ-панели;
- страница оформления заказа.

Экраны авторизации и регистрации должны обеспечивать простую и быструю регистрацию. Для этих экранов должна быть обязательно реализована валидация данных, введенных пользователем.

Для страницы аккаунта текущего пользователя основными требованиями является простота представления информации, а также удобное расположение основных функциональных блоков.

Главная страница является «лицом» разрабатываемых программных средств, поэтому важно тщательно спроектировать ее, так как пользователь, в первую очередь, оценивает именно дизайн и структуру главной страницы. На этом экране будут располагаться основные блоки – преимущества платформы, популярные картины и художники, основные категории, а также счетчики с количеством опубликованных картин на платформе и зарегистрированных художников.

Экран каталога является немаловажным элементом пользовательского интерфейса, так как большую часть времени пользователь будет работать именно с ним. Важно, чтобы этот экран предоставлял пользователю различные возможности: поиск картины по названию, автору или категории; фильтрации работ, сортировка работ по различным критериям.

Страницы избранного и корзины также должны быть комфортными в использовании и интуитивно понятными.

Экран публичного профиля художника должен сочетать в себе удобство представления основной информации о художнике, наличие структурированного портфолио данного художника, а также контактные данные автора, чтобы можно было с ним связаться.

Страница картины является одним из главных элементов, так как грамотное проектирование данной страницы позволит привлечь пользователей. Поэтому основными требованиями к данной странице является наличие блока с характеристиками картины, возможность перехода к странице автора, возможность просмотра всех изображений картины, наличие блоков с другими работами художника и похожими работами.

Страница оформления заказа должна предоставлять пользователю удобную для заполнения форму с валидацией пользовательского ввода для предотвращения ошибок.

Экраны публикации новой картины и редактирования уже опубликованной доступны только художнику. Данные экраны должны предоставлять понятную форму для заполнения.

Админ-панель представляет собой страницу, доступную только администратору, с которой он сможет управлять всеми пользователями платформы, а также редактировать или удалять выбранные картины. Для комфортной работы необходимо, чтобы админ-панель была понятной и структурированной.

Помимо функциональной части пользовательский интерфейс должен обладать приятным и привлекательным для пользователей дизайном.

3 Реализация цифровой платформы для формирования онлайн-пространства искусства и взаимодействия художников с аудиторией

3.1 Реализация базы данных

Физическая модель базы данных – это детальное описание ее реализации. В отличие от реляционных баз данных, в которых данные хранятся в виде заранее регламентированных схем таблиц, в NoSQL базах данных данные хранятся в виде документов.

Документо-ориентированная база данных – это хранилище «ключ-значение», где значение является документом, то есть в самом объекте содержится описание его структуры [15]. Другими словами, каждый документ, хранящийся в базе данных, представляет собой полностью изолированную информационную единицу.

В контексте `mongoDB`, такой документ обладает произвольным набором атрибутов, то есть полей, и может быть представлен в виде JSON. На рисунке 11 приведены структура документа в виде схемы для базы данных микросервиса «Пользователи».

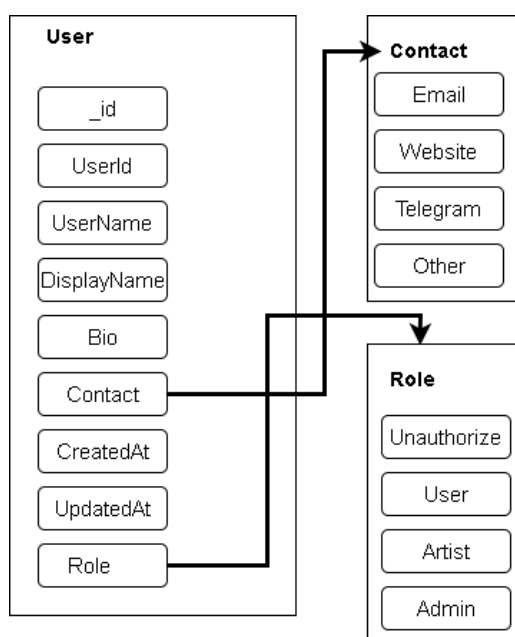


Рисунок 11 – Схема документа для базы данных микросервиса «Пользователи»

Документ имеет два поля, которые являются составными объектами – «Contact» и «Role». В таблице 6 приведены атрибуты для полей документа.

Таблица 6 – Строки документа базы данных микросервиса «Пользователи»

Значение поля	Описание	Тип
_id	Уникальный идентификатор документа	String
UserId	Уникальный идентификатор пользователя	String
UserName	Уникальное имя пользователя (логин)	String
DisplayName	Имя пользователя	String
Bio	Описание профиля	String
Contact	Вложенный объект – контактная информация	Object
CreatedAt	Время создания аккаунта	Date
UpdatedAt	Время обновления аккаунта	Date
Role	Роль пользователя	String

Для более наглядного представления структуры документа, из базы данных был получен один документ. На рисунке 12 приведена структура документа в виде JSON.

```
{
  "_id": "aa8be4dd-b548-44cd-8713-06e9e8d8940f",
  "UserId": "00000000-0000-0000-0000-000000000000",
  "UserName": "admin",
  "DisplayName": "admin",
  "Bio": "",
  "Contact": {
    "Email": null,
    "Website": null,
    "Telegram": null,
    "OtherContact": null
  },
  "CreatedAt": "2026-05-16T05:42:34.830Z",
  "UpdatedAt": "2026-05-16T05:42:35.240Z",
  "Role": "Admin"
}
```

Рисунок 12 – Документ базы данных «ArtPlatform_Users_DB» в виде JSON

На приведенном рисунке представлена не только структура документа, но и значения для каждого из ключей строк документа.

На рисунке 13 представлена схема документа для базы данных микросервиса «Авторизация».

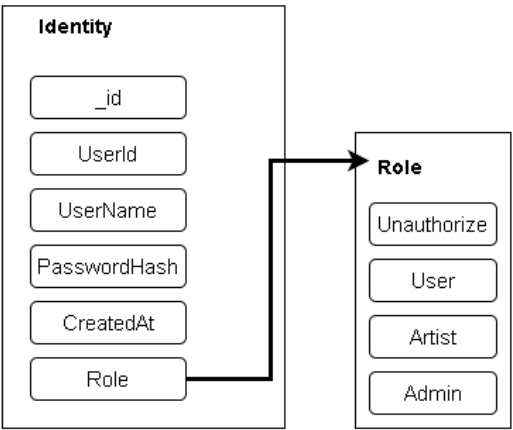


Рисунок 13 – Схема документа для базы данных микросервиса «Авторизация»

Документ имеет поле, которые являются составным объектом – «Role». В таблице 7 приведено описание и тип для значений строк документа.

Таблица 7 – Строки документа базы данных микросервиса «Авторизация»

Значение поля	Описание	Тип
_id	Уникальный идентификатор документа	String
UserId	Уникальный идентификатор пользователя	String
UserName	Уникальное имя пользователя (логин)	String
PasswordHash	Пароль пользователя в виде хэша	String
CreatedAt	Время создания аккаунта	Date
Role	Роль пользователя	String

Данный документ обладает более простой структурой.

Схема документа для базы данных микросервиса «Роли» представлена на рисунке 14.

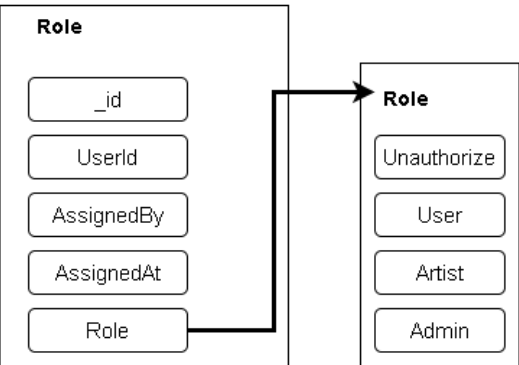


Рисунок 14 – Схема документа для базы данных микросервиса «Роли»

Данный документ обладает простой структурой.

В таблице 8 приведено описание и тип для значений строк документа.

Таблица 8 – Строки документа базы данных микросервиса «Роли»

Значение поля	Описание	Тип
_id	Уникальный идентификатор документа	String
UserId	Уникальный идентификатор пользователя	String
AssignedAt	Время назначения роли	Date
AssignedBy	Кем назначена роль	String
Role	Роль пользователя	String

Стоит отметить, что в логике программы предусмотрено назначение роли системой или администратором.

На рисунке 15 приведена схема документа базы данных микросервиса «Картины».



Рисунок 15 – Схема документа для базы данных микросервиса «Картины»

Данный документ хранит в себе массив значений в поле «AdditionalImageUrls».

В таблице 9 приведено описание и тип для значений строк документа.

Таблица 9 – Строки документа базы данных микросервиса «Картины»

Значение поля	Описание	Тип
_id	Уникальный идентификатор документа	String
ArtistId	Уникальный идентификатор художника	String
ArtistName	Имя художника (логин)	String
Title	Название картины	String
Description	Описание картины	String
Category	Категория	String
Style	Стиль	String
Material	Материал	String
Price	Цена	Double
Quantity	Количество	Integer
Width	Ширина	Double
Height	Высота	Double
MainImageUrl	Строка с ссылкой на основное изображение	String
AdditionalImmgeUrls	Массив ссылок на дополнительные изображения	Array
CreatedAt	Время публикации картины	Date
UpdatedAt	Время обновления записи картины	Date

Для более наглядного представления, был получен JSON документа. На рисунке 16 приведен JSON.

```
{
  "_id": "18da7c02-b01b-49ae-a7fc-7f6dda5977b5",
  "ArtistId": "8512a1b0-9e48-408a-9fd1-461cae0750db",
  "ArtistName": "elena_morozova",
  "Title": "Утренний сон",
  "Description": "Картина выполнена в жанре Анималистика. На картине изображено 3 кошки, отдыхающие на диване. В описании картины использовались материалы: масло, холст.\r\nВы можете написать для заказа картины в другом ра...",
  "Category": "Живопись",
  "Style": "Анималистика",
  "Material": "Масло",
  "Price": {
    "$numberDecimal": "90000"
  },
  "Quantity": 1,
  "Width": 100,
  "Height": 80,
  "MainImageUrl": "/uploads/artworks/de2ab7298b7543db89140a55e906e634.jpg",
  "AdditionalImmgeUrls": [
    "/uploads/artworks/ba1ac552c51c4bf0ae2a8b925832f402.jpg",
    "/uploads/artworks/eb0b50908a034ae49fcc53a70da57ed6.jpg"
  ],
  "CreatedAt": "2026-06-01T17:31:50.409Z",
  "UpdatedAt": "2026-06-01T17:31:50.409Z"
}
```

Рисунок 16 – JSON документа базы данных «ArtPlatform_Art_DB»

На рисунке также видны данные одного из документов.

Схема документа, хранящегося в базе данных микросервиса «Избранное» представлена на рисунке 17.

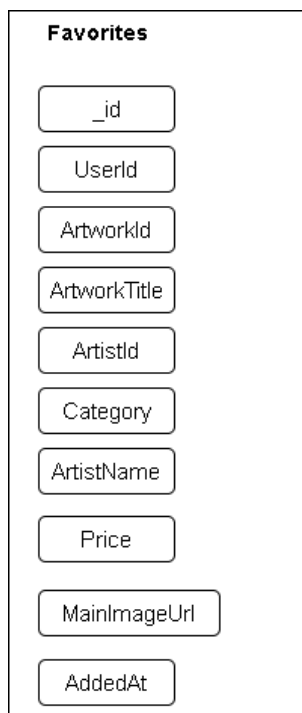


Рисунок 17 – Схема документа для базы данных микросервиса «Избранное»

Данная схема является простой и не содержит в себе вложенных объектов или сложных типов.

В таблице 10 приведено описание и тип для значений строк документа.

Таблица 10 – Строки документа базы данных микросервиса «Избранное»

Значение поля	Описание	Тип
_id	Уникальный идентификатор документа	String
ArtistId	Уникальный идентификатор художника, опубликовавшего картину	String
ArtworkId	Уникальный идентификатор картины, добавленной в избранное	String
UserId	Уникальный идентификатор пользователя, добавившего картину избранное	String
ArtworkTitle	Название картины	String
Category	Категория	String
ArtistName	Имя художника (логин)	String
Price	Цена	Double
MainImageUrl	Строка с ссылкой на основное изображение	String
AddedAt	Время добавления картины	Date

Схема документа, хранящегося в базе данных микросервиса «Корзина» является очень похожей. Она приведена на рисунке 18.

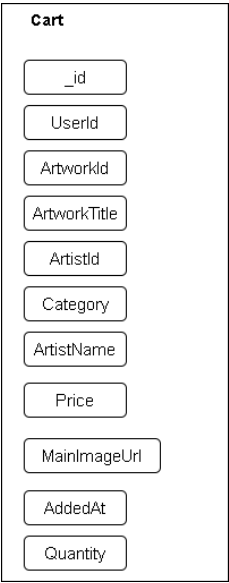


Рисунок 18 – Схема документа для базы данных микросервиса «Корзина»

В таблице 11 приведено описание и тип для значений строк документа.

Таблица 11 – Строки документа базы данных микросервиса «Корзина»

Значение поля	Описание	Тип
_id	Уникальный идентификатор документа	String
ArtistId	Уникальный идентификатор художника, опубликовавшего картину	String
ArtworkId	Уникальный идентификатор картины, добавленной в избранное	String
UserId	Уникальный идентификатор пользователя, добавившего картину избранное	String
ArtworkTitle	Название картины	String
Category	Категория	String
ArtistName	Имя художника (логин)	String
Price	Цена	Double
MainImageUrl	Строка с ссылкой на основное изображение	String
AddedAt	Время добавления картины	Date
Quantity	Количество экземпляров картины	Integer

Таким образом, была реализована распределенная база данных для микросервисов. Реализованная база данных обладает хорошей масштабируемостью и высокой скоростью чтения и выполнения операций над данными.

3.2 Реализация серверной части приложения

Разработка серверной части программных средств для формирования онлайн-пространства и взаимодействия художников с аудиторией проводилась на языке C# в среде разработки Microsoft Visual Studio 2022 с использованием платформы ASP.NET Core, представляющей собой веб-фреймворк, помогающий разработчикам разделить бизнес-логику, маршрутизацию, сервисы и представления, а также предлагает новые методы конфигурирования и работы с расширениями [16]. Фрагмент кода серверной части представлен в приложении Б. Полный код серверной части представлен на сервисе GitHub [17].

В качестве архитектуры для разработки серверной части программных средств была выбрана микросервисная архитектура за ее распределенность, ограниченный контекст и масштабируемость. При этом для каждого микросервиса выбран единый стиль – многослойная архитектура, состоящая из слоя представления, слоя сервисов, слоя доступа к данным, слоя базы данных и слоя обмена асинхронными сообщениями. Достоинства такой архитектуры в легкости поддержки и изменения кода, а также модульности и повторном использовании кода [18].

В результате проектирования и реализации серверной части программных средств было разработано 6 микросервисов, а именно: микросервис «Авторизация», микросервис «Пользователи», микросервис «Роли», микросервис «Картины», микросервис «Избранное» и микросервис «Корзина». Все эти микросервисы работают независимо друг от друга и не знают о существовании друг друга. Благодаря этому, в случае ошибки или критической ситуации в одном микросервисе – остальные продолжают свою работу.

При этом микросервисы обмениваются асинхронными сообщениями для получения актуальной информации для обновления некоторых полей. Для обмена асинхронными сообщениями используется брокер сообщений Apache Kafka.

Помимо этого, запросы пользователей идут к микросервисам через API-шлюз, который направляет запросы к конкретным микросервисам. Для реализации проекта API-шлюза используется ocelot. API-шлюз Ocelot разработан на основе .NET Core специально для архитектуры микрослужб, в которой нужны единые точки входа в систему. Он не требует большого количества ресурсов, быстро работает и легко масштабируется [19].

Разработанные микросервисы предоставляют точки доступа. Точки доступа для микросервиса «Авторизация» приведены в таблице 12.

Таблица 12 – Точки доступа микросервиса «Авторизация»

Метод	Точка доступа	Параметры
POST	/auth/register	нет
POST	/auth/login	нет
POST	/auth/change-password	Authorization: Bearer

Данные три точки доступа представляют функционал для регистрации, авторизации, а также изменения пароля, соответственно. Точки доступа для микросервиса «Пользователь» приведены в таблице 13.

Таблица 13 – Точки доступа для микросервиса «Пользователи»

Метод	Точка доступа	Параметры
GET	/profile	Authorization: Bearer
GET	/profile/{userId}	Path: userId Authorization: Bearer
PUT	/profile/me	Authorization: Bearer
DELETE	/profile/me	Authorization: Bearer
DELETE	/profile/admin/{userId}	Path: userId Authorization: Bearer
GET	/profile/search	Path: ?query=
GET	/profile/check-username/{username}	Path: username
GET	/profile/me	Authorization: Bearer

Микросервис «Пользователи» содержит восемь точек доступа, которые предоставляют функционал для получения своего профиля, его изменения и удаления, а также получения всех профилей пользователей, удаления конкретного пользователя, поиска пользователя и проверки доступности логина. При этом некоторые точки доступа доступны только администратору, а именно метод GET «/profile» и метод DELETE «/profile/admin/{userId}». Для получения

профиля текущего пользователя используется специальный Bearer-токен, который хранит в себе необходимую информацию о пользователе. Точки доступа для микросервиса «Роли» приведены в таблице 14.

Таблица 14 – Точки доступа для микросервиса «Роли»

Метод	Точка доступа	Параметры
GET	/role/{userId}	Path: userId
POST	/role/{userId}/assign	Path: userId Authorization: Bearer
GET	/role/by-role/{role}	Path: userId Authorization: Bearer

Микросервис «Роли» предоставляет три точки доступа – для назначения роли пользователю, получения роли у пользователя и просмотра всех пользователей с указанной ролью. Методы POST «/role/{userId}/assign» и GET «/role/by-role/{role}» доступны только администратору. Точки доступа для микросервиса «Картины» приведены в таблице 15.

Таблица 15 – Точки доступа для микросервиса «Картины»

Метод	Точка доступа	Параметры
GET	/artworks	нет
GET	/artworks/{id}	Path: id
GET	/artworks/artist/{artistId}	Path: artistId
GET	/artworks/{id}/similar	Path: id
GET	/artworks/search	Path: ?query=
GET	/artworks/filter	Path: request
POST	/artworks	Path: request Authorization: Bearer
PUT	/artworks/{id}	Path: id, request Authorization: Bearer
DELETE	/artworks/{id}	Path: id Authorization: Bearer

Микросервис «Картины» предоставляет девять точек доступа, содержащих функционал для получения картины, получения всех картин художника, получения всех картин, получения списка похожих картин, поиска и фильтрации работ, публикации новой картины, изменение записи картины и удаления картины. Метод публикации работы доступен художникам, а методы редактирования и удаления – художникам и администратору. Точки доступа для микросервиса «Избранное» приведены в таблице 16.

Таблица 16 – Точки доступа микросервиса «Избранное»

Метод	Точка доступа	Параметры
GET	/favorites/me	Authorization: Bearer
POST	/favorites/me/{artworkId}	Path: artworkId Authorization: Bearer
DELETE	/favorites/me/{artworkId}	Path: artworkId Authorization: Bearer
GET	/favorites/me/{artworkId}/exists	Path: artworkId Authorization: Bearer

Микросервис «Избранное» имеет четыре точки доступа, предоставляющий инструменты для добавления картины в избранное, удаление картины из избранного, проверки наличия картины в избранном и просмотра всего избранного. Все методы доступны только авторизованным пользователям. Точки доступа для микросервиса «Корзина» приведены в таблице 17.

Таблица 17 – Точки доступа микросервиса «Корзина»

Метод	Точка доступа	Параметры
GET	/cart/me	Authorization: Bearer
POST	/cart/me/{artworkId}	Path: artworkId Authorization: Bearer
PATCH	/cart/me/{artworkId}/decrease	Path: artworkId Authorization: Bearer
DELETE	/cart/me/{artworkId}	Path: artworkId Authorization: Bearer
DELETE	/cart/me	Authorization: Bearer
GET	/cart/me/{artworkId}/exists	Path: artworkId Authorization: Bearer

Микросервис «Корзина» предоставляет шесть точек доступа с функционалом: получение корзины текущего пользователя, добавление картины в корзину, уменьшение количества экземпляров картины в корзине, удаление картины из корзины, очистка корзины, проверка наличия картины в корзине. Все методы доступны только авторизованным пользователям.

Шесть разработанных микросервисов, а также проект «APIGateway», представляющий API-шлюз, при помощи файлов Dockerfile, прописанных для каждого проекта, оборачиваются в образы – docker image. Образ представляет собой набор слоев файловой системы и некоторые метаданные, которые, взятые вместе, могут быть запущены как контейнеры Docker [20].

Один общий файл для всего решения – `docker-compose.yaml` собирает все образы, загружает необходимые программы и, следуя прописанным настройкам, запускает работу серверной части программных средств.

Использование программы Docker для контейнеризации и разворачивания серверной части имеет следующие преимущества:

- быстрота запуска всех микросервисов, благодаря кэшированию образов;
- задание настроек для запуска при помощи `docker-compose.yaml`;
- запуск сервера осуществляется при помощи одной команды.

Таким образом, благодаря выбранной архитектуре и используемым инструментам, серверная часть включает в себя все необходимое для самостоятельной работы, а также для быстрого разворачивания на любой платформе, поддерживающей Docker.

3.3 Реализация клиентской части приложения

Главной задачей при разработке клиентского приложения является создание удобного, интуитивно понятного интерфейса, простого в управлении и не требующего дополнительных навыков или знаний для начала работы. Помимо этого, клиентское приложение должно реализовывать визуально приятный пользовательский интерфейс, отвечающий требованиям предметной области.

Для реализации клиентского приложения использовался язык Typescript. В качестве среды разработки использовалось приложение Microsoft Visual Studio Code. В качестве локального сервера разработки был выбран сервер Vite за его быстроту и простоту использования. Для разработки пользовательского интерфейса использовалась библиотека React, а для стилизации элементов пользовательского интерфейса использовался CSS-framework Tailwind CSS, а также плагин DaisyUI. Помимо этого, в проект клиентского приложения для выполнения HTTP-запросов была включена библиотека Axios. Для валидации данных использовались инструменты библиотеки Zod.

