

Министерство образования Республики Беларусь  
Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Объектно-ориентированное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовой работе  
на тему

СЕРВЕРНАЯ ЧАСТЬ ВЕБ-ПРИЛОЖЕНИЯ АУКЦИОНА

БГУИР КП 1-40 04 01

Студент: гр. 253505 Азаров Е. А.

Руководитель: Тушинская Е. В.

Минск 2024

## СОДЕРЖАНИЕ

Введение . . . . .	3
1 Анализ предметной области . . . . .	4
1.1 Обзор аналогов . . . . .	4
1.2 Постановка задачи . . . . .	5
2 Проектирования программного средства . . . . .	6
2.1 Общая информация . . . . .	6
2.2 Разработка функциональности программного средства . . . . .	6
2.3 Архитектура программного средства . . . . .	7
3 Разработка программного средства . . . . .	8
3.1 Описание моделей данных . . . . .	8
3.2 Реализация слоя инструментов для хранения данных . . . . .	9
3.3 Реализация бизнес-логики . . . . .	10
3.4 Реализация веб-представления . . . . .	11
4 Проверка работоспособности приложения . . . . .	13
5 Руководство пользователя . . . . .	15
Заключение . . . . .	17
Список использованных источников . . . . .	18
Приложение А . . . . .	19
Приложение Б . . . . .	24

# ВВЕДЕНИЕ

Серверная часть приложения аукциона - это важная часть системы, которая обеспечивает автоматизацию основных процессов аукциона. Она позволяет упростить и оптимизировать процессы хранения, создания, поиска товаров аукциона, а также управление информацией о пользователях и их ставках.

Целью данного курсового проекта является создание и разработка серверной части приложения аукциона, которая позволит автоматизировать основные процессы аукциона. Это включает в себя упрощение и оптимизацию процессов хранения, поиска, создания аукционов, создания ставок.

Задачи данного курсового проекта:

- провести анализ требований к приложению аукциона, определить функциональные требования;
- разработать архитектуру приложения, определить ее основные компоненты и интерфейс;
- разработать и реализовать программное обеспечение;
- сформулировать выводы, исходя из нашей работы.

В данном курсовом проекте будут рассмотрены такие главы, как анализ предметной области, проектирование программного средства, разработка программного средства, проверка работоспособности приложения и руководство пользователя.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Обзор аналогов

Веб-сайт eBay - это международная онлайн-аукционная платформа, которая позволяет пользователям покупать и продавать товары. eBay была основана в 1995 году и является одним из крупнейших онлайн-аукционов в мире.

Система eBay предлагает широкий спектр функций. В частности данная платформа позволяет пользователям создавать аукционы на различные товары, включая антиквариат, коллекционные предметы, электронные устройства и многое другое. Пользователи могут делать ставки на товары, а также покупать товары в режиме реального времени.

Кроме того, eBay Seller Center предоставляет широкие возможности для интеграции с внешними системами и сервисами. Это дает продавцам возможность автоматизировать многие рутинные операции и повысить эффективность своей работы на eBay. Скриншот веб-сайта «eBay» указан на рисунке 1.1.