Typescript и React – одна из самых популярных связок для разработки веб-приложения. Она обеспечивает быструю разработку, а также удобство и простоту написания кода. Tailwind CSS и DaisyUI предоставляют готовые стили для компонентов пользовательского интерфейса. Благодаря данным технологиям, разработка клиентского приложения будет быстрой, простой и результативной.

Работа в приложении начинается с главной страницы, поэтому главная страница должна привлекать внимание своим дизайном. Неавторизованный пользователь, входя на сайт, увидит интерфейс главной страницы, верхняя часть которой представлена на рисунке 19.

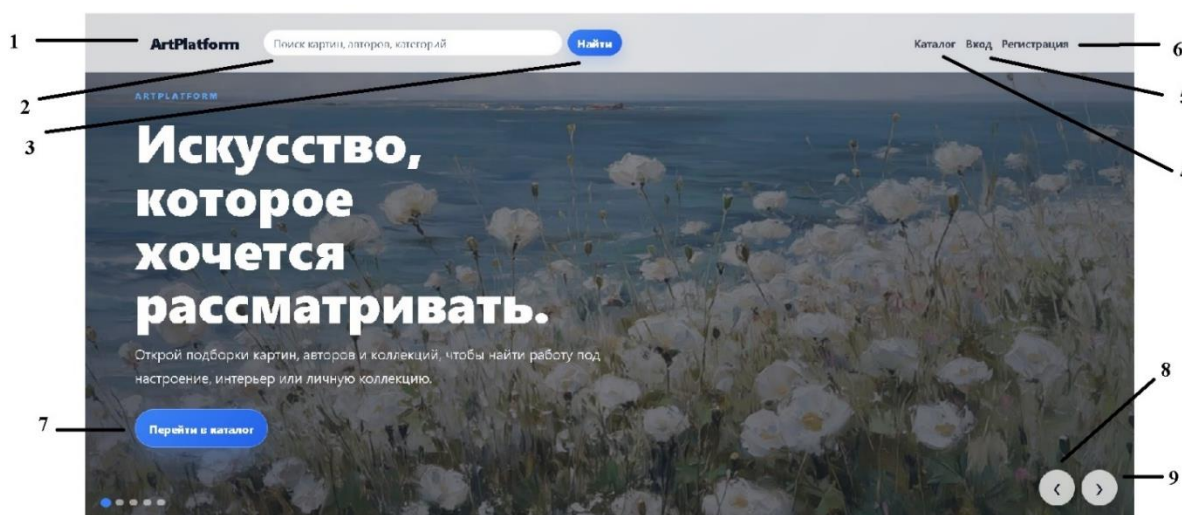


Рисунок 19 – Интерфейс главной страницы

Основными элементами этой страницы являются:

- 1 – кнопка для перехода на главную страницу;
- 2 – поисковая строка для поиска картин по запросу;
- 3 – кнопка «Найти» для поиска картин по запросу в поисковой строке;
- 4 – кнопка для перехода на страницу каталога;
- 5 – кнопка для перехода на страницу авторизации;
- 6 – кнопка для перехода на страницу регистрации;
- 7 – кнопка для перехода на страницу каталога;
- 8 – кнопка для перелистывания изображения назад;

9 – кнопка для перелистывания изображения вперед.

При этом, в случае, если пользователь будет авторизованным, то «шапка» сайта будет иметь другой вид. На рисунке 20 представлено продолжение главной страницы.

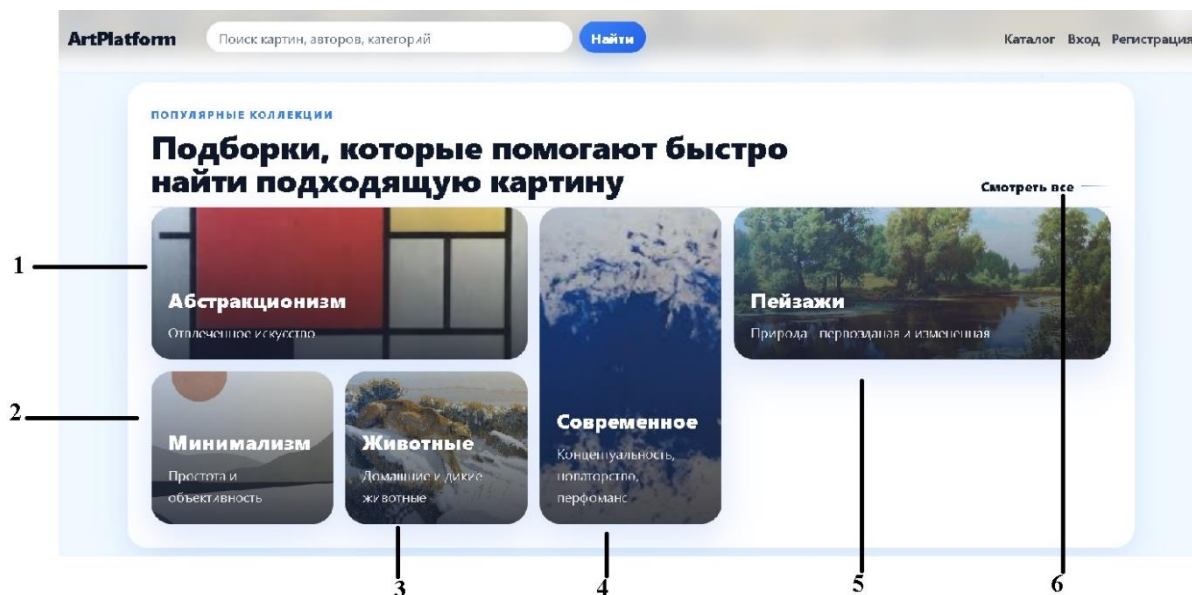


Рисунок 20 – Продолжение главной страницы

Основными элементами являются:

1 – блок с ссылкой на страницу каталога с заданным фильтром «стиль – абстракционизм»;

2 – блок с ссылкой на страницу каталога с заданным фильтром «стиль – минимализм»;

3 – блок с ссылкой на страницу каталога с заданным фильтром «стиль – анималистика»;

4 – блок с ссылкой на страницу каталога с заданным фильтром «стиль – современное искусство»;

5 – блок с ссылкой на страницу каталога с заданным фильтром «стиль – пейзаж»;

6 – кнопка перехода на страницу каталога.

На рисунке 21 приведено окончание главной страницы.

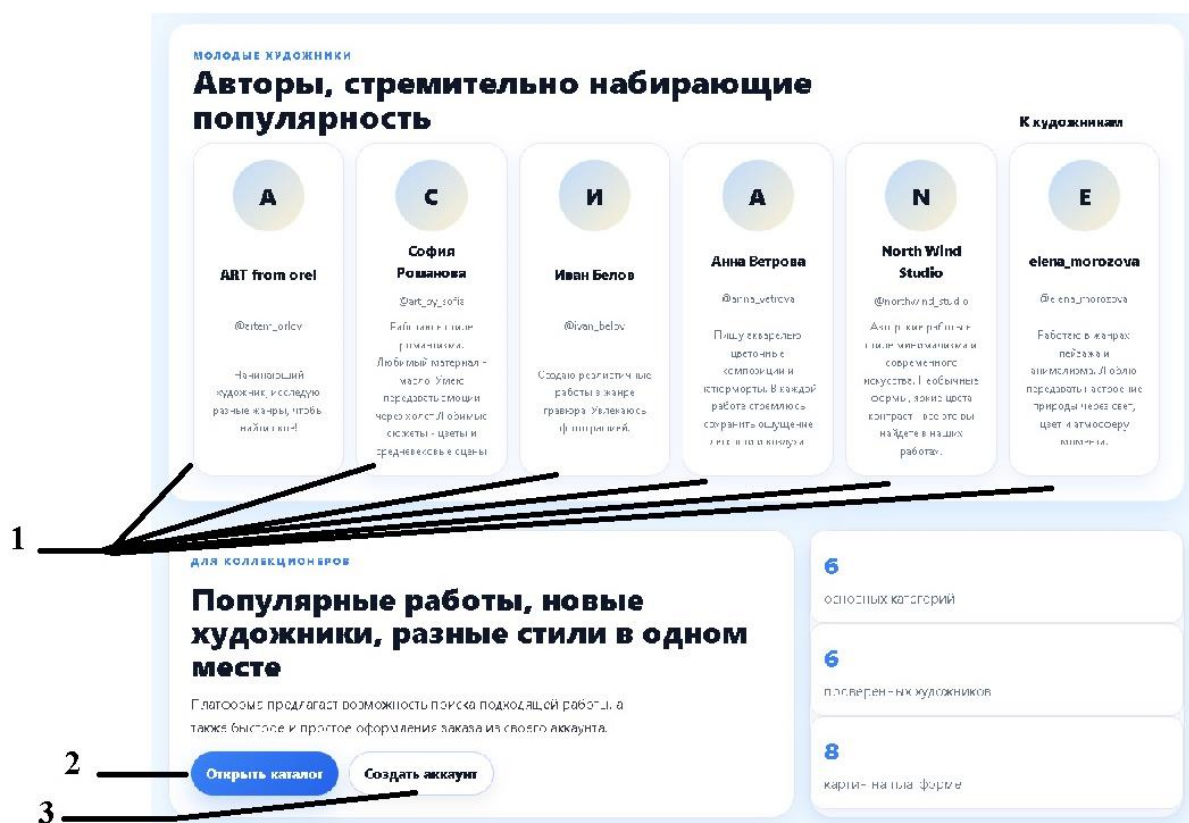


Рисунок 21 – Окончание главной страницы

Основные элементы рассматриваемого экрана:

- 1 – блоки с ссылками на страницы художников;
- 2 – кнопка перехода на страницу каталога;
- 3 – кнопка перехода на страницу регистрации.

Главная страница обладает визуально приятным дизайном и содержит в себе интерактивные элементы для привлечения пользовательского внимания. Также на главной странице реализованы блоки с подборками, картинками и авторами для того, чтобы пользователи смогли быстро изучить цифровую платформу.

Если пользователь захочет перейти на страницу авторизации, он сможет это сделать несколькими способами – перейти с главной странице или по ссылке в «шапке» платформы. Интерфейс страницы авторизации представлен на рисунке 22.

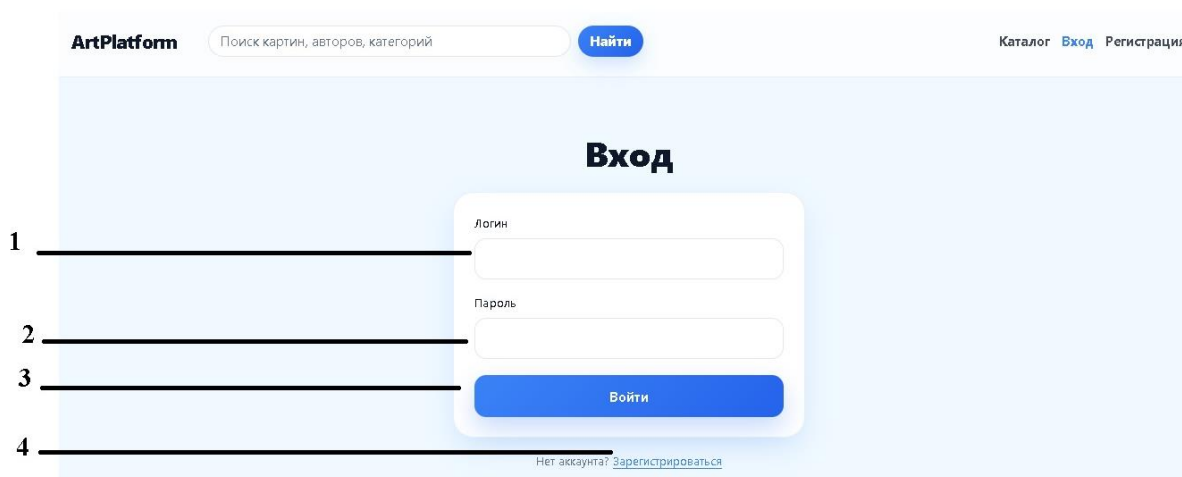


Рисунок 22 – Интерфейс страницы авторизации

Основные компоненты страницы авторизации:

- 1 – поле для ввода логина;
- 2 – поле ввода для пароля;
- 3 – кнопка «Войти», осуществляющая авторизацию;
- 4 – ссылка на страницу регистрации.

При переходе по ссылке на страницу регистрации пользователь попадает на экран, интерфейс которого представлен на рисунке 23.

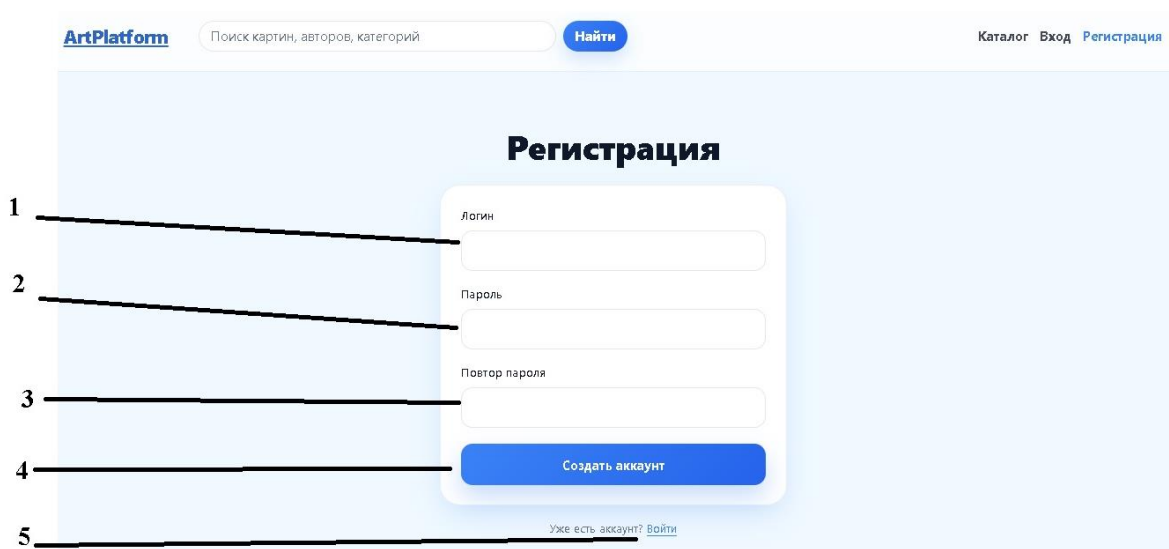


Рисунок 23 – Экран регистрации

Компонентами данного экрана являются:

- 1 – поле для ввода логина;

- 2 – поле для ввода пароля;
- 3 – поле для повторного ввода пароля;
- 4 – кнопка «Создать аккаунт», осуществляющая регистрацию на платформе;
- 5 – ссылка на страницу авторизации.

Также, на «шапке» страницы при помощи подсветки отображается на какой странице сейчас находится пользователь.

Главная страница, страницы входа и регистрации, а также страницы каталога, картины и публичного профиля художника доступны всем пользователям – как авторизованным, так и не зарегистрированным.

Интерфейс страницы каталога представлен на рисунке 24.

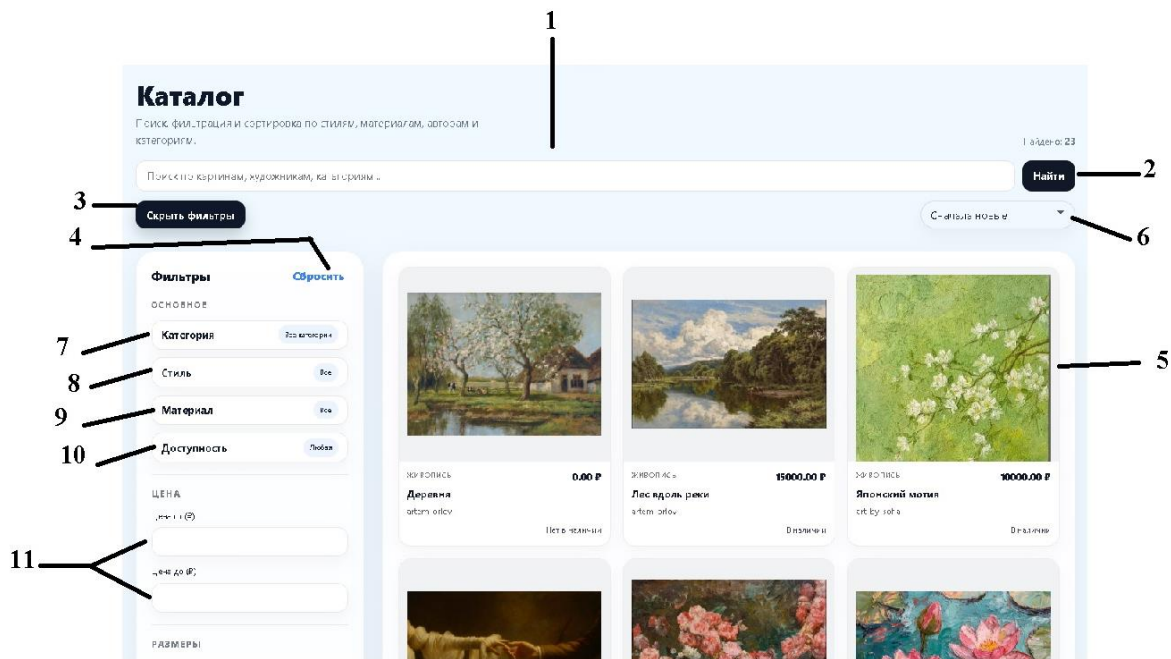


Рисунок 24 – Интерфейс страницы каталога

Основными элементами страницы каталога являются:

- 1 – поисковая строка для ввода запроса;
- 2 – кнопка «Найти», осуществляющая поиск;
- 3 – кнопка «Скрыть фильтры», скрывающая форму фильтров;
- 4 – кнопка «Сбросить», удаляющая текущие значения фильтров и устанавливающая значения по умолчанию;

5 – карточка картины, при нажатии на которую осуществляется переход на страницу картины;

6 – выпадающее меню для выбора режима сортировки;

7 – выпадающее меню для выбора категорий для фильтрации;

8 – выпадающее меню для выбора стилей для фильтрации;

9 – выпадающее меню для выбора материалов для фильтрации;

10 – выпадающее меню для выбора доступности картины для фильтрации;

11 – поля для установки значений для фильтрации.

Если пользователь авторизуется, то для каждой картины появятся две дополнительные кнопки – для добавления картины в избранное и для добавления картины в корзину.

Интерфейс страницы картины представлен на рисунке 25.

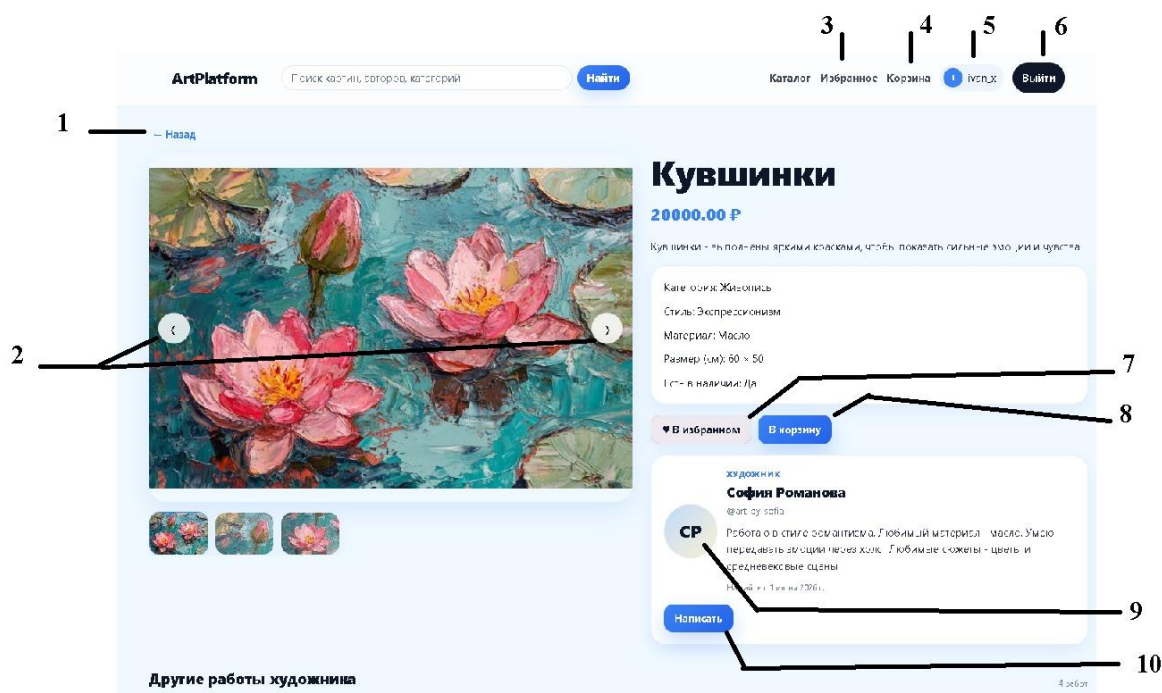


Рисунок 25 – Интерфейс страницы картины

Главными компонентами данной страницы являются:

1 – ссылка для перехода на предыдущую страницу;

2 – стрелки для просмотра всех изображений;

3 – кнопка для перехода на страницу избранного;

4 – кнопка для перехода на страницу корзины;

- 5 – кнопка для перехода на страницу аккаунта;
- 6 – кнопка для выхода из учетной записи;
- 7 – кнопка для добавления картины в избранное;
- 8 – кнопка для добавления картины в корзину;
- 9 – блок с ссылкой на аккаунт художника;
- 10 – кнопка для создания чата с художником.

На данном экране «шапка» сайта выглядит так, как ее видят авторизованные в системе пользователи. Новые компоненты на ней обозначены и подписаны.

Если пользователь нажмет на кнопку «В избранное», то картина добавится в избранные для этого пользователя. Интерфейс страницы избранного представлен на рисунке 26.

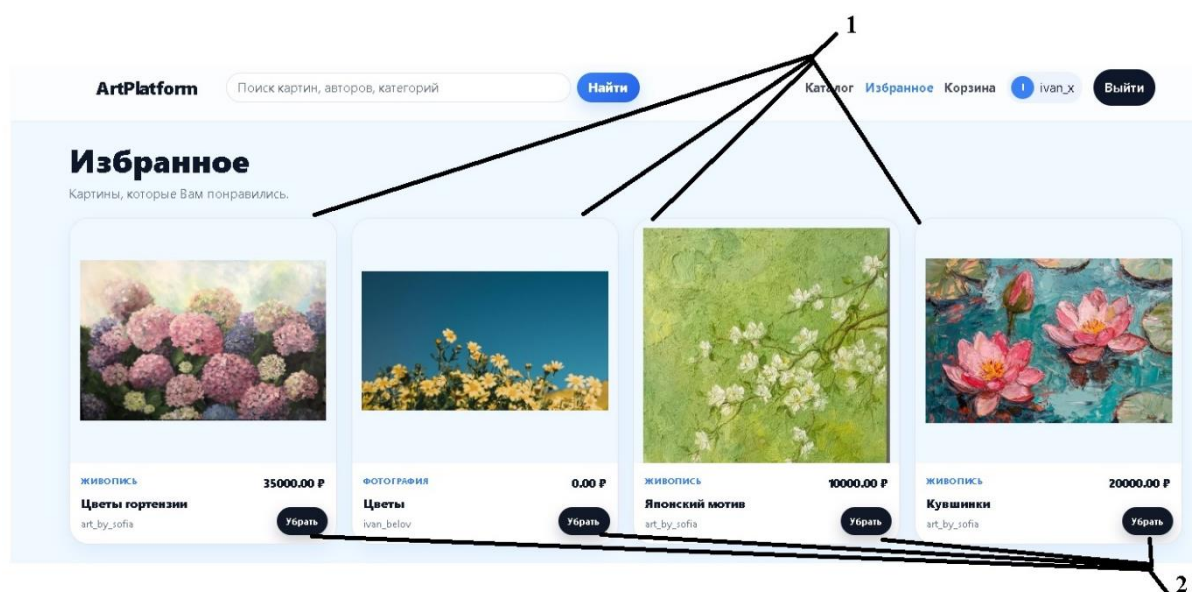


Рисунок 26 – Интерфейс экрана избранного

Основные элементы страницы:

- 1 – карточки картин, при нажатии на которые осуществляется переход на страницу картины;
- 2 – кнопка для удаления картины из избранного.

Если пользователь нажмет на ссылку для перехода в корзину, то он увидит интерфейс, представленный на рисунке 27.

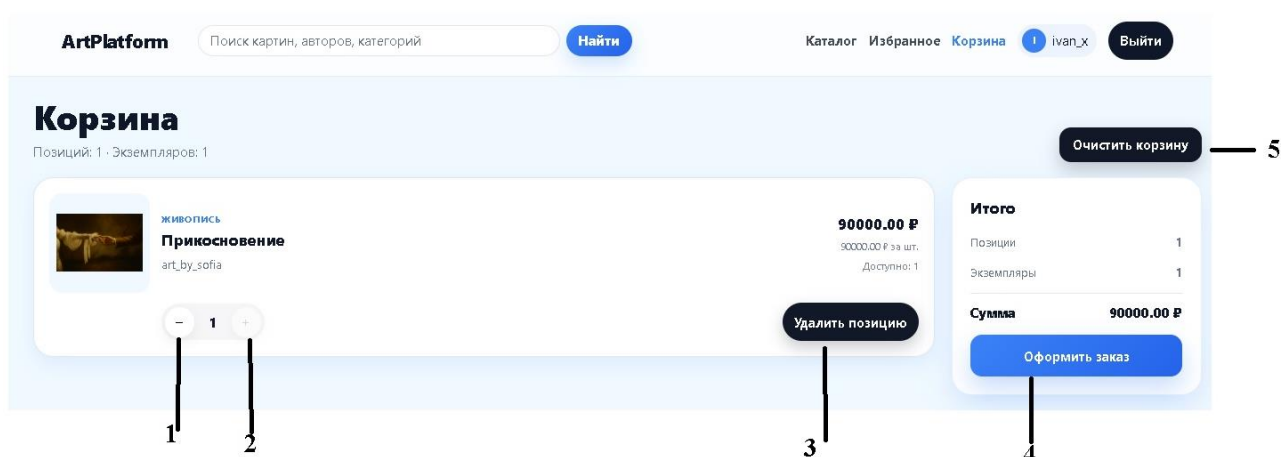


Рисунок 27 – Интерфейс страницы корзины

Основными компонентами экрана являются:

- 1 – кнопка для уменьшения количества экземпляров картины;
- 2 – кнопка для увеличения количества экземпляров картины;
- 3 – кнопка для удаления всех экземпляров выбранной картины из корзины;
- 4 – кнопка для оформления заказа;
- 5 – кнопка для удаления всех картин из корзины.

При этом, на данном экране реализован функционал для уменьшения экземпляров картины или удаления всей позиции. Как позиции, так и экземпляры подсчитываются и отображаются в левом верхнем углу. Пользователь сможет управлять количеством добавленных экземпляров картины в корзину – добавлять или удалять их. При этом, для увеличения числа экземпляров картины в корзине реализована проверка наличия картины. Если картины нет в наличии, то пользователь не сможет оформить заказ. Пользователь также может полностью очистить корзину или выбрать позицию для удаления.

На странице автоматически подсчитывается общая сумма товаров в корзине, а также стоимость каждой позиции, при заказе нескольких экземпляров.

При нажатии на кнопку «Оформить заказ» пользователь попадает на страницу оформления, интерфейс которой представлен на рисунке 28.

Оформление заказа
Заполните контактные данные и выберите способ доставки

ФИО: Иванова Ирина Ивановна

Email: irina@examplc.com

Телефон: 7 (000) 123 45 67

Город: Москва

Адрес доставки: Улица, дом, квартира

Способ доставки: Доставка курьером

Способ оплаты: Картой

Комментарий к заказу: Например: оставить у охраны, связаться заранее...

Нажав на кнопку ниже, вы отправляете заявку на оформление заказа.

[Оформить заказ](#) [Вернуться в корзину](#)

Рисунок 28 – Интерфейс страницы оформления заказа

Главные элементы страницы:

- 1 – поля для заполнения пользователем;
- 2 – поля для выбора одного из доступных вариантов;
- 3 – кнопка для подтверждения оформления заказа;
- 4 – кнопка для возврата на страницу корзины.

Стоит отметить, что на данной странице выполняется строгая валидация пользовательского ввода, чтобы уменьшить вероятность ошибок и неточностей при оформлении заказа. Помимо этого, реализована предварительная проверка заказа – после оформления заказа пользователь получит уведомление об успешной операции и сообщение, содержащее информацию, что менеджер свяжется с покупателем в ближайшее время.

Если пользователь на странице картины перейдет к блоку художника, чья работа представлена на странице, и перейдет на его профиль, при помощи нажатия на аватарку или имя художника, то он увидит страницу художника, интерфейс которой представлен на рисунке 29.

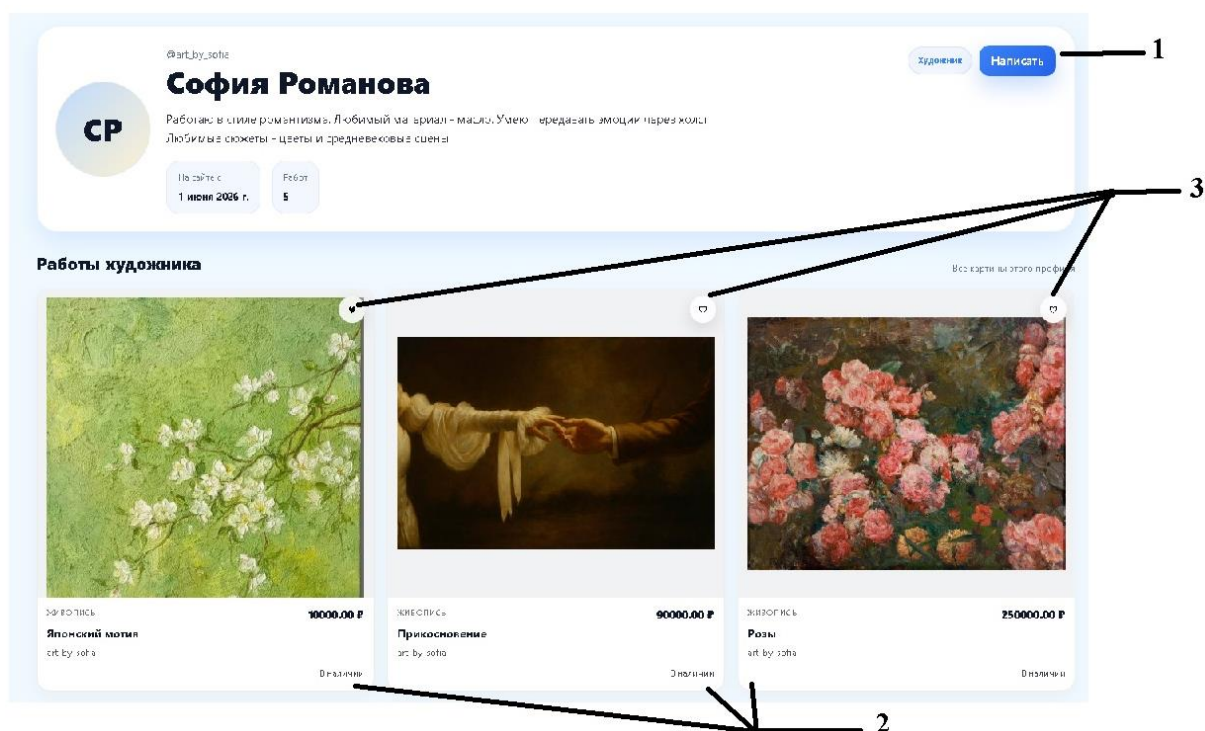


Рисунок 29 – Интерфейс публичной страницы художника

Элементы, используемые на экране:

- 1 – кнопка для создания чата с художником;
- 2 – карточки картин, при нажатии на которые откроется страница картины;
- 3 – кнопка для добавления/удаления в избранное.

На странице художника будут отображены все его работы чтобы пользователи смогли их изучить. Помимо этого, у аккаунта художника есть описание и контактные данные для связи. Также есть кнопка «Написать» для того, чтобы пользователь смог напрямую связаться с художником для уточнения деталей или с вопросом о возможности выполнения картины на заказ.

Аккаунт художника отличается от аккаунта обычного пользователя – на «шапке» появляется дополнительная кнопка, при нажатии на которую откроется страница публикации новой картины, а также у художника есть расширенный функционал, который позволяет ему работать с картинами – публиковать, редактировать или удалять их.

Аккаунт художника представлен на рисунке 30.

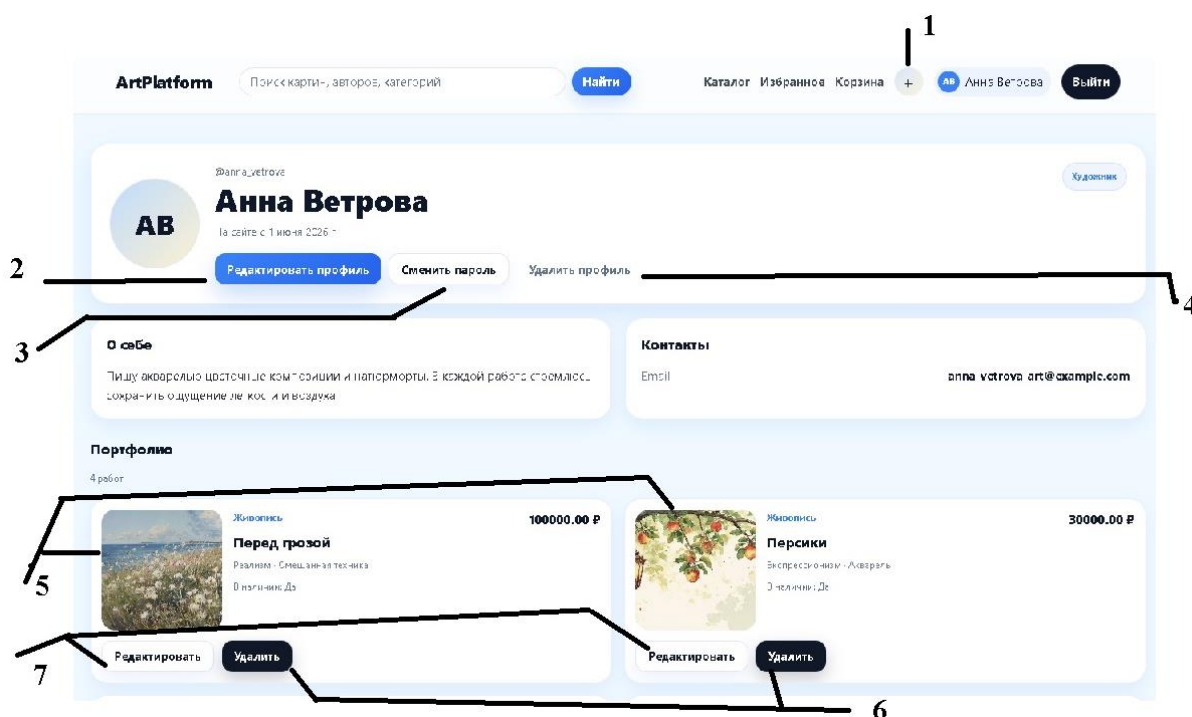


Рисунок 30 – Страница аккаунта пользователя

Основные элементы данного экрана:

- 1 – кнопка для публикации новой картины, доступная только художникам;
- 2 – кнопка для редактирования данных профиля;
- 3 – кнопка для изменения пароля;
- 4 – кнопка для удаления аккаунта;
- 5 – блоки с картинами, при нажатии на которые осуществляется переход на страницу картины;
- 6 – кнопки для удаления картины из портфолио (доступно только художникам);
- 7 – кнопка для редактирования данных картины (доступно только художникам).

На странице профиля пользователя, обладающего ролью художник, представлено портфолио всех его работ, которое также будет видно пользователям платформы.

Функции, доступны только художникам, – это публикация и редактирование картины. Страницы, которые реализуют данный функционал

обладают схожим интерфейсом и функционалом. Интерфейс страницы публикации новой картины представлен на рисунке 31.

The screenshot shows a web form titled 'Публикация картины' (Publication of a picture) under the heading 'НОВАЯ РАБОТА'. The form includes several input fields and buttons. Numbered annotations point to specific elements: 1 points to the 'Имя' (Name) field; 2 points to the 'Категория' (Category), 'Жанр' (Genre), and 'Материал' (Material) dropdown menus; 3 points to the 'Загрузить изображение' (Upload image) and 'Загрузить описание' (Upload description) buttons; 4 points to the 'Опубликовать картину' (Publish picture) button; 5 points to the 'Отмена' (Cancel) button; and 6 points to the 'x' close button in the top right corner. The form also includes fields for 'Стиль' (Style), 'Регион' (Region), 'Семейство' (Family), 'Високос' (Leap year), and 'Описание' (Description).

Рисунок 31 – Интерфейс страницы публикации новой картины

Основные компоненты, использованные на странице:

- 1 – поля для заполнения художником;
- 2 – поля для выбора одного из предложенных вариантов;
- 3 – кнопки для загрузки графических файлов;
- 4 – кнопка для публикации картины на платформе;
- 5 – кнопка для отмены всех действий, выполненных на странице;
- 6 – кнопка для выхода со страницы.

На странице реализована строгая валидация для проверки ввода. Для поля «Название» используется регулярное выражение, для остальных полей, которые заполняются пользователем – ограничение по длине или диапазону. Для полей «Категория», «Стиль» и «Материал» предусмотрен список, из которого художник может выбрать значение.

Для администраторов платформы доступна страница «Админ-панель», содержащая в себе функционал для управления пользователями и картинами на платформе. На рисунке 32 представлен интерфейс этой страницы.

4 Контроль качества программного обеспечения

4.1 Выбор режима тестирования и метрик кода

Тестирование является завершающим этапом разработки ПО. Данный шаг является важнейшим для корректного функционирования разработанного программного обеспечения, так как тестирование выявляет ошибки в работе программы, некорректное поведение программных средств или реакцию системы на непредвиденные ситуации и ее поведение в этом случае. Причиной неудовлетворительной работы программных средств может стать как непредвиденный пользовательский ввод, так и критические ошибки в самой программе. Поэтому очень важно на этапе тестирования выявить уязвимые места в разработанном программном обеспечении и устранить их. Помимо этого, при помощи тестирования можно оценить насколько разработанная программа соответствует выявленным требованиям.

Существует несколько видов тестирования, к которым относятся функциональное, модульное, интеграционное, системное, эксплуатационное приемочное, нагрузочное и другие [21].

Для тестирования клиентской части разработанных программных средств уместно использовать функциональное тестирование, так как оно позволяет убедиться, что все функции выполняются корректно, а система работает так, как этого ожидают пользователи. Функциональное тестирование – один из самых популярных видов тестирования, который тестирует конкретные функции приложения, а также проверяет соответствие функциональности продукта тому, как он был задуман [22].

Для тестирования серверной части программных средств уместно использовать модульное тестирование, так как оно используется для проверки конкретного блока кода. Модульное тестирование – это тестирование, при котором проверяются отдельные небольшие части программного кода, которые можно тестировать вне зависимости от остального кода [23].

Помимо этого, для оценки разработанных программных средств формирования онлайн-пространства искусства и взаимодействия художников с аудиторией будут применяться метрики. Метрики кода – это набор программных показателей, которые дают лучшее представление о коде [24]. В качестве метрик кода будут использоваться такие метрики, как: метрика Чепина, цикломатическая сложность Маккейба и метрика SLOC.

Метрика SLOC (Source Lines Of Code) – это метрика для измерения сложности кода, которая хорошо подходит для небольших проектов. На данную метрику сильно влияет как язык программирования, для которого находится метрика, так и стиль оформления программы. Тем не менее, метрика SLOC – это одна из самых простых и быстрых для вычисления метрик, которая демонстрирует численную оценку сложности кода.

Суть метрики Чепина заключается в оценке информационной прочности программного модуля при помощи анализа характера использования переменных ввода-вывода [25]. То есть все переменные ввода-вывода, используемые в рассматриваемом блоке разделяются на четыре группы. После чего, при помощи формулы, рассчитывается значение метрики. Данная метрика более наглядно демонстрирует сложность модуля в отличие от метрики SLOC.

Объектно-ориентированная метрика WMC позволяет определить сложность класса, при помощи подсчета цикломатической сложности его методов. Цикломатическая сложность Маккейба – это программная метрика, которая для модуля кода подсчитывает количество линейно независимых путей, то есть цикломатическая сложность определяется числом, показывающим сложность модуля кода для понимания. Чем выше это число, тем сложнее код и тем выше вероятность, что в нем есть ошибки.

Подсчет метрик для кода даст понимание о сложности реализации функций, наличия в них дублирования или неиспользуемых переменных, а также о сложности поддержки разработанной платформы.

4.2 Тестирование программных средств формирования онлайн-пространства искусства и взаимодействия художников с аудиторией

4.2.1 Тестирование пользовательской части цифровой платформы

Для проведения тестирования пользовательской части цифровой платформы было выбрано функциональное тестирование, а именно тестирование методом «белый ящик». Метод тестирования «белый ящик», он же – прозрачное тестирование, позволяет исследовать внутреннюю структуру программы, путем проверки каждого пути каждой ветки алгоритма [26]. Основным отличием метода тестирования «белый ящик» от метода «черный ящик» заключается в том, что для прозрачного тестирования необходимо понимание логики работы программы и исходного кода. Для «черного ящика» достаточно иметь входные и выходные данные, по которым будет тестироваться код.

Целью тестирования пользовательской части цифровой платформы является оценка качества исходного кода.

В метод «белый ящик» входят такие составляющие, как: покрытие операторов, покрытие решений, покрытие условий, покрытие решений и условий, комбинаторное покрытие условий.

Для проведения тестирования была спроектирована схема, демонстрирующая работу алгоритма заказа картины пользователем. Для того, чтобы пользователь смог оформить заказ, необходимо, чтобы выполнялись одновременно два условия, а именно: пользователь имеет аккаунт и в данный момент авторизован, картина есть в наличии. Если оба этих условия соблюдаются, то пользователь сможет успешно оформить заказ. Если же какое-то условие не было соблюдено – оформить заказ у пользователя не получится.

Спроектированная схема работы алгоритма заказа картины представлена на рисунке 33.

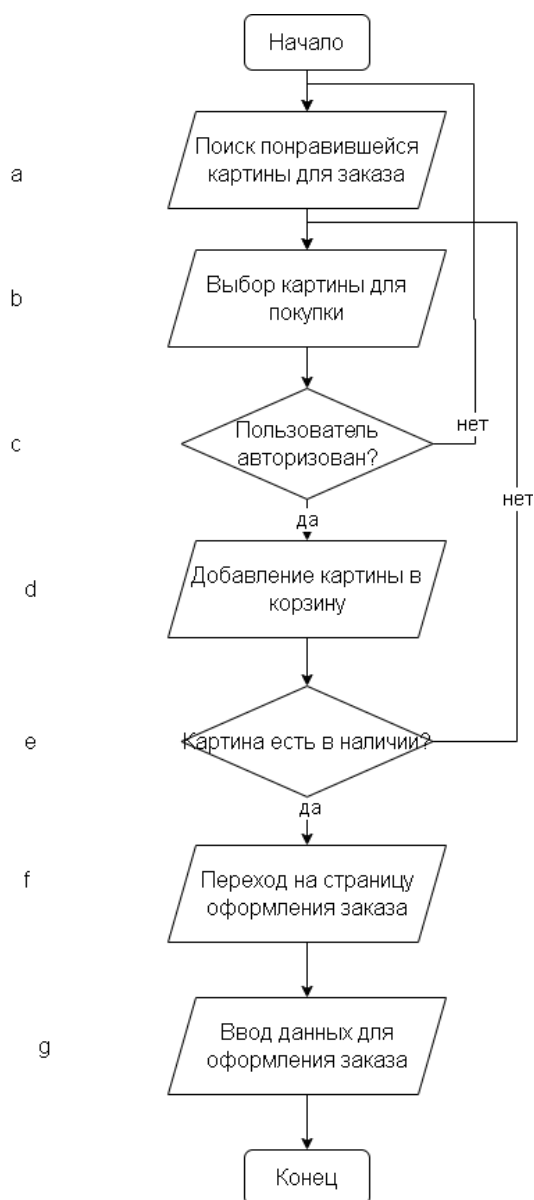


Рисунок 33 – Схема алгоритма заказа картины пользователем

Рассматриваемый алгоритм содержит в себе два условия («Пользователь авторизован?» и «Картина есть в наличии?»), обозначенные, как *c* и *e*, соответственно. В этих местах алгоритм ветвится, то есть для одного входа на блок условия, есть два выхода. Стоит также отметить, что оба эти условия могут быть как истинными, так и ложными.