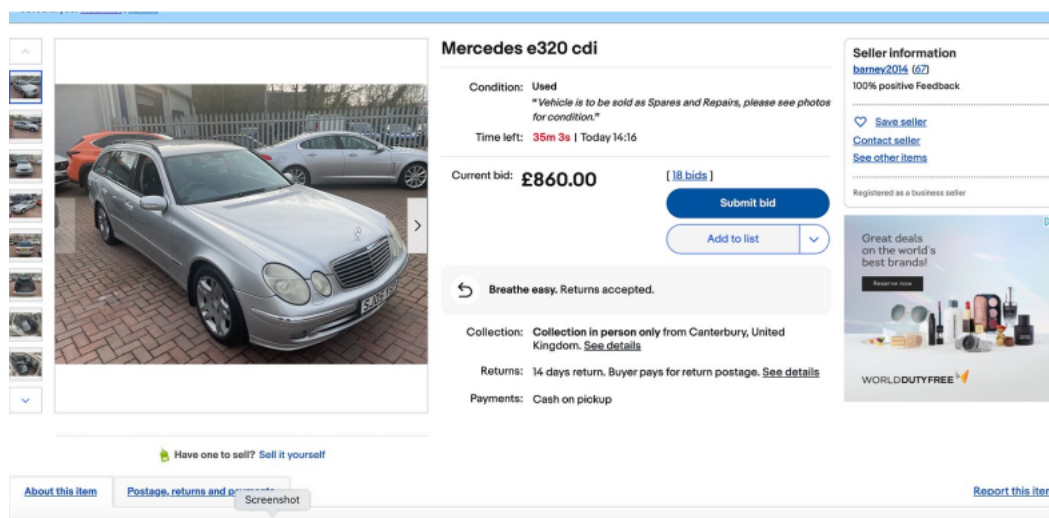
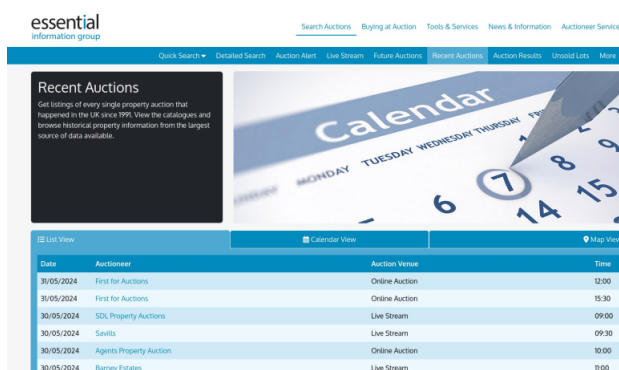


Рисунок 1.1 – Скриншот веб-сайта «eBay»

Auction House Management System (AHMS) - это комплексное серверное приложение, предназначенное для управления деятельностью профессиональных аукционных домов. Оно охватывает весь цикл аукционного процесса: от каталогизации лотов и регистрации участников до проведения торгов, осуществления расчетов и формирования всесторонней отчетности. Также

данная система может предоставить полную историю торгов. Скриншот проведения аукционов в системе AHMS показан на рисунке 1.2.



Date	Auctioneer	Auction Venue	Time
30/05/2024	First for Auctions	Online Auction	12:00
30/05/2024	First for Auctions	Online Auction	15:30
30/05/2024	SDL Property Auctions	Live Stream	09:00
30/05/2024	Savills	Live Stream	09:30
30/05/2024	Agents Property Auction	Online Auction	10:00
30/05/2024	Barney Estates	Live Stream	11:00

Рисунок 1.2 – Скриншот системы «Auction House Management System»

Анализ данных средств, позволяет выявить функциональные требования к нашей системе.

## 1.2 Постановка задачи

В рамках данного курсового проекта была поставлена задача: разработать серверную часть системы управления аукционом.

Был выделен следующий ряд подзадач:

- описать модели данных проекта, включая модели для лотов, категорий лотов, ставок, пользователей;
- реализовать механизмы сохранения, чтения, обновления и удаления данных в инструментах хранения данных;
- реализовать функционал создания аукциона, создание ставок;
- реализовать механизмы авторизации и аутентификации по ролям;
- реализовать механизмы администрирования системы аукциона;
- разработать приложение в формате веб-приложения;
- реализовать механизмы уведомления пользователей об изменном состоянии аукциона в режиме реального времени;
- разработать интерактивную документацию, которая будет содержать подробное описание функциональности системы, API-интерфейсов.

Разработав данный набор задач можно перейти непосредственно к проектированию программного средства.

## **2 ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО СРЕДСТВА**

### **2.1 Общая информация**

Для реализации системы будет использован язык программирования C# 8. Он предоставляет высокую производительность и эффективность разработки, что особенно важно для системы, которая должна обрабатывать большое количество запросов на поиск и выдачу ответов. Проект будет разрабатываться на операционной системе linux, которая является одной из популярных операционных систем.

В качестве СУБД будет использована SQLite3, а также для работы с ней будет задействована технология с поддержкой ORM (object-relational mapping - отображения данных на реальные объекты) Entity Framework Core.

Для обработки пользовательских REST (передача состояния представления) запросов будет использована платформа ASP.NET, а также уведомление пользователей об измененном состоянии системы аукциона в режиме реального времени будет проходить с применением библиотеки SignalR, используя WebSockets, Server-Sent Events и Long Polling.