Остальные блоки схемы обозначены буквами *a*, *b*, *d*, *f*, *g*. Они представляют из себя операторы, в каждом из которых происходит какое-то действие.

Если условие будет истинным, то алгоритм продолжает свою работу и переходит в следующий оператор. Если же условие ложно, то алгоритм начинается с предыдущего места до условия.

В качестве первой составляющей части тестирования методом «белый ящик» было рассмотрено покрытие операторов. Метод покрытия операторов выполняет каждый оператор хотя бы один раз [26]. То есть, каждая инструкция программы должна быть выполнена хотя бы единожды.

В таблице 18 приведен метод покрытия операторов, для которого выделены все операторы, а также подобраны такие входные данные, которые дают возможность пройти по всем операторам и выполнить каждый из них. Чтобы удовлетворить поставленные условия был проведен один тест с такими входными данными, что были пройдены все операторы алгоритма.

Таблица 18 – Покрытие операторов

№	Операторы	Входные данные	Ожидаемый результат	Фактический результат
1	a, b, d, f, g	Название картины = «Цветы гортензии»; Пользователь авторизован = да; В наличии = да.	Пользователь авторизовался, выбрал картину, и она есть в наличии. Пользователь успешно оформил заказ.	Фактический результат соответствует ожидаемому. Результаты тестирования приведены на рисунках Г.1 и Г.2 в приложении

Входные данные подобранные в таблице позволяют пройти по всем операторам алгоритма и в результате получить ответ об успешном оформлении заказа.

Следующим рассматриваемым критерием является покрытие решений. Покрытие решений – это метод, для которого необходимо, чтобы каждая точка принятия решения была выполнена как минимум один раз для каждой из возможных веток. То есть, необходимо составить такое число тестов, чтобы для каждого условия алгоритма приняло истинное и ложное значения. Таких тестов было составлено два – первый, где оба условия истинны, второй – где оба условия ложны. Метод покрытия решений приведен в таблице 19.

Таблица 19 – Покрытие решений

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
1	с – да е – да	Название картины = «Японский мотив»; Пользователь авторизован = да; В наличии = да.	Пользователь авторизовался, выбрал картину, и она есть в наличии. Пользователь успешно оформил заказ.	Фактический результат соответствует ожидаемому. Рисунок с результатом тестирования Г.3 приведен в приложении
2	с – нет е – нет	Название картины = «Деревня»; Пользователь авторизован = нет; В наличии = нет.	Пользователь не авторизовался и выбрал картину, которой нет в наличии. Пользователь не сможет заказать картину.	Фактический результат соответствует ожидаемому. Рисунок с результатом тестирования Г.4 приведен в приложении

Для первого теста были подобраны такие входные данные, которые обращают оба условия в истинные, а для второго теста – в ложные.

Если в первом случае, когда оба условия истинны, пользователь сможет оформить заказ, то во втором случае – он этого сделать не сможет. Так как программой предусмотрен только базовый функционал, включающий в себя просмотр картин и профилей художников, для неавторизованных пользователей. То есть неавторизованный пользователь не сможет добавить картину в корзину.

Следующим рассматриваемым критерием является критерий покрытия условий. Покрытие условий – это метод, для которого необходимо, чтобы каждое подвыражение в составе сложного логического условия принимало, как истинное, так и ложное значения, хотя бы один раз. В связи с этим, необходимо подобрать такие входные данные для тестов, при которых все возможные результаты каждого из условия в решении будут выполнены по крайней мере один раз.

Для покрытия условий хватило трех составленных тестов, с входными данными, удовлетворяющими поставленным требованиям.

В таблице 20 приведено покрытие условий.

Таблица 20 – Покрытие условий

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
1	с – да е – да	Название картины = «Кувшинки»; Пользователь авторизован = да; В наличии = да.	Пользователь авторизовался, выбрал картину, и она есть в наличии. Пользователь успешно оформил заказ.	Фактический результат соответствует ожидаемому. Результаты тестирования приведены в приложении на рисунке Г.5
2	с – да е – нет	Название картины = «Облака»; Пользователь авторизован = да; В наличии = нет.	Пользователь авторизовался, выбрал картину, но ее нет в наличии. Пользователь не сможет оформить заказ.	Фактический результат соответствует ожидаемому. Результаты тестирования приведены в приложении на рисунке Г.6
3	с – нет е – нет	Название картины = «Облака»; Пользователь авторизован = нет; В наличии = нет.	Пользователь не вошел в аккаунт, картины нет в наличии. Заказ не может быть оформлен.	Фактический результат соответствует ожидаемому. Результаты тестирования приведены в приложении на рисунке Г.7

Для метода покрытие условий было подобрано такое число тестов, которого является достаточно, чтобы были получены все результаты каждого из условий. Таких результатов всего три.

В первом случае оба условия истинны, в результате чего, пользователь успешно оформляет заказ для выбранной картины. Во втором случае пользователь вошел в свой аккаунт, но картины нет в наличии, в результате чего, он не сможет добавить картину в корзину, так как кнопка будет заблокированной. Также система уведомит пользователя о том, что картины нет в наличии. В третьем случае, пользователь не сможет добавить картину в корзину, так как системой предусмотрен только базовый функционал для пользователей, не вошедших в свой аккаунт.

Критерий покрытия решений и условий представляет собой необходимость подобрать такое количество тестов, которых достаточно для того, чтобы результаты каждого условия выполнялись хотя бы один раз, результаты каждого решения так же выполнялись хотя бы один раз, и каждый оператор должен быть выполнен хотя бы один раз. Данный критерий можно представить, как граф маршрутов, то есть нахождение всех возможных путей от а к g. В таблице 21 приведено покрытие решений и условий.

Таблица 21 – Покрытие решений и условий

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
1	a, b, c (да), d, e (да), f, g	Название картины = «Прикосновение»; Пользователь авторизован = да; В наличии = да.	Пользователь нашел понравившуюся картину, выбрал ее, авторизовался, добавил картину в корзину, картина есть в наличии, пользователь перешел на страницу оформления заказа, ввел данные для заказа. Пользователь успешно оформил заказ.	Фактический результат соответствует ожидаемому. Результаты тестирования приведены в приложении на рисунке Г.8
2	a, b, c (да), d, e (нет), f, g	Название картины = «Полосы»; Пользователь авторизован = да; В наличии = нет.	Пользователь нашел понравившуюся картину, выбрал ее, авторизовался, добавил картину в корзину, картины нет в наличии, пользователь не смог оформить заказ.	Фактический результат соответствует ожидаемому. Результаты тестирования приведены в приложении на рисунке Г.9
3	a, b, c (нет), d, e (да), f, g	Название картины = «Персики»; Пользователь авторизован = нет; В наличии = да.	Пользователь нашел понравившуюся картину, которая есть в наличии, выбрал ее, но не авторизовался, в результате чего не смог продолжить оформление заказа.	Фактический результат соответствует ожидаемому. Результаты тестирования приведены в приложении на рисунке Г.10

Продолжение таблицы 21

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
4	a, b, c (нет), d, e (нет), f, g	Название картины = «Полосы»; Пользователь авторизован = нет; В наличии = нет.	Пользователь нашел понравившуюся картину, которой нет в наличии, выбрал ее, но не авторизовался, в результате чего не смог продолжить оформление заказа	Фактический результат соответствует ожидаемому. Результаты тестирования приведены в приложении на рисунке Г.11

В результате были рассмотрены четыре теста такие, что результаты каждого условия выполняются хотя бы один раз, результаты каждого решения также выполняются хотя бы один раз и каждый из операторов также выполняется хотя бы один раз.

В первом тесте рассмотрен успешный исход алгоритма заказа картины – пользователь нашел понравившуюся картину и выбрал ее, пользователь вошел в свой аккаунт и смог продолжить оформление заказа. Он добавил картину в корзину. Система проверяет наличие картины, и, так как картина есть в наличии, пользователь оформляет заказ, предварительно заполнив форму для оформления. В результате чего получает уведомление об успешном оформлении заказа.

Во втором тесте пользователь нашел понравившуюся картину, выбрал ее, авторизовался и смог продолжить оформление заказа. Но картины не было в наличии, поэтому пользователь не смог добавить ее в корзину, так как система блокирует кнопку и уведомляет пользователя об отсутствии в данный момент картины.

В третьем тесте пользователь нашел понравившуюся картину, выбрал ее, и, хотя картина была в наличии, не смог оформить заказ, так как не авторизовался.

В четвертом тесте рассматривается ситуация, когда пользователь не вошел в свой аккаунт, и картины не было в наличии.

Комбинаторное покрытие условий – это метод, требующий, чтобы все возможные комбинации результатов условий в каждом решении, а также каждый оператор выполнились хотя бы один раз. Тесты, спроектированные для данного критерия, полностью соответствуют тестам для критерия покрытие решений и условий.

4.2.2 Тестирование серверной части цифровой платформы

Для проверки корректности работы серверной части будет использоваться модульное тестирование. Модульное тестирование, оно же unit-тестирование, позволяет проверить корректность работы отдельного блока программы.

Основными требованиями к написанию unit-тестов являются:

- unit-тест должен проверять корректность работы небольшого фрагмента кода;
- фрагмент кода должен тестироваться быстро;
- тестирование должно проходить изолированно от всего остального кода.

Для проведения модульного тестирования была использована одна из возможностей среды разработки Microsoft Visual Studio. Для создания unit-тестов в текущее решение серверного приложения был добавлен новый проект «Проект тестов MSTest». После чего была настроена зависимость в созданном проекте, то есть добавлена ссылка на проект, который необходимо протестировать.

Для разработанной цифровой платформы важнейшим местом, играющим ключевую роль в корректной работе приложения, является метод, который доступен только администратору, и который работает с ролями пользователей. Исходный код тестируемой функции приведен в приложении Б в листинге Б.1.

Суть данной функции следующая: система проверяет, есть ли у пользователя в настоящий момент роль. Если у проверяемого пользователя есть роль, при этом новая роль соответствует текущей, то система пропускает пользователя, без изменений – программа в этом случае не должна обращаться к базе данных для внесения изменений, то есть перезаписывания информации. В

случае, если у проверяемого пользователя есть роль, но она отличается от новой роли, программа должна обратиться к базе данных для поиска необходимой записи и внесения изменений в нее. После чего программа публикует соответствующее сообщение в брокер сообщений, чтобы остальные микросервисы, что прослушивают текущий топик, могли обновить свои данные. Если же роли у пользователя нет, то система назначает ему роль, сохраняет в базе данных новую запись и публикует сообщение в брокер. Данная функция является ключевой, так как в случае ее неисправности, пользователи не смогут авторизоваться в системе.

Исходный код составленных unit-тестов для тестируемой функции приведен в приложении Б листинге Б.2.

Разработанные unit-тесты проверяют поведение тестируемой функции. Тесты проверяют две основные ветки – создание новой роли, если у пользователя не была установлена роль до вызова метода, и проверка корректности существующей роли, если у пользователя роль была установлена в момент вызова функции.

Результат работы тестов приведен на рисунке 34.

```

PS C:\Users\Marin\source\repos\ArtPlatform-demo\src\ArtPlatform-demo\Tests> dotnet test
Восстановление завершено (1,3 с)
RoleService успешно выполнено (1,3 с) → C:\Users\Marin\source\repos\ArtPlatform-demo\src\ArtPlatform-demo\Role
Tests успешно выполнено (2,3 с) → bin\Debug\net8.0\Tests.dll
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.5.3.1+6b60a9e56a (64-bit .NET 8.0.22)
[xUnit.net 00:00:00.19] Discovering: Tests
[xUnit.net 00:00:00.27] Discovered: Tests
[xUnit.net 00:00:00.28] Starting: Tests
Warning:
The component "Fluent Assertions" is governed by the rules defined in the Xceed License Agreement and
the Xceed Fluent Assertions Community License. You may use Fluent Assertions free of charge for
non-commercial use only. An active subscription is required to use Fluent Assertions for commercial use.
Please contact Xceed Sales mailto:sales@xceed.com to acquire a subscription at a very low cost.
A paid commercial license supports the development and continued increasing support of
Fluent Assertions users under both commercial and community licenses. Help us
keep Fluent Assertions at the forefront of unit testing.
For more information, visit https://xceed.com/products/unit-testing/fluent-assertions/
[xUnit.net 00:00:00.69] Finished: Tests
Tests (тест) успешно выполнено (3,1 с)

Сводка теста: всего: 3; сбой: 0; успешно: 3; пропущено: 0; длительность: 3,1 с
Сборка успешно выполнено через 9,7 с
PS C:\Users\Marin\source\repos\ArtPlatform-demo\src\ArtPlatform-demo\Tests>
  
```

Рисунок 34 – Результат проведения unit-тестирования

Первый тест проверяет корректность поведения функции при создании новой записи роли, при различных обстоятельствах. Проверяются наличие в возвращаемом объекте ожидаемых полей, совершение операции вставки объекта в коллекцию, публикация сообщения в брокер.

Второй тест проверяет поведение метода при попытке изменения текущей роли на точно такую же. Важно, чтобы в таком случае не была осуществлена операция вставки в базу данных точно такой же записи, так как такое поведение может привести к критическим ошибкам при работе серверной части. В данном случае, программа должна после сравнения ролей, вернуть текущий объект и не изменять информацию в базе данных, а также не публиковать сообщения в брокер.

Unit-тестирование было успешно пройдено, что означает корректность поведения серверной части приложения в общем и ключевой функции программы, влияющей на состояние системы, в частности. Поведение системы при различных ситуациях полностью соответствует ожидаемому.

4.3 Оценка качества программного кода

4.3.1 Метрика SLOC

Расчет метрики SLOC и других метрик будет производиться для тестируемой при помощи Unit-тестов функции, которая играет ключевое значение в работе всей системы.

Метрика SLOC (Source Lines of Code) – это метрика, которая подсчитывает количество строк исходного кода. Для расчета метрики SLOC для тестируемой функции, необходимо дать определение физическим и логическим строкам кода.

Физическими строками считаются все непустые строки текстового файла. То есть каждая строка, содержащая в себе какие-либо символы, например, команды, комментарии, относится к физической. Обычно, количество физических строк кода сильно зависит от стиля программирования – как много комментариев, какое оформление кода.

Логические строки кода – это все операторы в программе. Логических строк кода будет меньше чем физических для одного и того же фрагмента кода, так как к логическим строкам относятся только строки с какими-либо командами, то есть строки, которые будут исполняться.

При помощи метрики SLOC можно оценить масштаб разработанного модуля и количество комментариев в нем.

В данной функции:

- физических строк кода: 64;
- логических строк кода: 27;
- комментариев: 15.

SLOC является простейшей метрикой, оценивающей трудозатраты. Проанализировав полученные результаты подсчета метрики SLOC, можно сделать вывод, что код анализируемой функции, хорошо прокомментирован, что делает текст функции более понятным. Помимо этого, функция не является простой, но в то же время, она не перегружена, о чем свидетельствует количество логических строк кода. При этом стиль написания кода является объемным, о чем свидетельствует разница между физическими строками кода и суммой логических строк кода с комментариями. То есть, в рассматриваемой функции большое число служебных строк, которые не исполняются программой, но служат для разграничения блоков и модулей.

4.3.2 Метрика Чепина

Метод Чепина состоит в оценке информационной прочности программного модуля на основе результатов анализа характера использования переменных, входящих в состав списка ввода и вывода.

Такие переменные ввода и вывода образуют четыре группы:

- группа Р – вводимые переменные для расчётов и для обеспечения вывода;
- группа М – модифицируемые переменные, создаваемые внутри программы;

- группа С – переменные, участвующие в управлении работой программного модуля, то есть управляющие переменные;
- группа Т – переменные, которые не используются в ходе программы, но объявляются в ней.

Разобьем переменные анализируемой функции на четыре группы.

К первой группе переменных, относятся следующие семь вводимых переменных:

- `userId` – параметр метода;
- `role` – параметр метода;
- `assignedBy` – параметр метода;
- `existing` – входная переменная из базы данных;
- `now` – входная переменная для некоторых полей;
- `_roles` – интерфейс базы данных, вводимая переменная;
- `_producer` – внешний `producer` для ввода сообщений.

Ко второй группе относятся следующие восемь модифицируемых переменных:

- `update` – создается внутри объекта;
- `filter` – создается внутри как критерий поиска;
- `options` – создается внутри вызываемого метода;
- `updated` – результат вызываемого метода;
- `evt` – создается внутри перед публикацией сообщений;
- `newEntry` – создается внутри метода;
- `ex` – создается внутри блока обработки исключений;
- `_logger` – внешняя служба логирования.

К третьей группе относится одна переменная управления: `existing`, которая влияет на ветвление.

К последней группе неиспользуемых переменных не относится не одной переменной, так как в программе отсутствуют переменные, которые объявляются, но нигде не используются.

Рассчитаем метрику Чепина при помощи приведенной формулы с учетом весовых коэффициентов и подсчитанного количества переменных, для каждой из групп:

$$Q = 1 \times P + 2 \times M + 3 \times C + 0.5 \times T,$$

где Q – значение метрики Чепина;

P – количество вводимых переменных;

M – количество модифицируемых переменных;

C – количество управляющих переменных;

T – количество неиспользуемых переменных.

Подставив рассчитанные значения для каждой из используемых в формуле переменных, получим следующее значение метрики Чепина:

$$Q = 1 \times 7 + 2 \times 8 + 3 \times 1 + 0.5 \times 0 = 26.$$

Проанализировав получившееся значение метрики Чепина, можно сделать вывод, что функция является средней по сложности, так как значение метрики входит в определенный диапазон. Для бизнес-логики данное значение сложности является нормальным. Функция не является простой, так как внутри нее происходит работа с базой данных, брокером сообщений, а также обработка ошибок. Основной вклад в полученный результата внесла группа модифицируемых переменных. Хорошим аспектом служит отсутствие неиспользуемых переменных. Также в функции мало управляющих переменных, что упрощает ее понимание.

4.3.3 Объектно-ориентированная метрика WMC

Объектно-ориентированная метрика WMC (Weighted Methods Per Class) позволяет оценить сложность всего класса с учетом сложности всех методов, входящих в него.

Метрика WMC будет рассчитана для класса RolesService.cs, который является одним из классов микросервиса «Роли». Исходный код рассматриваемого класса RolesService.cs приведен в приложении Б.

Для нахождения метрики необходимо рассчитать цикломатическую сложность Маккейба для каждого метода.

Цикломатическая сложность Маккейба (или просто цикломатическая сложность) – это программная метрика, которая измеряет количество линейно независимых путей через исходный код модуля, то есть цикломатическая сложность показывает, насколько сложным для тестирования и понимания является код. Чтобы рассчитать цикломатическую сложность Маккейба (СС), используется формула:

$$CC=1+(\text{число точек принятия решения}),$$

где число точек принятия решения – это логические операторы, тернарные, а также операторы внутри условных выражений.

WMC, в свою очередь, находится, как сумма всех полученных СС для каждого метода. То есть для подсчета метрики необходимо сложить показатели цикломатической сложности каждого из его методов.

В результате подсчета точек ветвления, по приведенной формуле Маккейба, для каждого метода, была подсчитана метрика WMC:

$$WMC=1+1+3+2+1=8.$$

В результате анализа кода класса, а также нахождения цикломатической сложности для каждого метода и подсчета объектно-ориентированной метрики WMC, можно сделать вывод, что код анализируемого класса не является сложным для понимания и для тестирования, так как в нем не много ветвлений и различных путей. Для серверного класса, WMC, равное восьми является хорошим результатом.

В результате контроля качества разработанной цифровой платформы было выявлено, что реализованное приложение функционирует без ошибок. Поведение системы при различных ситуациях полностью соответствует ожидаемому. Разработанное приложение удовлетворяет требованиям, выявленным на этапе анализа.

Заключение

В рамках выпускной квалификационной работы разработана цифровая платформа для формирования онлайн-пространства и взаимодействия художников с аудиторией.

На стадии анализа предметной области и постановки задачи на разработку сформулирована основная проблема, для решения которой разрабатывалось программное обеспечение. Помимо этого, поставлена цель на разработку и выявлены функциональные и нефункциональные требования к системе, которые легли в основу разработки. Также, проведен сравнительный анализ программного обеспечения аналогичного назначения, в ходе которого выявлены проблемы в имеющихся программных средствах, которые решает разработанная цифровая платформа.

Следующим шагом стало проектирование цифровой платформы, в ходе которого спроектирована архитектура приложения и классы серверного приложения. В качестве архитектуры выбрана микросервисная архитектура за ее производительность и масштабируемость. Для проектирования классов сервера построены диаграммы классов UML. На данном также спроектированы структуры данных и клиентское приложение. Для хранения данных использована распределенная база данных, то есть для каждого микросервиса спроектирована своя документо-ориентированная база данных. При проектировании клиентского приложения выделены все страницы цифровой платформы.