Для удобства разработки будет задействована система контроля версий Git и платформа GitHub для хостинга репозитория. Это позволит легко отслеживать изменения в коде и управлять различными версиями. Кроме того, будет использоваться редактор кода Visual Studio Code (Vs Code), который обладает множеством полезных функций, таких как автодополнение кода, отладка и интеграция с Git, что создаст комфортную среду разработки и повысит производительность.

Для тестирования приложения будет использован программное средство Postman. Postman - это популярный инструмент для тестирования API, который позволяет отправлять запросы к серверу и получать ответы. Он также может быть использован для документирования API и создания запросов.

### **2.2 Разработка функциональности программного средства**

При анализе требований для реализации программного продукта для тестирования была построена use-case диаграмма, которая отображает взаимодействие между пользователем и приложением. На рисунке 2.1, представленном ниже, отображены основные возможности использования приложения.

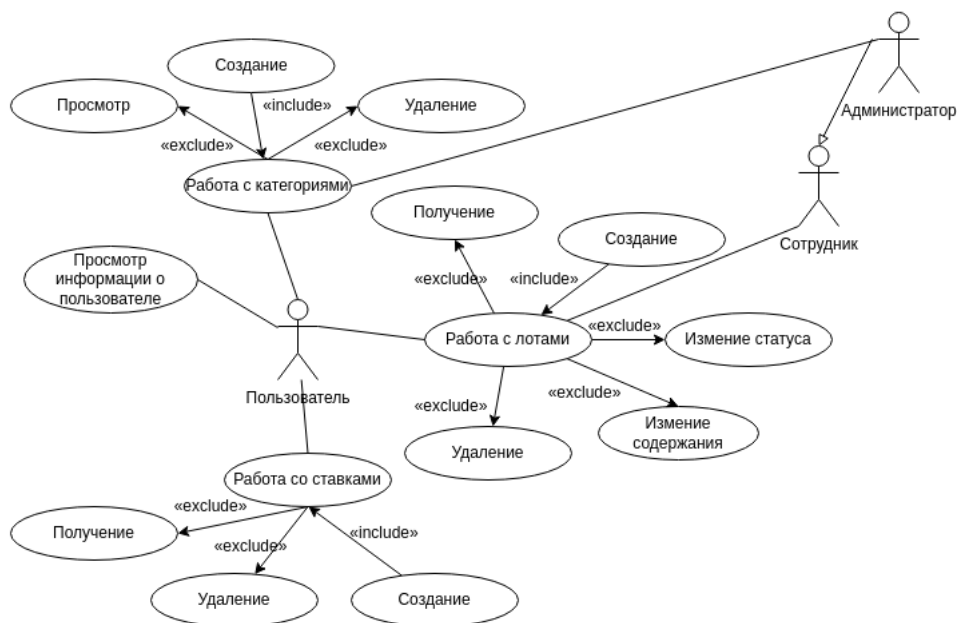


Рисунок 2.1 – Use-case диаграмма приложения

Таким образом рассмотрены возможности использования от лица обычного пользователя, от лица администратора и сотрудника.

## 2.3 Архитектура программного средства

В качестве архитектуры приложения была выбрана трёхуровневая архитектура. В приложении можно выделить следующие слои:

- слой инфраструктуры – обеспечивает доступ к данным, реализует паттерны репозиторий и Unit of Work, содержит реализацию абстракции репозитория и классов для работы непосредственно с базой данных. Вся работа с базой данных происходит в репозиториях, которые объединены классом Unit of Work, организующим всю работу с базой данных;

- слой приложения – полностью независимый слой, который определяет модели и абстракции для работы с базой данных, а также содержит бизнес-логику системы аукциона.

- слой представления – содержит реализованный с использованием платформы ASP.NET Core веб-интерфейс приложения. При помощи библиотеки SignalR реализует обмен сообщениями об измененном состоянии системы аукциона между клиентом и сервером. Также данный слой реализует систему авторизации и аутентификации пользователей.

### 3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

#### 3.1 Описание моделей данных

Диаграмма моделей данных указана на рисунке 3.1.

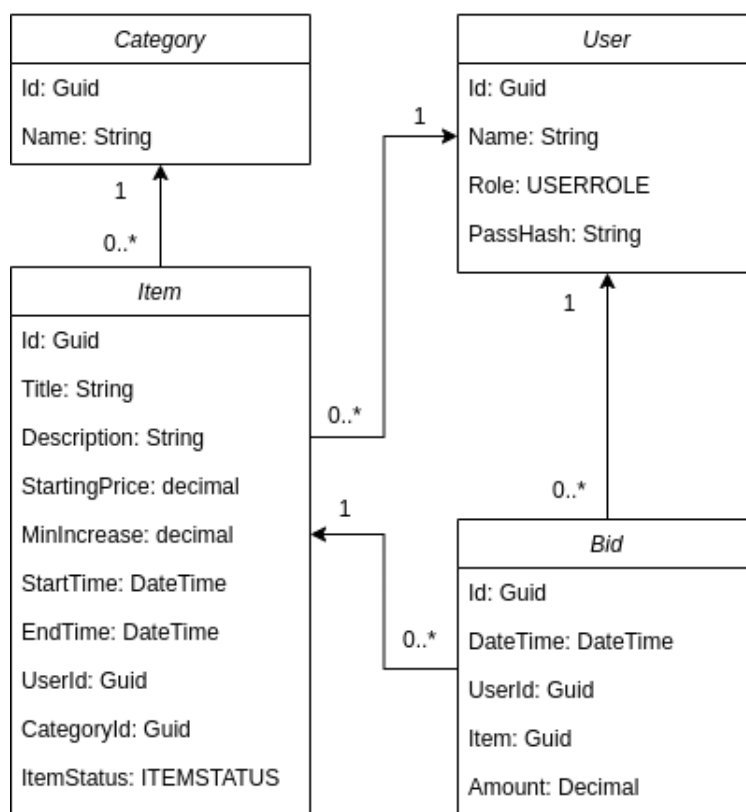


Рисунок 3.1 – Диаграмма моделей данных приложения и их полей

Модель User предназначена для представления информации, необходимой для аутентификации и авторизации пользователей системы. Она содержит уникальный идентификатор (Id), имя пользователя (Name), хеш пароля (PassHash) и роль пользователя (Role). Идентификатор служит для однозначной идентификации каждого пользователя, имя и хеш пароля используются для аутентификации, а роль определяет права доступа пользователя к различным функциям системы. Эта модель играет ключевую роль в обеспечении безопасности и контроля доступа к системе, позволяя управлять пользователями и их привилегиями.



Модель Category представляет собой сущность для хранения информации о категориях лотов в системе. Она содержит уникальный идентификатор (Id) категории реализованный в виде Guid, который позволяет однозначно идентифицировать каждую категорию, и Name, хранящее название категории. Эта модель играет ключевую роль в организации и структурировании лотов пользователей в системе, позволяя группировать элементы по соответствующим категориям и обеспечивая логическую классификацию и удобную навигацию по информации.

Модель Item представляет собой сущность, предназначенную для хранения подробной информации об элементах, выставляемых в системе аукциона. Она включает в себя уникальный идентификатор Id, название Title и описание Description элемента, начальную цену StartingPrice, минимальный шаг увеличения MinIncrease, даты начала StartTime и окончания EndTime аукциона, идентификатор пользователя UserId, выставившего элемент, категорию CategoryId, к которой он относится, а также статус элемента ItemStatus в системе аукциона. Эта модель обеспечивает всестороннее описание объектов, участвующих в коммерческих операциях системы, и служит важным элементом ее функционирования.

Модель Bid представляет собой сущность, которая хранит информацию о ставках, сделанных пользователями в рамках аукционных процессов системы. Она включает в себя уникальный идентификатор Id для каждой ставки, размер ставки Amount, идентификаторы пользователя UserId, сделавшего ставку, и элемента ItemId, на который была сделана ставка, а также дату и время DateTime, когда ставка была сделана. Эта модель играет ключевую роль в отслеживании и управлении ставками, что является основополагающим аспектом функционирования аукционных механизмов в системе, позволяя хранить историю торгов.

### **3.2 Реализация слоя инструментов для хранения данных**

Слой инфраструктуры обеспечивает доступ приложения к данным посредством реализации паттерна репозиторий, а также облегчает работу с репозиториями посредством реализации паттерна UnitOfWork. Рассмотрим классы, находящиеся на данном слое.