На этапе реализации цифровой платформы разработаны базы данных, для документов каждой из которых приведены схемы и спецификации. Реализована серверная часть, состоящая из шести микросервисов, общающихся между собой при помощи асинхронных сообщений через брокер. Помимо этого, микросервисы контейнеризированны при помощи инструмента Docker. В качестве API-шлюза для микросервисов настроен проект с использованием

шлюза Ocelot. Клиентская часть реализована на языке TypeScript и библиотеки React.

На стадии контроля качества разработанного программного обеспечения проведено модульное тестирование для проверки работы серверной части приложения. Для модульного тестирования использовались Unit-тесты, которые успешно пройдены программой. Для клиентской части приложения проведено функциональное тестирование, с использованием метода «белый ящик». Для программного обеспечения рассчитаны метрики, которые показали, что код приложения структурирован, понятен и не содержит в себе лишних или дублирующихся строк.

Таким образом, поставленная на этапе анализа цель для выпускной квалификационной работы успешно достигнута: реализована единая цифровая платформа для формирования онлайн-пространства искусства и взаимодействия художников с аудиторией, сочетающая в себе функционал, позволяющий художникам публиковать свои работы, продвигать их, набирать постоянную аудиторию и монетизировать свое творчество, а пользователям — искать подходящие картины, приобретать их, формировать персональные рекомендации, а также взаимодействовать с художником напрямую для уточнения информации или заказа картины.

Список использованных источников

1. Полевая, В. М. Популярная художественная энциклопедия / В. М. Полевая, В. Ф. Маркузон, Д. В. Сарабьянов, В. В. Синюков. – М.: Советская энциклопедия, 1986. – 447 с.
2. Bizar // <https://bizar.art/> [Электронный ресурс]. URL: <https://bizar.art/about> (дата обращения: 10.05.2026).
3. Ричардс, М. Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы / М. Ричардс, Н. Форд. – СПб.: Питер, 2023. – 448 с.
4. Васильев, А. UML – диаграмма вариантов использования (use case diagram) // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/47940/> (дата обращения: 12.05.2026).
5. Арлоу, Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу. – СПб.: Символ-Плюс, 2007. – 624 с.
6. Иванов, П.В. Управление проектами: Учебное пособие / П.В. Иванов. – Рн/Д: Феникс, 2018. – 102 с.
7. Кузнецова, А. Создаем диаграмму Ганта // UNISENDER [Электронный ресурс] URL: <https://www.unisender.com/ru/blog/chto-takoe-diagramma-ganta-i-kak-ee-sdelat/> (Дата обращения: 17.05.2026).
8. REST API // Skillfactory Media. [Электронный ресурс]. URL: <https://blog.skillfactory.ru/glossary/rest-api/> (дата обращения: 18.05.2026)
9. UML: обзор основных типов диаграмм, диаграмма компонентов. Часть 2 // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/756552/> (дата обращения: 19.05.2026)
10. Круз, Л. Р. Структуры данных и проектирование программ / Л. Р. Круз. – М.: БИНОМ, 2014. – 765 с.

11. Верников, Г. Основы методологии IDEF1X // Проект «Корпоративный менеджмент» [Электронный ресурс]. URL: <https://www.cfin.ru/vernikov/idef/idef1x.shtml/> (дата обращения 20.05.2026)
12. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон. – М.: ДМК-Пресс, 2022. – 496 с.
13. UML диаграмма классов // Обучающий портал «GeeksforGeeks» [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/> (дата обращения: 21.05.2026).
14. Процесс разработки пользовательского интерфейса: 6 простых шагов для создания отличного пользовательского интерфейса. // Обучающий портал «GeeksforGeeks». [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/ui-design-process-everything-you-need-to-know/> (дата обращения: 22.05.2026).
15. Комаров, В. Путеводитель по базам данных / В. И. Комаров. – М.: ДМК-Пресс, 2024. – 520 с.
16. Чамберс, Д. ASP.NET Core. Разработка приложений / Д. Чамберс, Д. Пэккетт, С. Тиммс. – СПб.: Питер, 2018. – 464 с.
17. Киселева, М. Е. Цифровая платформа формирования онлайн-пространства и взаимодействия художников с аудиторией // GitHub [электронный ресурс]. URL: <https://github.com/Kallistara/ArtPlatform-demo.git> (дата обращения 03.06.2026)
18. Чикилев, Д. Д. Виды архитектуры приложений. // Журнал «Научный лидер». [Электронный ресурс]. URL: <https://scilead.ru/article/6673-vidi-arkhitekturi-prilozhenij> (дата обращения: 25.05.2026)
19. Микрослужбы .NET: архитектура для контейнерных приложений .NET // Обучающий портал «Microsoft Learn». [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/architecture/microservices/#copyright> (дата обращения: 26.05.2026)
20. Милл, И. Docker на практике / И. Милл, Э. Х. Сейерс. – М.: ДМК Пресс, 2020. – 516 с.

21. Общая картина модульного тестирования // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/companies/vk/articles/412695/> (дата обращения: 27.05.2026)
22. Функциональное тестирование ПО: задачи, виды, методы проведения // Обучающий портал «Яндекс Практикум». [Электронный ресурс]. URL: <https://practicum.yandex.ru/blog/funkcionalnoe-testirovanie-po/> (дата обращения: 28.05.2026)
23. Куликов, С. Тестирование программного обеспечения. Базовый курс / С. Куликов. – Минск.: EPAM Systems, 2016. – 300 с.
24. Значения метрик кода // Официальная документация Microsoft [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/visualstudio/code-quality/code-metrics-values?view=vs-2022> (дата обращения: 29.05.2025)
25. Рыжков, Е. Программный код и его метрики // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 30.05.2025)
26. Семахин, А. М. Методы верификации и оценки качества программного обеспечения: учебное пособие / А. М. Семахин. – Курган: Изд-во КГУ, 2018. – 150 с.

Приложение А
(справочное)
Глоссарий

Apache Kafka – распределенный программный брокер сообщений.

API (Application Programming Interface) – программный интерфейс приложения. Набор правил, по которым программы общаются между собой.

API-Gateway (API-шлюз) – сервис, выступающий единой точкой входа в приложение для всех клиентских запросов, в микросервисной архитектуре.

ASP.NET Core – кроссплатформенный фреймворк для создания веб-приложений.

Axios – JavaScript библиотека для выполнения HTTP запросов.

Bearer-токен – уникальный ключ, используемый для аутентификации, авторизации и доступа к данным.

Consumer (потребитель) – сервис, читающий сообщения из брокера.

CSS – язык стилей, при помощи которого оформляются веб-приложения.

DaisyUI – библиотека компонентов для Tailwind CSS.

Docker – платформа для контейнеризации приложений, позволяющая упаковывать программы со всех их зависимостями в изолированный контейнеры.

Docker image (docker образ) – шаблон для создания контейнеров docker.

Dockerfile – файл, содержащий инструкции для создания образа docker.

DTO (Data Transfer Object) – шаблон проектирования, используемый для передачи данных между слоями приложения.

Framework (Фреймворк) – готовая программная платформа, задающая структуру проекта и предоставляющая инструменты, компоненты и правила для разработки.

GitHub – онлайн-платформа для хранения кода и совместной работы над проектами.

HTTP – протокол передачи данных, предназначенный для передачи гипертекстовых документов.

JavaScript – интерпретируемый язык программирования.

JSON (JavaScript Object Notation) – текстовый формат обмена данными.

Microsoft Office Project – программа для управления проектами.

Microsoft Visual Studio – интегрированная среда разработки.

Microsoft Visual Studio Code – легкий текстовый редактор для кроссплатформенной разработки веб-приложений.

MongoDB – документо-ориентированная система управления базами данных.

NoSQL (Not only SQL) – нереляционные базы данных.

Ocelot – API-шлюз, разработанный для платформы .NET.

Producer (продьюсер) – сервис отправляющий сообщения в брокер.

React – JavaScript библиотека для разработки пользовательских интерфейсов.

REST API (Representational State Transfer) – архитектурный стиль взаимодействия между клиентом и сервером через HTTP.

SLOC – метрика, используемая для измерения объема программы.

Tailwind CSS – CSS фреймворк для создания веб-сайтов.

Typescript – язык, являющийся надстройкой над JavaScript.

UML (Unified Modeling Language) – унифицированный язык моделирования.

Unit-тесты – вид тестирования, при котором проверяются отдельные модули.

Vite – локальный сервер разработки.

Web-сайт (веб-сайт) – логически связанные страницы, доступные в интернете.

WMC (Weighted Methods per Class) – это объектно-ориентированная метрика, используемая для оценки сложности класса на основе количества и сложности его методов.

Zod – библиотека для валидации и типизации данных.

Диаграмма Ганта – инструмент визуального представления календарного плана проекта в виде полос на временной шкале.

Приложение Б
(обязательное)
Листинг приложения

Листинг Б.1 – Исходный код файла RolesService.cs

```

using MongoDB.Driver;
using RoleService.data;
using RoleService.Models.Entities;
using RoleService.Models.Kafka;
using RoleService.Services.Kafka;

namespace RoleService.Services
{
    /// <summary>
    /// Сервис управления ролями пользователей
    /// </summary>
    public class RolesService : IRolesService
    {
        private readonly IMongoCollection<UserRoleEntry> _roles;
// коллекция ролей
        private readonly KafkaProducerService _producer; //
        продюсер
        private readonly ILogger<RolesService> _logger; //
        логирование

        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="context">БД</param>
        /// <param name="producer">продюсер</param>
        /// <param name="logger">логирование</param>
        public RolesService(MongoDBContext context,
        KafkaProducerService producer, ILogger<RolesService> logger)
        {
            _roles = context.Roles;
            _producer = producer;
            _logger = logger;
        }

        /// <summary>
        /// Получить роль пользователя по UserId
        /// </summary>
        /// <param name="userId">идентификатор
        пользователя</param>
        /// <returns>роль или null</returns>
        public async Task<UserRoleEntry?> GetByUserIdAsync(string
        userId)
        {
            return await _roles.Find(r => r.UserId ==
        userId).FirstOrDefaultAsync();
        }

        /// <summary>
        /// Обновить роль пользователя
        /// </summary>

```

```

        /// <param name="userId">идентификатор
пользователя</param>
        /// <param name="role">поль</param>
        /// <param name="assignedBy">кем назначена</param>
        /// <returns>объект роли</returns>
        public async Task<UserRoleEntry> UpsertRoleAsync(string
userId, UserRole role, string? assignedBy = null)
        {
            var now = DateTime.UtcNow;

            // Проверяем есть ли уже роль у пользователя
            var existing = await GetByUserIdAsync(userId);

            if (existing != null)
            {
                // Если роль не изменилась – ничего не делаем
                if (existing.Role == role)
                {
                    _logger.LogInformation("UpsertRole: role for
{UserId} is already {Role}, no-op", userId, role);
                    return existing;
                }

                // Обновляем существующую запись
                var update = Builders<UserRoleEntry>.Update
                    .Set(r => r.Role, role)
                    .Set(r => r.AssignedAt, now)
                    .Set(r => r.AssignedBy, assignedBy);

                var filter = Builders<UserRoleEntry>.Filter.Eq(r
=> r.UserId, userId);
                var options = new
FindOneAndUpdateOptions<UserRoleEntry> { ReturnDocument =
ReturnDocument.After };

                // Обновляем роль
                var updated = await
_roles.FindOneAndUpdateAsync(filter, update, options);

                // Публикуем событие о смене роли
                try
                {
                    var evt = new RoleChangedEvent
                    {
                        EventId = Guid.NewGuid().ToString(),
                        UserId = userId,
                        Role = role.ToString(),
                        AssignedBy = assignedBy,
                        AssignedAt = now
                    };
                    await _producer.ProduceAsync("role-changed",
evt);
                }
            }
        }
    }

```

```

        catch (Exception ex)
        {
            _logger.LogError(ex, "Failed to produce role-
changed event after update for UserId {UserId}", userId);
        }

        return updated!;
    }
    else
    {
        // Создаем новую запись
        var newEntry = new UserRoleEntry
        {
            UserId = userId,
            Role = role,
            AssignedBy = assignedBy,
            AssignedAt = now
        };

        // Создаем новую роль
        await _roles.InsertOneAsync(newEntry);

        // Публикуем событие о назначении роли
        try
        {
            var evt = new RoleChangedEvent
            {
                EventId = Guid.NewGuid().ToString(),
                UserId = userId,
                Role = role.ToString(),
                AssignedBy = assignedBy,
                AssignedAt = now
            };
            await _producer.ProduceAsync("role-changed",
evt);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Failed to produce role-
changed event after insert for UserId {UserId}", userId);
        }

        return newEntry;
    }
}

/// <summary>
/// Удалить роль пользователя
/// </summary>
/// <param name="userId"></param>
/// <returns>1 - если удалена, 0 - если не
удалена</returns>
public async Task<bool> DeleteRoleAsync(string userId)

```

```

        {
            var filter = Builders<UserRoleEntry>.Filter.Eq(r =>
r.UserId, userId);
            var result = await _roles.DeleteOneAsync(filter);
            return result.DeletedCount > 0;
        }

        /// <summary>
        /// Получить всех пользователей с указанной ролью
        /// </summary>
        /// <param name="role">роль</param>
        /// <returns>список пользователей с ролью</returns>
        public async Task<IEnumerable<UserRoleEntry>>
GetByRoleAsync(UserRole role)
        {
            return await _roles.Find(r => r.Role ==
role).ToListAsync();
        }
    }
}

```

Листинг Б.2 – Исходный код тестов приложения

```

using System;
using System.Threading;
using System.Threading.Tasks;
using FluentAssertions;
using Microsoft.Extensions.Logging;
using Moq;
using MongoDB.Driver;
using RoleService.data;
using RoleService.Models.Entities;
using RoleService.Models.Kafka;
using RoleService.Services;
using RoleService.Services.Kafka;
using Xunit;

namespace RoleService.Tests
{
    /// <summary>
    /// unit тесты для микросервиса RoleService
    /// </summary>
    public class RolesServiceTests
    {
        [Fact]
        public async Task
UpsertRoleAsync_WhenRoleDoesNotExist_InsertsAndProducesEvent()
        {
            var userId = "user-123";
            var role = UserRole.ContentCreator;
            var assignedBy = "unit-test";
            var mockCollection = new
Mock<IMongoCollection<UserRoleEntry>>();

```

```

        mockCollection
            .Setup(c =>
c.InsertOneAsync(It.IsAny<UserRoleEntry>(),
It.IsAny<InsertOneOptions>(), It.IsAny<CancellationToken>()))
            .Returns(Task.CompletedTask)
            .Verifiable();
        var mockContext = new Mock<IMongoDBContext>();
        mockContext.SetupGet(c =>
c.Roles).Returns(mockCollection.Object);

        var mockProducer = new Mock<IKafkaProducerService>();
        mockProducer
            .Setup(p => p.ProduceAsync("role-changed",
It.IsAny<RoleChangedEvent>(), It.IsAny<CancellationToken>()))
            .Returns(Task.CompletedTask)
            .Verifiable();

        var mockLogger = new Mock<ILogger<RolesService>>();

        (CallBase = true).
            var mockService = new
Mock<RolesService>(mockContext.Object, mockProducer.Object,
mockLogger.Object) {
                CallBase = true
            };
        mockService
            .Setup(s =>
s.GetByUserIdAsync(It.IsAny<string>()))
            .ReturnsAsync((UserRoleEntry?)null);

        var result = await
mockService.Object.UpsertRoleAsync(userId, role, assignedBy);

        result.Should().NotNull();
        result.UserId.Should().Be(userId);
        result.Role.Should().Be(role);

        mockCollection.Verify(c => c.InsertOneAsync(
            It.Is<UserRoleEntry>(e => e.UserId == userId &&
e.Role == role && e.AssignedBy == assignedBy),
            It.IsAny<InsertOneOptions>(),
            It.IsAny<CancellationToken>()),
            Times.Once);

        mockProducer.Verify(p => p.ProduceAsync("role-
changed",
            It.Is<RoleChangedEvent>(ev => ev.UserId == userId
&& ev.Role == role.ToString() && ev.AssignedBy == assignedBy),
            It.IsAny<CancellationToken>()),
            Times.Once);
    }

    [Fact]

```

```

        public async Task
UpsertRoleAsync_WhenRoleIsSame_ReturnsExistingAndDoesNotProduce()
    {
        var userId = "user-456";
        var role = UserRole.User;
        var assignedBy = "unit-test-2";

        var existing = new UserRoleEntry
        {
            UserId = userId,
            Role = role,
            AssignedBy = "someone",
            AssignedAt = DateTime.UtcNow.AddMinutes(-15)
        };

        var mockCollection = new
Mock<IMongoCollection<UserRoleEntry>>();
        mockCollection
            .Setup(c =>
c.InsertOneAsync(It.IsAny<UserRoleEntry>(),
It.IsAny<InsertOneOptions>(), It.IsAny<CancellationToken>()))
            .Returns(Task.CompletedTask);

        var mockContext = new Mock<IMongoDBContext>();
        mockContext.SetupGet(c =>
c.Roles).Returns(mockCollection.Object);

        var mockProducer = new Mock<IKafkaProducerService>();
        var mockLogger = new Mock<ILogger<RolesService>>();

        var mockService = new
Mock<RolesService>(mockContext.Object, mockProducer.Object,
mockLogger.Object) { CallBase = true };
        mockService
            .Setup(s =>
s.GetByUserIdAsync(It.IsAny<string>()))
            .ReturnsAsync(existing);

        var result = await
mockService.Object.UpsertRoleAsync(userId, role, assignedBy);

        result.Should().BeSameAs(existing);
        mockProducer.Verify(p =>
p.ProduceAsync(It.IsAny<string>(), It.IsAny<RoleChangedEvent>(),
It.IsAny<CancellationToken>()), Times.Never);

        mockCollection.Verify(c =>
c.InsertOneAsync(It.IsAny<UserRoleEntry>(),
                It.IsAny<InsertOneOptions>(),
                It.IsAny<CancellationToken>()),
                Times.Never);
    }

```

```

        mockCollection.Verify(c =>
c.FindOneAndUpdateAsync(It.IsAny<FilterDefinition<UserRoleEntry>>()
),
        It.IsAny<UpdateDefinition<UserRoleEntry>>(),
        It.IsAny<FindOneAndUpdateOptions<UserRoleEntry,
UserRoleEntry>>() ,
        It.IsAny<CancellationToken>()),
        Times.Never);
    }
}
}

```

Листинг Б.3 – Исходный код файла RoleController.cs

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using RoleService.Models.DTO;
using RoleService.Models.Entities;
using RoleService.Models.Kafka;
using RoleService.Services;
using RoleService.Services.Kafka;

namespace RoleService.Controllers
{
    /// <summary>
    /// Контроллер для управления ролями пользователей.
    /// Базовый маршрут api/role/
    /// </summary>
    [Route("api/[controller]")]
    [ApiController]
    public class RoleController : ControllerBase
    {
        private readonly IRolesService _roleService; // Сервис
управления ролями

        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="roleService">бизнес-сервис управления
ролями</param>
        public RoleController(IRolesService roleService)
        {
            _roleService = roleService;
        }

        /// <summary>
        /// Получить роль пользователя по его идентификатору.
        /// GET /api/role/{userId}
        /// Доступен всем.
        /// </summary>
        [HttpGet("{userId}")]
        public async Task<ActionResult> GetRole(string userId)

```

```

    {
        var r = await _roleService.GetByUserIdAsync(userId);

        if (r == null) return NotFound();
        return Ok(r);
    }

    /// <summary>
    /// Назначить роль пользователю.
    /// POST /api/role/{userId}/assign
    /// Доступен только админу.
    /// </summary>
    [HttpPost("{userId}/assign")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> AssignRole(string userId,
[FromBody] AssignRoleRequest req)
    {
        // Валидация роли
        if (!Enum.TryParse<UserRole>(req.Role, true, out var
parsed))
            return BadRequest(new { error = "Invalid role" });

        var updated = await
_roleService.UpsertRoleAsync(userId, parsed, req.AssignedBy);

        return Ok(updated);
    }

    /// <summary>
    /// Получить всех пользователей с указанной ролью.
    /// GET /api/role/by-role/{role}
    /// Доступен только админу
    /// </summary>
    [HttpGet("by-role/{role}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> GetByRole(string role)
    {
        // Валидация роли
        if (!Enum.TryParse<UserRole>(role, true, out var
parsed))
            return BadRequest(new { error = "Invalid role" });

        var list = await _roleService.GetByRoleAsync(parsed);
        return Ok(list);
    }
}

```

Листинг Б.4 – Исходный код файла UserRoleEntry.cs

```

using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;
using System.Text.Json.Serialization;

```

```

namespace RoleService.Models.Entities
{
    /// <summary>
    /// Класс - роли пользователей в системе
    /// </summary>
    public class UserRoleEntry
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string Id { get; set; } =
        ObjectId.GenerateNewId().ToString();

        // Внешний UserId
        public string UserId { get; set; } = string.Empty;

        // Текущая роль
        [BsonRepresentation(BsonType.String)]
        [JsonConverter(typeof(JsonStringEnumConverter))]
        public UserRole Role { get; set; } =
        UserRole.Unauthorized;

        // Кем назначено (UserId администратора)
        public string? AssignedBy { get; set; }

        // Время обновления роли
        public DateTime AssignedAt { get; set; } =
        DateTime.UtcNow;
    }
}

```

Листинг Б.5 – Исходный код файла AuthService.cs

```

using IdentityService.data;
using IdentityService.Models.DTO;
using IdentityService.Models.Entities;
using IdentityService.Services.Kafka;
using Microsoft.AspNetCore.Identity.Data;
using MongoDB.Driver;

namespace IdentityService.Services
{
    /// <summary>
    /// Сервис аутентификации.
    /// </summary>
    public class AuthService:IAuthService
    {
        private readonly IMongoCollection<User> _users; //
        коллекция пользователей
        private readonly ITokenService _tokenService; // бизнес-
        сервис работы с токенами
        private readonly KafkaProducerService _kafkaProducer; //
        продьюсер
    }
}

```

```

    /// <summary>
    /// Конструктор с внедрением зависимостей
    /// </summary>
    /// <param name="context">Контекст MongoDB</param>
    /// <param name="tokenService">Сервис работы с JWT
токенами</param>
    /// <param name="kafkaProducer"></param>
    /// <param name="logger"></param>
    public AuthService(MongoDBContext context, ITokenService
tokenService,
        KafkaProducerService kafkaProducer)
    {
        _users = context.Users;
        _tokenService = tokenService;
        _kafkaProducer = kafkaProducer;
    }

    /// <summary>
    /// Регистрация нового пользователя
    /// </summary>
    /// <param name="request">данные для регистрации</param>
    /// <returns>id созданного пользователя</returns>
    public async Task<string>
RegisterAsync (Models.DTO.RegisterRequest request)
    {
        var existing = await _users.Find(u => u.Username ==
request.Username).AnyAsync();

        if (existing) throw new Exception("Username уже
используется");

        // Генерация UserId
        var userId = Guid.NewGuid().ToString();

        // Хэширование пароля
        var passwordHash =
BCrypt.Net.BCrypt.HashPassword(request.Password);

        // Сохранение в MongoDB
        var user = new User
        {
            UserId = userId,
            Username = request.Username,
            PasswordHash = passwordHash,
            CreatedAt = DateTime.UtcNow
        };
        await _users.InsertOneAsync(user);

        // Создаем объект сообщения для кафки и отправляем его
        var evt = new Models.Kafka.UserRegisteredEvent
        {
            UserId = userId,

```

```

        Username = request.Username,
        CreatedAt = user.CreatedAt
    };
    await _kafkaProducer.ProduceAsync("user-registered",
    evt);

    return userId;
}

/// <summary>
/// Аутентификация пользователя
/// </summary>
/// <param name="request">данные для аутенфикации</param>
/// <returns>объект AuthResponse с id и токеном</returns>
public async Task<AuthResponse?>
LoginAsync(models.DTO.LoginRequest request)
{
    // Поиск пользователя
    var user = await _users.Find(u => u.Username ==
request.Username).FirstOrDefaultAsync();
    if (user == null)
        return null;

    // Проверка пароля
    if (!BCrypt.Net.BCrypt.Verify(request.Password,
user.PasswordHash))
        return null;

    // Генерация токена
    var token = _tokenService.GenerateToken(user.UserId,
user.Username, user.Role);

    return new AuthResponse
    {
        UserId = user.UserId,
        AccessToken = token
    };
}

/// <summary>
/// Изменение пароля
/// </summary>
/// <param name="userId">идентификатор
пользователя</param>
/// <param name="currentPassword">текущий пароль</param>
/// <param name="newPassword">новый пароль</param>
/// <returns>1 - при изменении пароля, 0 - при ошибке
изменения</returns>
public async Task<bool> ChangePasswordAsync(string userId,
string currentPassword, string newPassword)
{
    // Находим пользователя в коллекции

```

```

        var user = await _users.Find(u => u.UserId ==
userId).FirstOrDefaultAsync();
        if (user == null) throw new Exception("User not
found");

        // Проверяем текущий пароль
        if (!BCrypt.Net.BCrypt.Verify(currentPassword,
user.PasswordHash))
            throw new Exception("Current password is
incorrect");

        var newHash =
BCrypt.Net.BCrypt.HashPassword(newPassword);
        var filter = Builders<User>.Filter.Eq(u => u.UserId,
userId);
        var update = Builders<User>.Update.Set(u =>
u.PasswordHash, newHash);

        // Изменяем пароль
        var result = await _users.UpdateOneAsync(filter,
update);
        return result.ModifiedCount > 0;
    }
}

```

Приложение В
(рекомендуемое)
Графическая часть

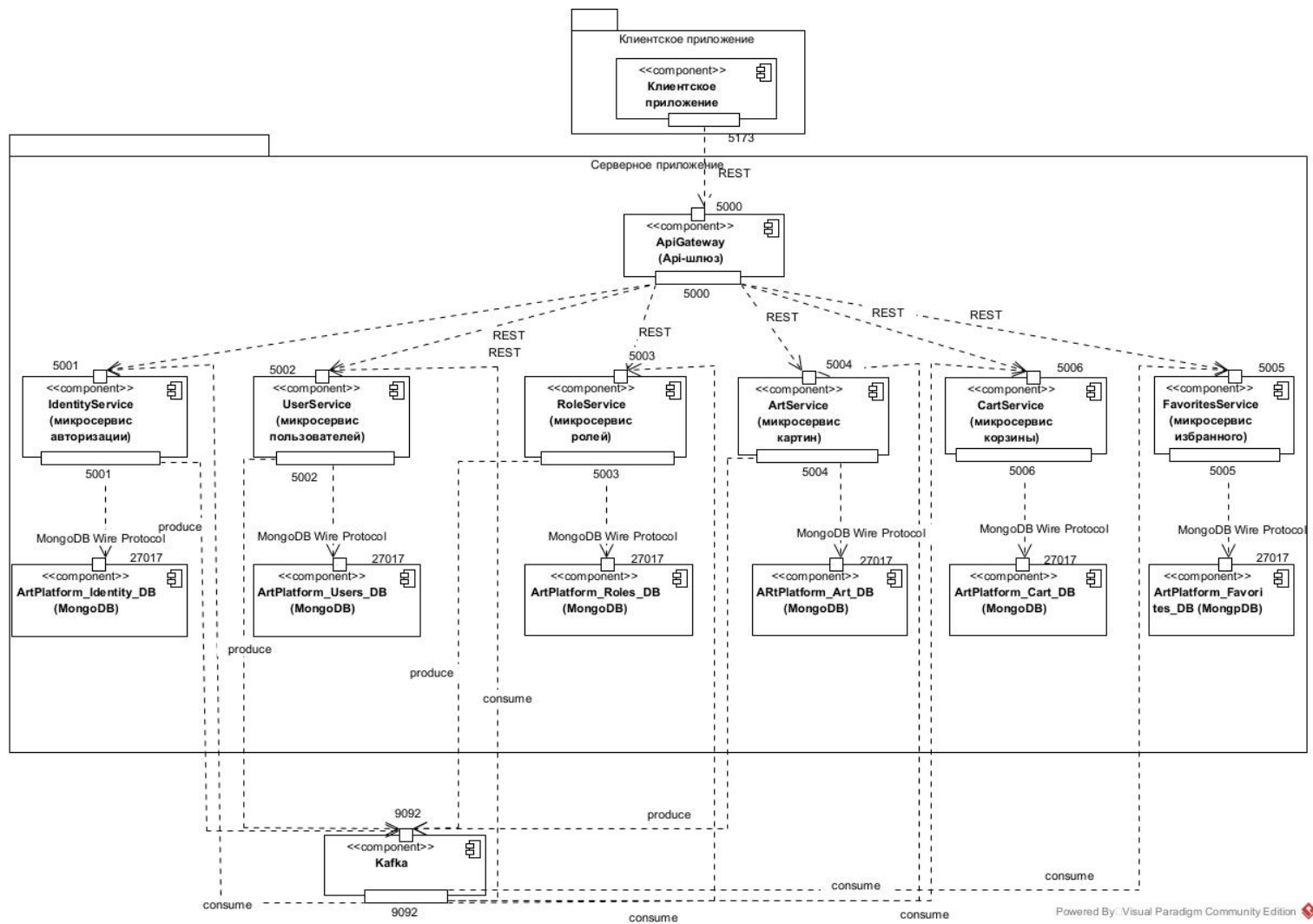


Рисунок В.1 – Диаграмма компонентов цифровой платформы

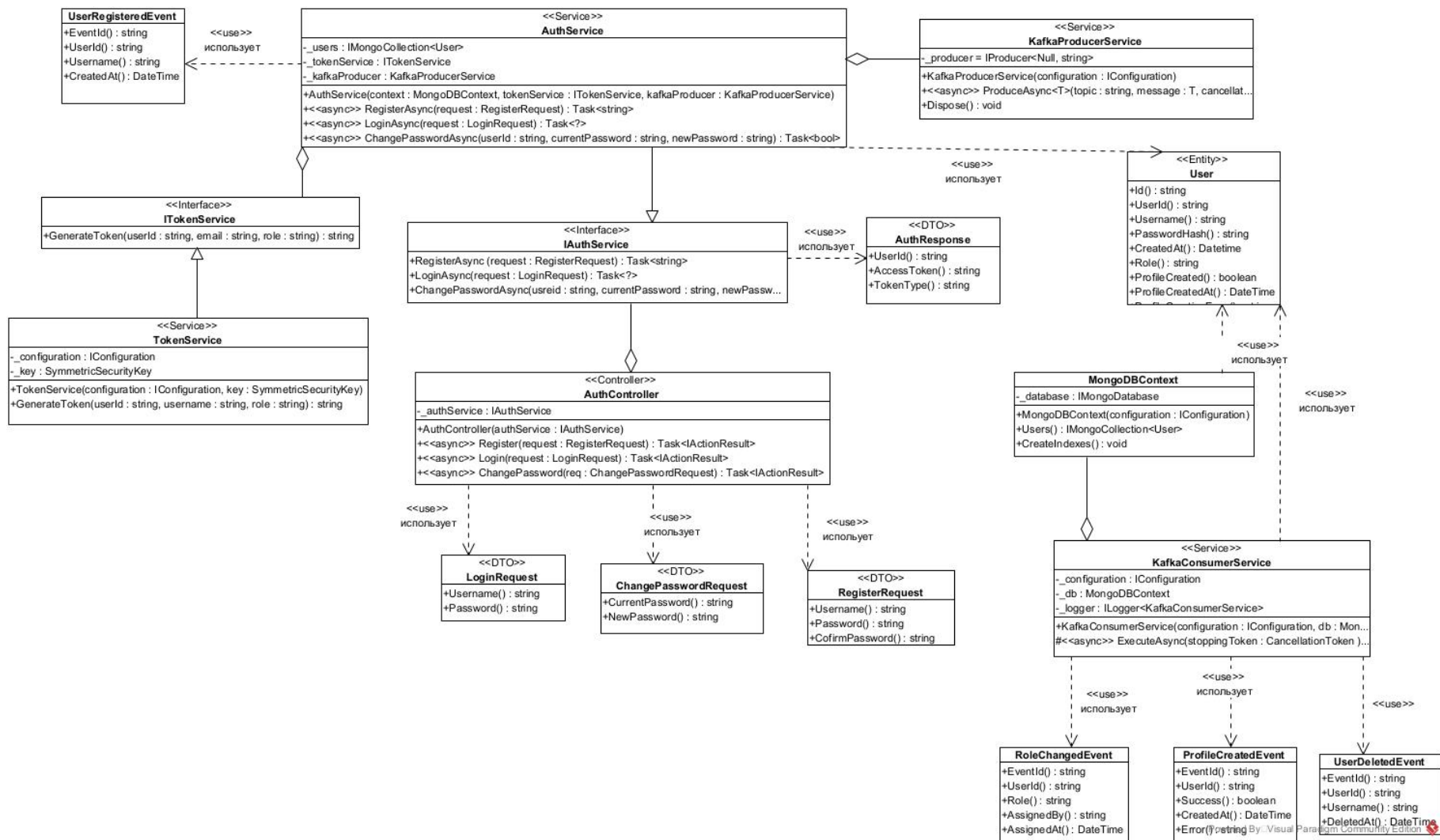


Рисунок В.2 – Диаграмма классов для микросервиса «Авторизация»

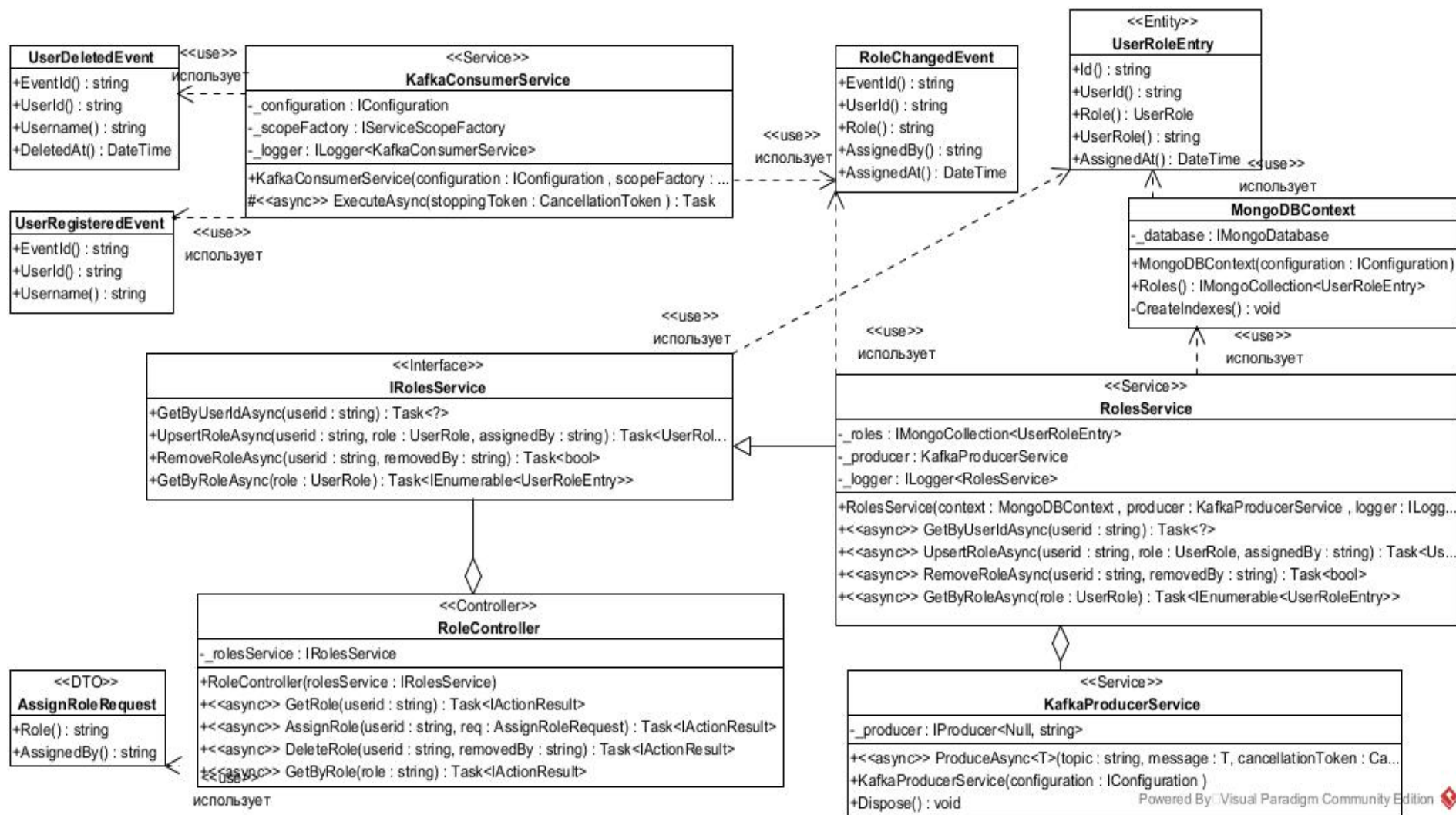


Рисунок В.4 – Диаграмма классов для микросервиса «Роли»

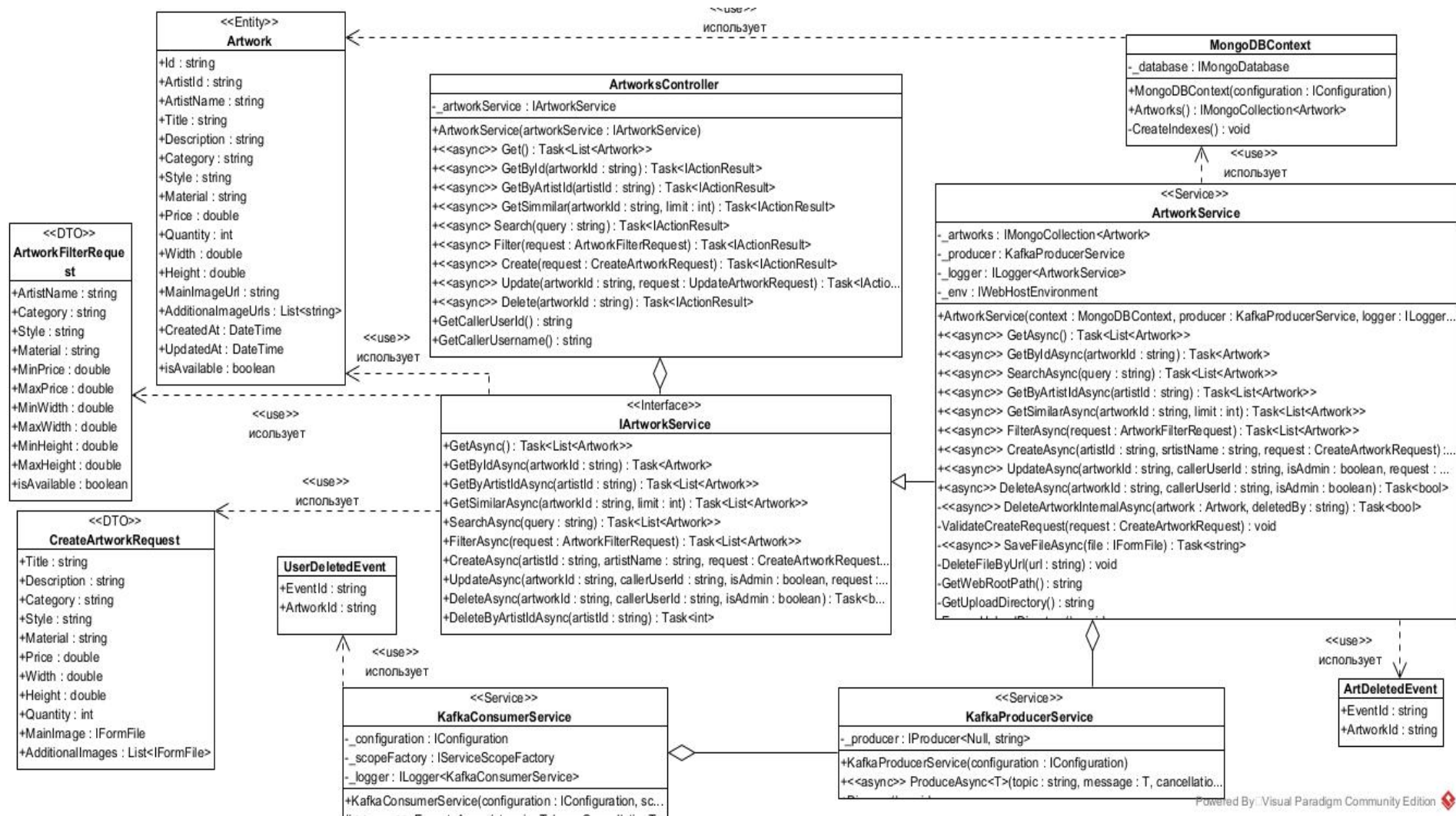


Рисунок В.5 – Диаграмма классов для микросервиса «Картины»

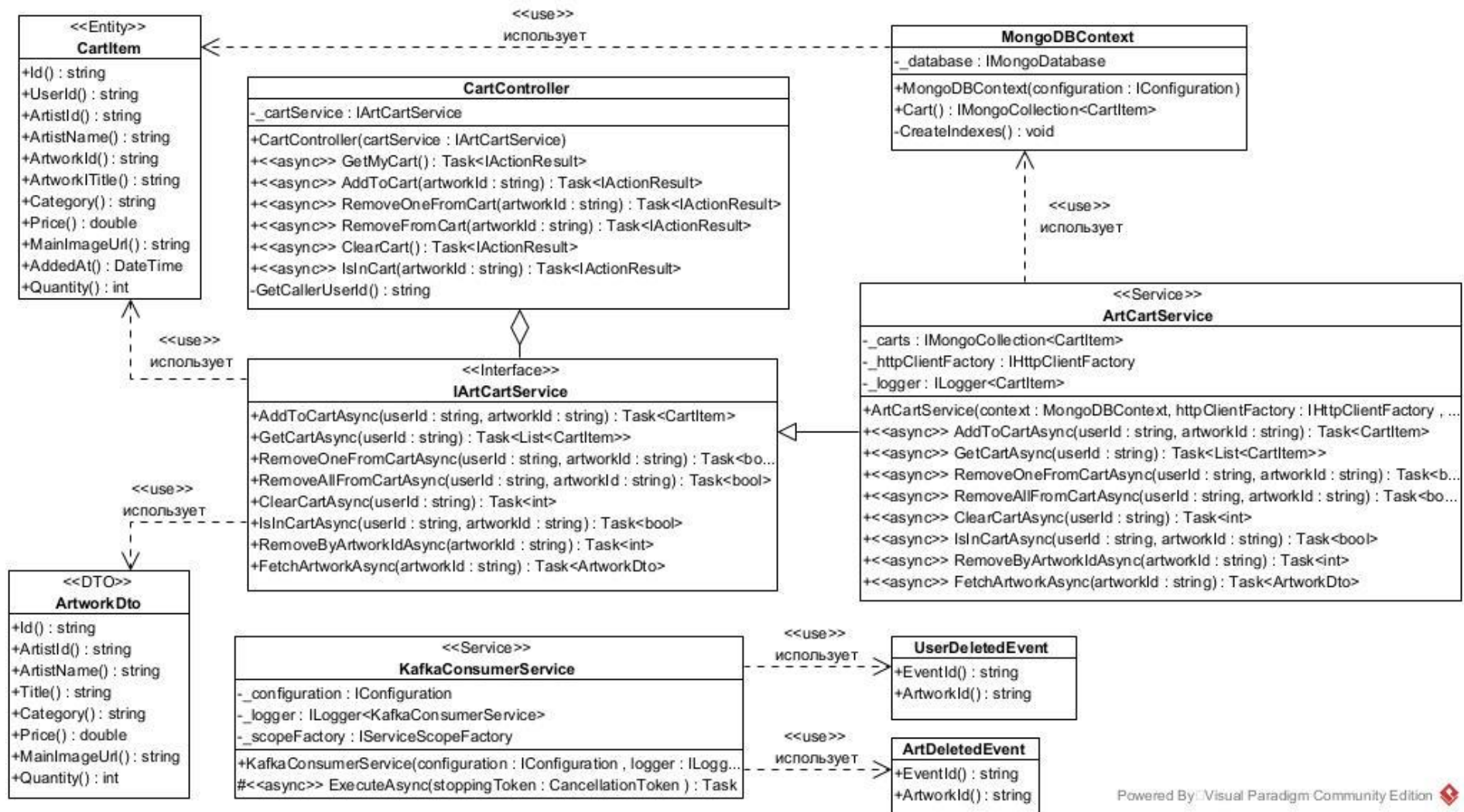


Рисунок В.6 – Диаграмма классов для микросервиса «Корзина»

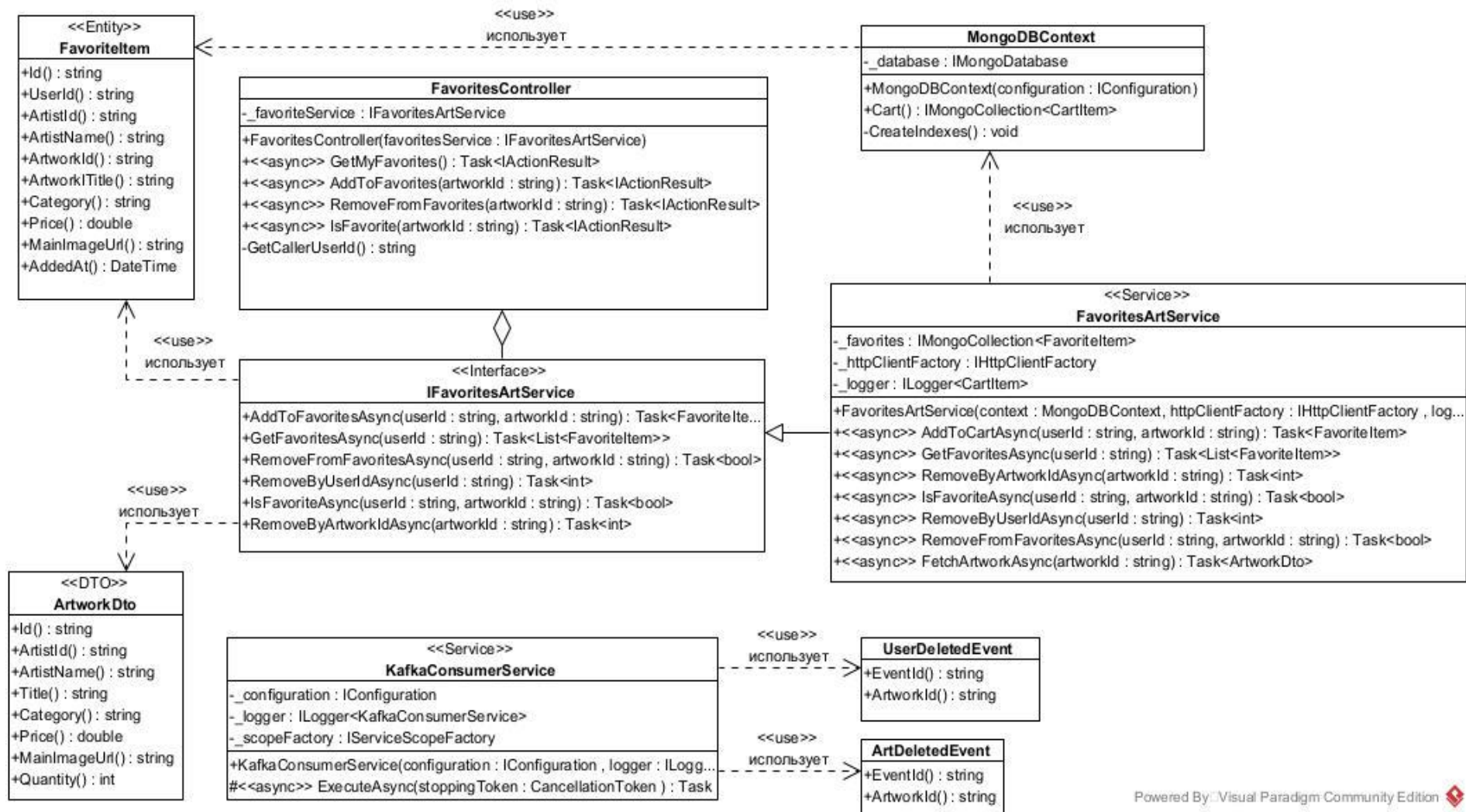


Рисунок В.7 – Диаграмма классов для микросервиса «Избранное»

Приложение Г
(рекомендуемое)
Результаты тестирования

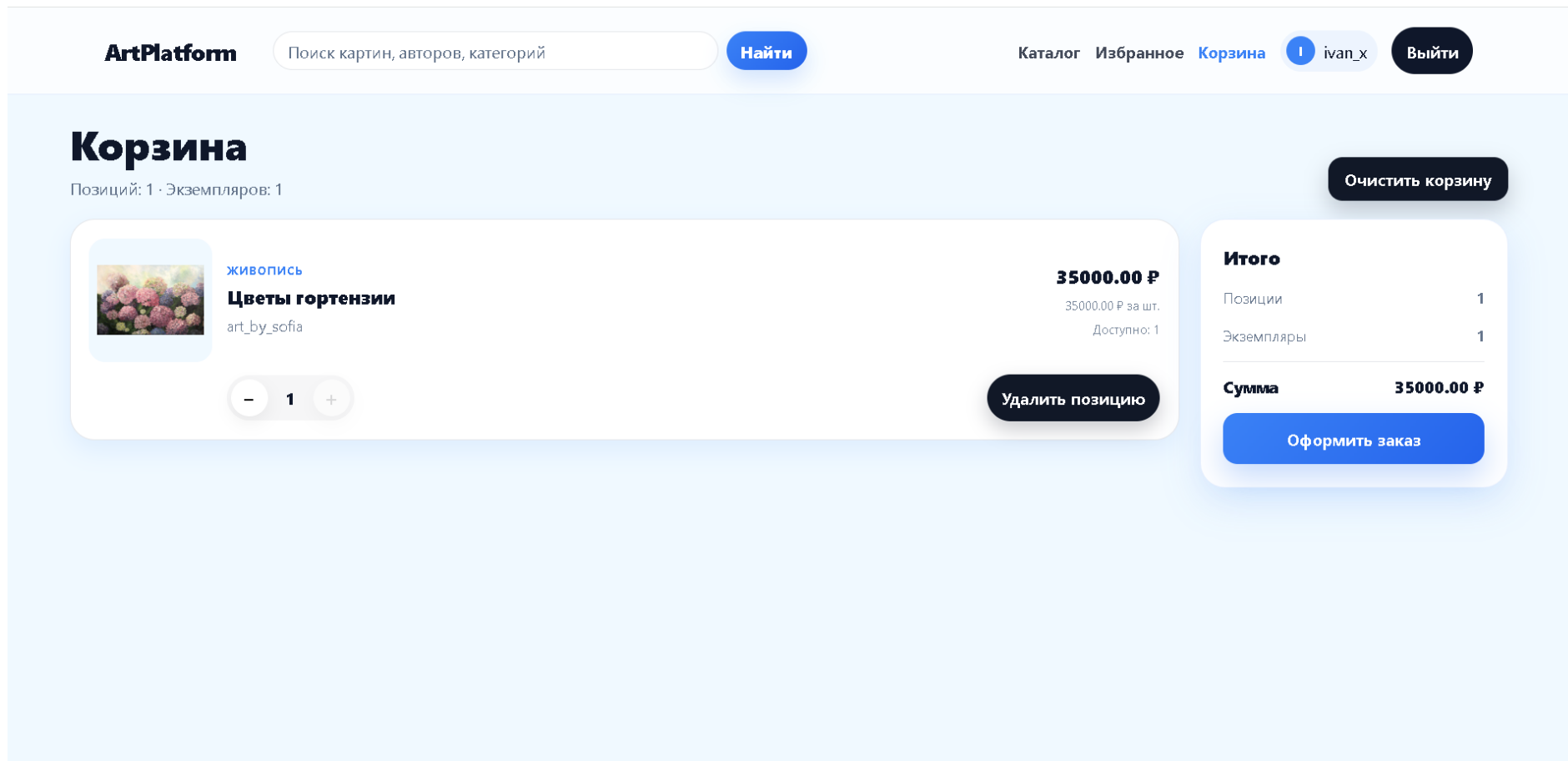


Рисунок Г.1 – Результат тестирования критерия «Покрытие операторов» с истинными условиями «Пользователь авторизован» и «Картина есть в наличии»

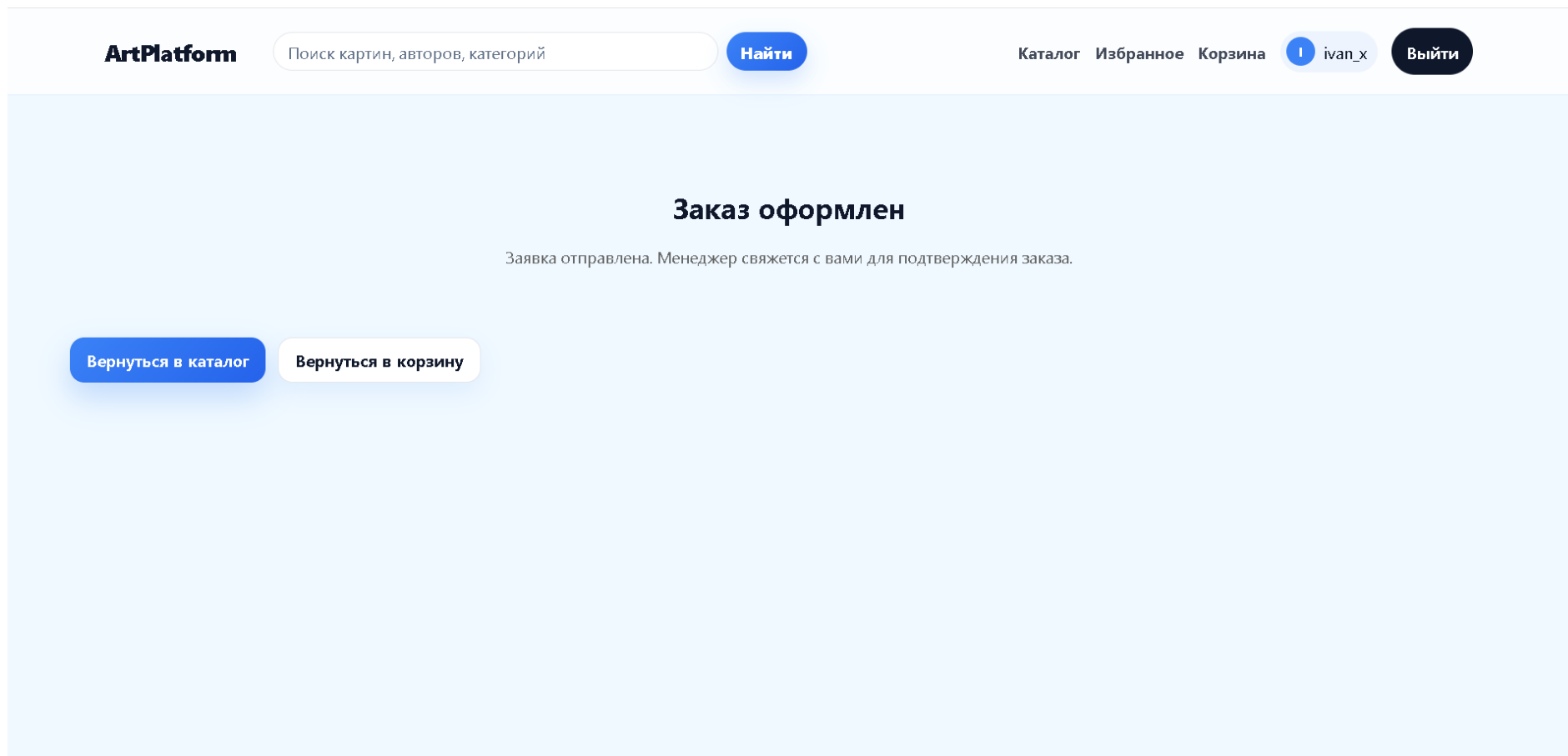


Рисунок Г.2 – Продолжение тестирования критерия «Покрытие операторов» с истинными условиями «Пользователь авторизован» и «Картина есть в наличии»

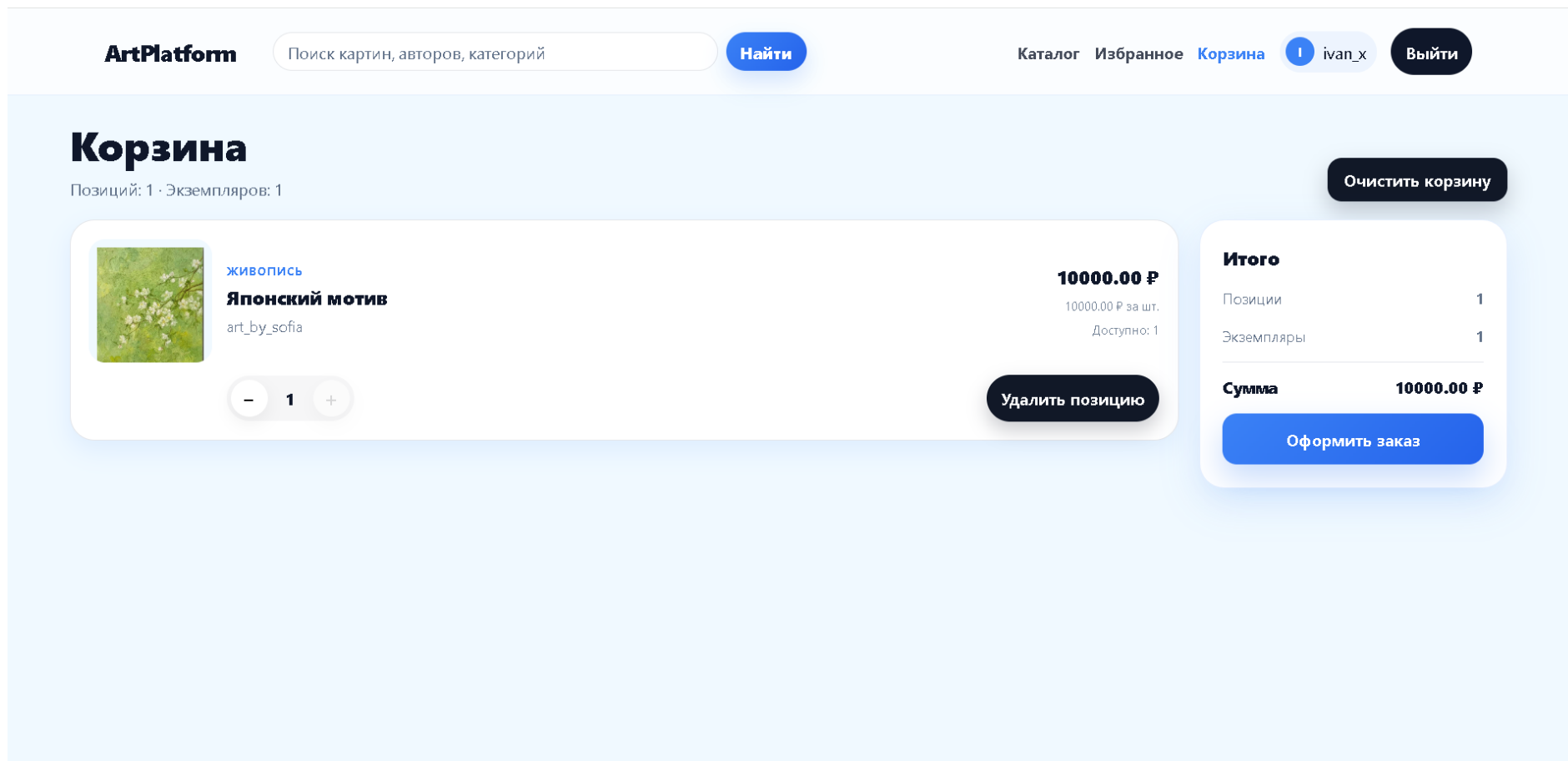


Рисунок Г.3 – Результат тестирования критерия «Покрытие решений» с истинными условиями «Пользователь авторизован» и «Картина есть в наличии»

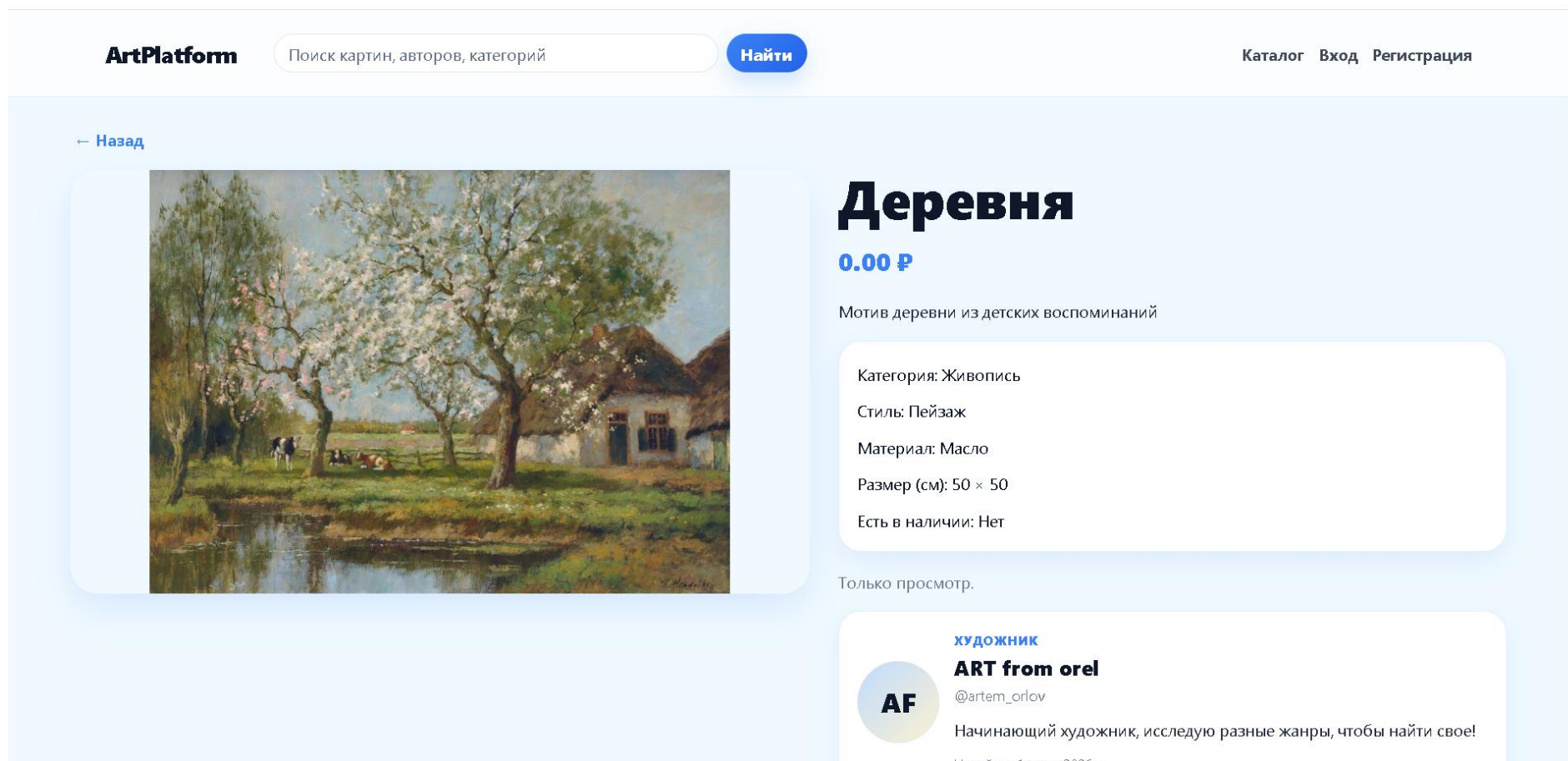


Рисунок Г.4 – Результат тестирования критерия «Покрытие решений» с истинными «Картина есть в наличии» и ложным «Пользователь авторизован»

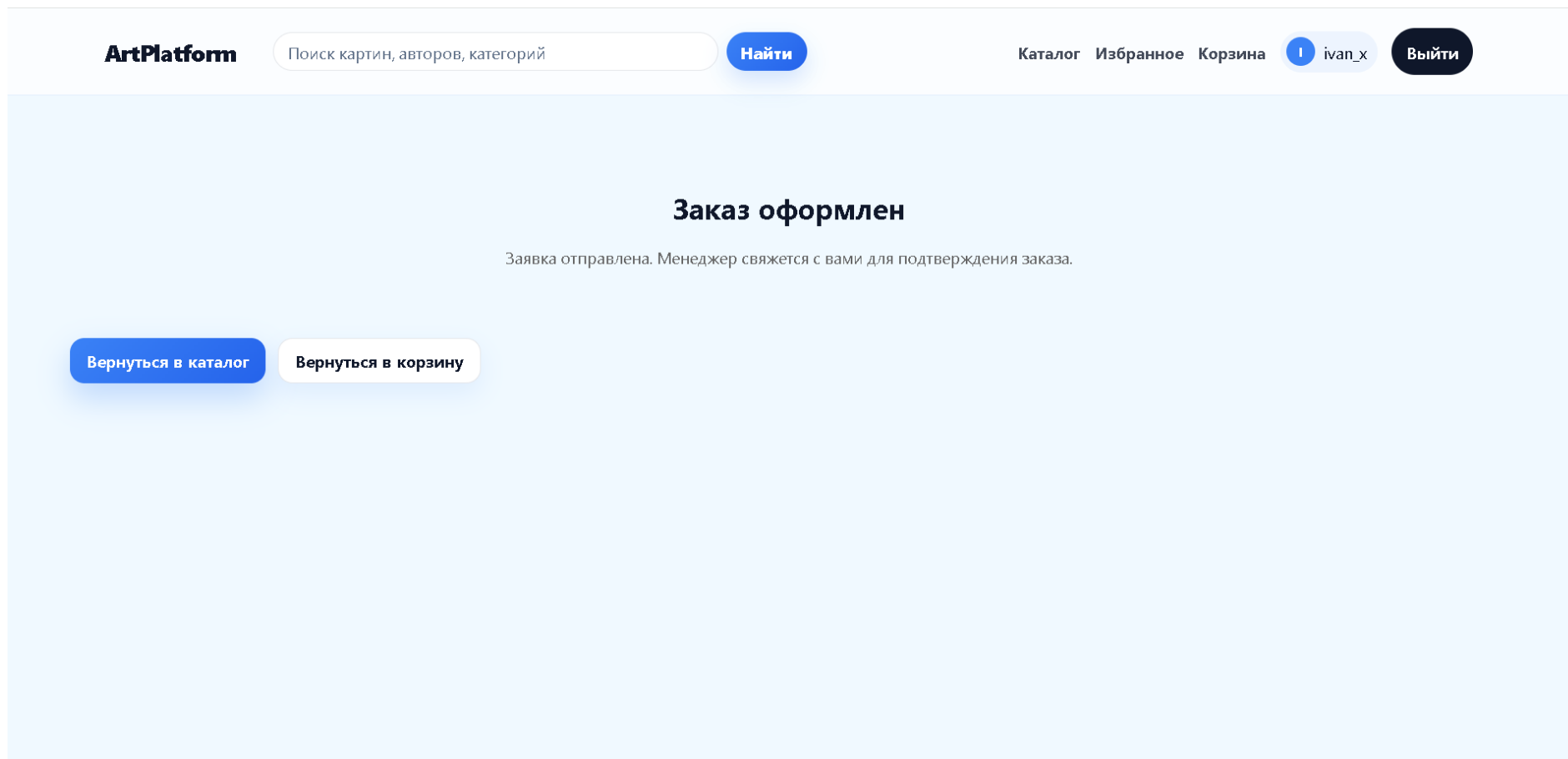


Рисунок Г.5 – Результат тестирования критерия «Покрытие условий» с истинными условиями «Пользователь авторизован» и «Картина есть в наличии»

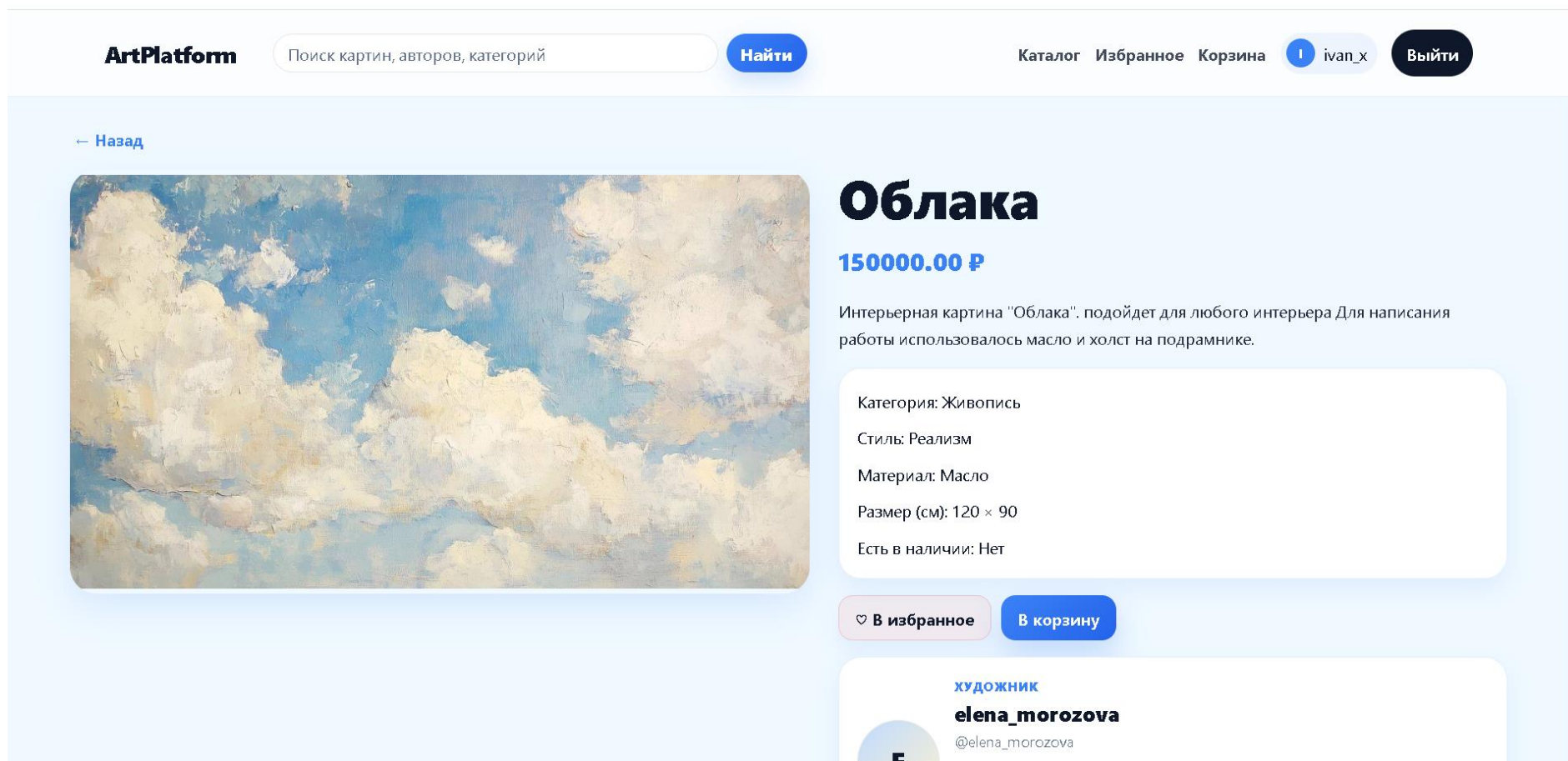


Рисунок Г.6 – Результат тестирования критерия «Покрытие условий» с ложным «Картина есть в наличии» и истинными «Пользователь авторизован»

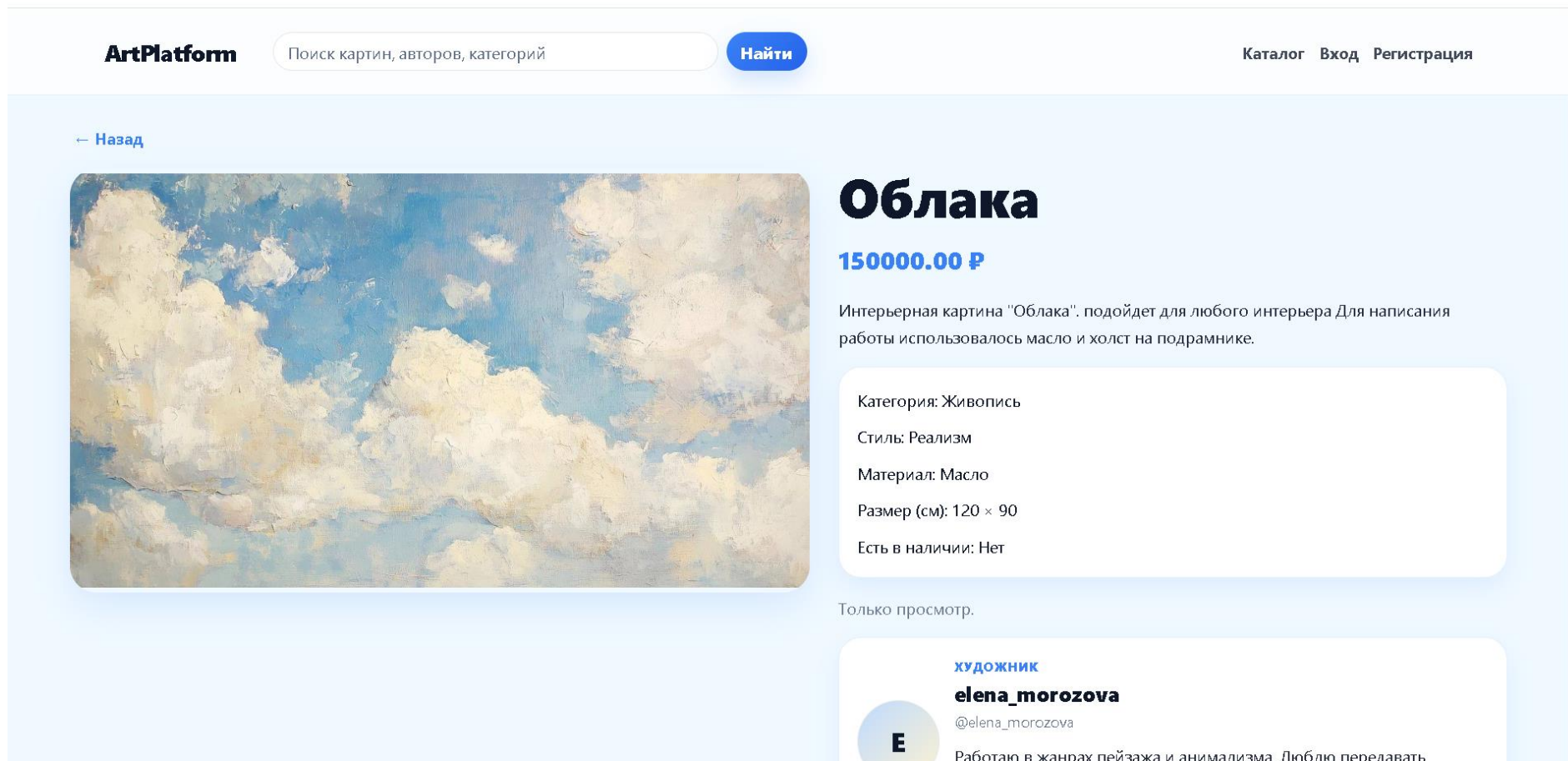


Рисунок Г.7 – Результат тестирования критерия «Покрытие условий» с истинными «Картина есть в наличии» и ложным «Пользователь авторизован»

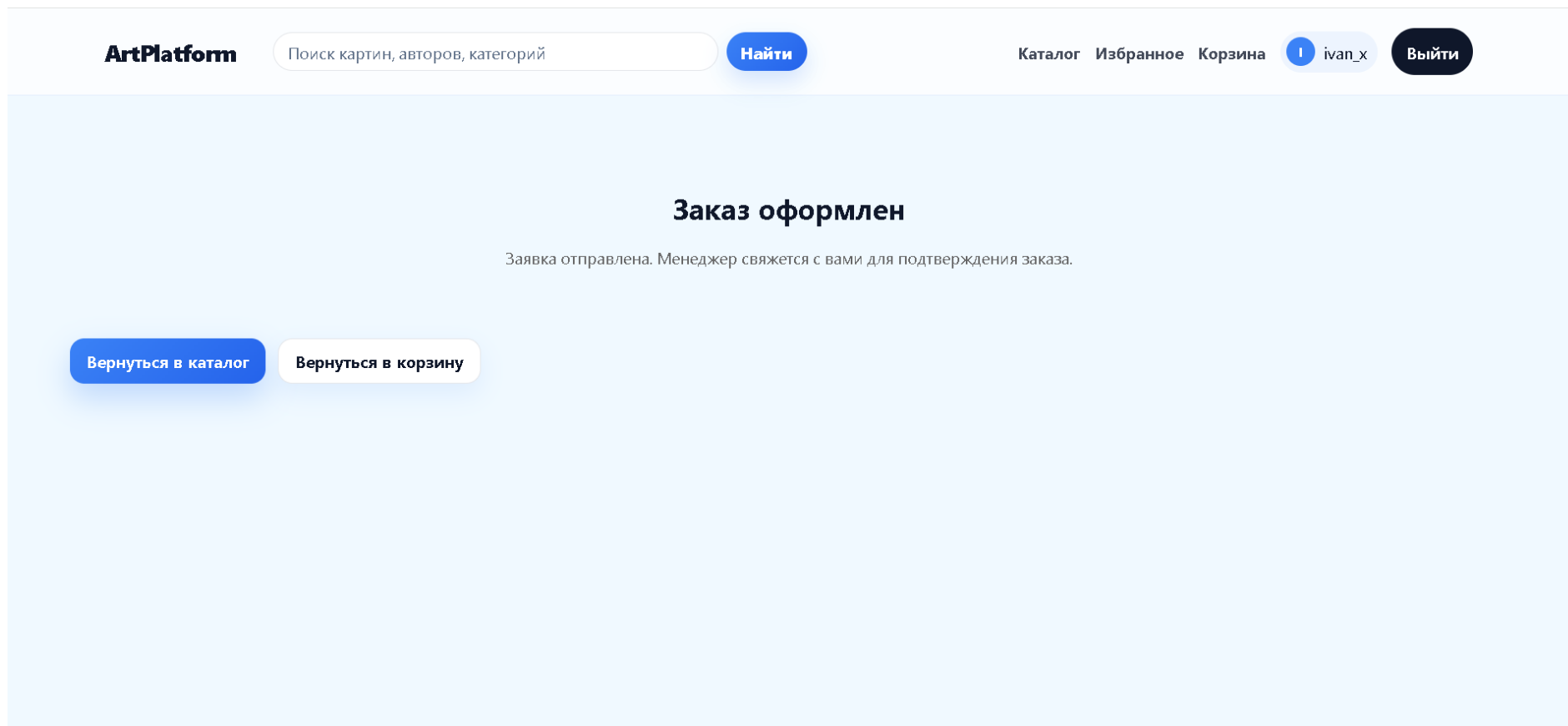


Рисунок Г.8 – Результат тестирования критерия «Покрытие решений и условий» с истинными условиями «Пользователь авторизован» и «Картина есть в наличии»

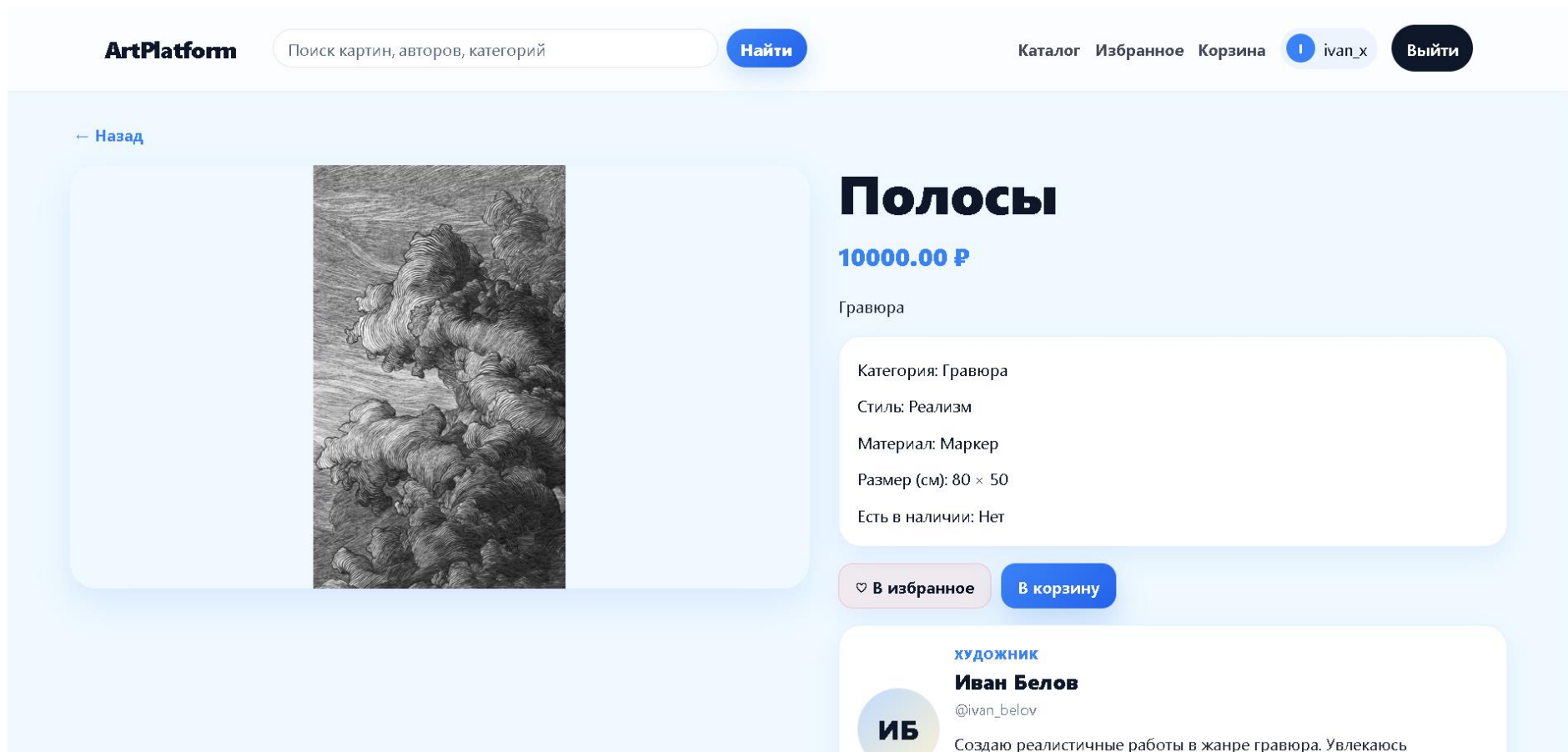


Рисунок Г.9 – Результат тестирования критерия «Покрытие решений и условий» с истинными условиями «Пользователь авторизован» и ложным «Картина есть в наличии»

ArtPlatform

Поиск картин, авторов, категорий

Найти

КаталогВходРегистрация

← Назад





Персики

30000.00 Р

Картина выполнена акварелью на плотной белой бумаге. Картина покрыта бесцветным матовым лаком для защиты работы.

Категория: Живопись

Стиль: Экспрессионизм

Материал: Акварель

Размер (см): 80 × 50

Есть в наличии: Да

Только просмотр.

художник

Анна Ветрова

@anna_vetrova

Пишу акварелью цветочные композиции и натюрморты. В каждой

Рисунок Г.10 – Результат тестирования критерия «Покрывте решений и условий» с ложным условием «Пользователь авторизован» и истинным «Картина есть в наличии»

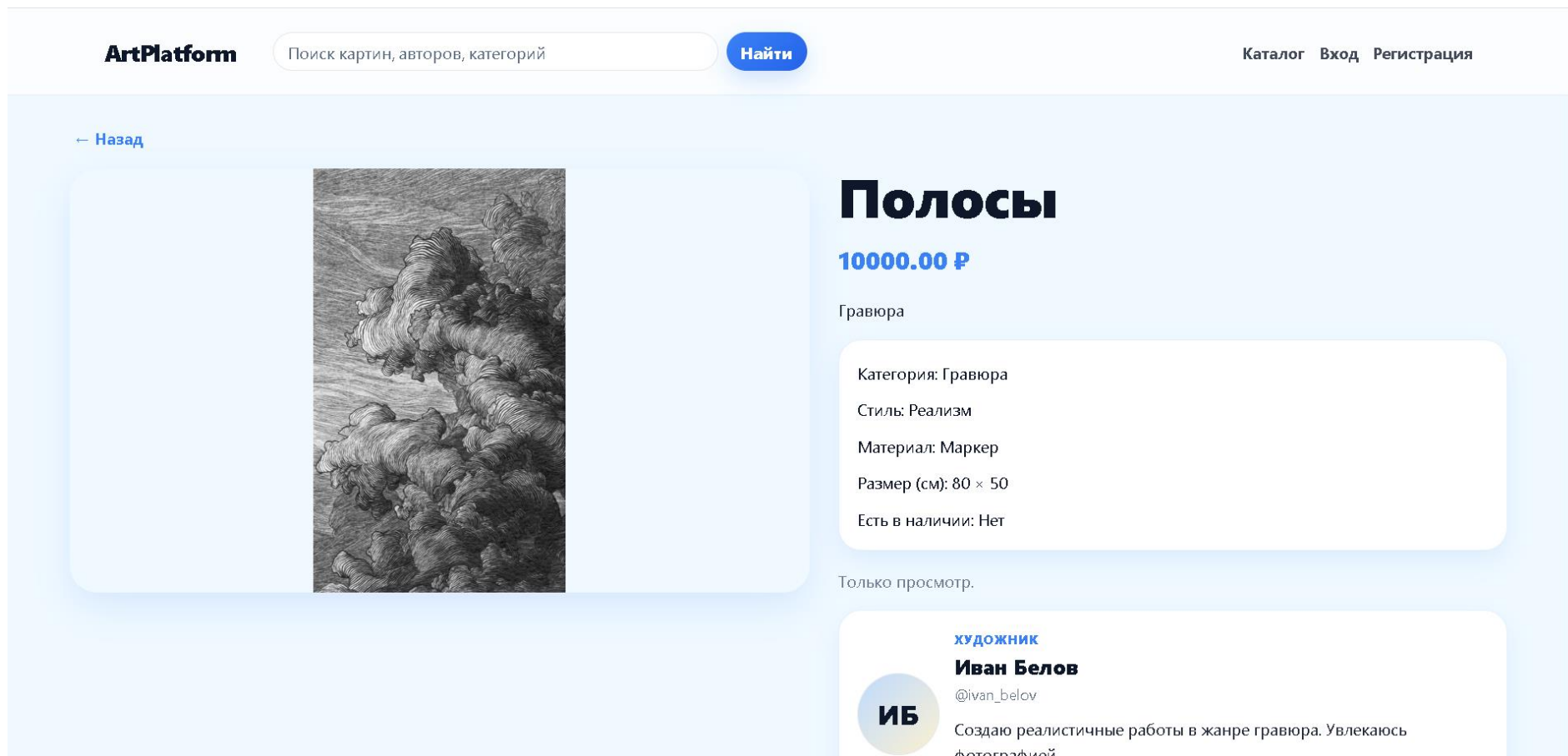


Рисунок Г.11 – Результат тестирования критерия «Покрытие решений и условий» с ложными условиями «Пользователь авторизован» и «Картина есть в наличии»