AppDbContext – класс, наследующийся от базового Entity Framework Core класса DbContext. В данном классе задается структура таблиц базы данных и проверяется существование базы данных при подключении к ней. В случае отсутствия необходимой базы данных, создается пустая база данных

с требуемыми параметрами таблиц. `EfRepository<T>` – шаблонный класс, реализующий интерфейс `IRepository`, который описывается в слое приложения. Он имеет доступ к таблице базы данных, содержащей объекты типа `T` и содержит следующие методы:

- `GetByIdAsync()` – метод, возвращающий объект из базы данных с требуемым идентификатором. В случае передачи соответствующих параметров также может вместе с самим объектом вернуть объекты из других таблиц, связанных с ним внешними ключами;

- `ListAllAsync()` – возвращает все объекты из таблицы базы данных, связанной с данным конкретным репозиторием;

- `ListAsync()` – возвращает все объекты из таблицы базы данных, соответствующие переданному фильтру. В случае передачи соответствующих параметров также может вместе с каждым из них вернуть объекты из других таблиц, связанных с ним внешними ключами;

- `AddAsync()` – добавляет переданный объект в базу данных;

- `UpdateAsync()` – изменяет переданный объект в базе данных;

- `DeleteAsync()` – удаляет переданный объект из базы данных;

- `FirstOrDefaultAsync()` – возвращает первый объект из базы данных, соответствующий переданному фильтру. В случае отсутствия такого объекта в базе данных, возвращает `null`. `EfUnitOfWork` – класс, реализующий интерфейс `IUnitOfWork`, который описывается в слое бизнес-логики. Содержит в себе объекты класса `EfRepository` для каждой модели, а также методы `CreateDataBaseAsync()` и `DeleteDataBaseAsync()`, которые создают и удаляют базу данных, а также метод `SaveAllAsync()`, который сохраняет внесенные в базу данных изменения.

### 3.3 Реализация бизнес-логики

Бизнес-логику описывают следующие интерфейсы:

- `IAuthService` – служит для управления авторизацией и аутентификацией пользователей. Он предлагает два метода: `Register`, который регистрирует пользователя с указанным именем пользователя и паролем, и `SignIn`, который проверяет вход пользователя с указанным именем пользователя и паролем.

- `IUserService` – служит для управления информацией о пользователях. Он предлагает три метода: `GetUser`, который возвращает информацию о пользователе с указанным именем пользователя, `GetUserById`, который возвращает информацию о пользователе с указанным идентификатором пользо-

вателя, и `GetUserName`, который возвращает имя пользователя с указанным идентификатором пользователя.

– `ICategoryService` – служит для управления информацией о категориях. Он предлагает шесть методов: `GetAll`, который возвращает список всех категорий, `GetCategory`, который возвращает информацию о категории с указанным идентификатором категории или именем категории, `CreateCategory`, который создает новую категорию с указанным именем, `ChangeCategory`, который изменяет информацию о категории с указанным идентификатором категории, `DeleteCategory`, который удаляет категорию с указанным идентификатором, и `GetCategoryName`, который возвращает имя категории с указанным идентификатором категории.

– `ItemService` – служит для управления информацией о товарах. Он предлагает восемь методов: `GetItem`, который возвращает информацию о товаре с указанным идентификатором, `GetItemsByCategory`, который возвращает список товаров, относящихся к указанной категории, `GetItemsByUserId`, который возвращает список товаров, созданных пользователем с указанным идентификатором пользователя, `GetItemsByUser`, который возвращает список товаров, созданных пользователем с указанным именем, `CreateItem`, который создает новый товар с указанными параметрами, `ChangeItemStatus`, который изменяет статус товара, `ChangeItemDetails`, который изменяет информацию о товаре, и `CloseItem`, который закрывает товар с указанным идентификатором товара.

– `IBidService` – служит для управления информацией о ставках. Он предлагает четыре метода: `GetBidsByItem`, который возвращает список ставок, сделанных на определенный товар, `GetBidsByUser`, который возвращает список ставок, сделанных пользователем с указанным именем пользователя, `CreateBid`, который создает новую ставку с указанными параметрами, и `DeleteBid`, который удаляет ставку с указанным идентификатором ставки.

### **3.4 Реализация веб-представления**

Данный слой реализован с использованием фреймворка ASP.NET Core. На данном слое содержатся классы для контроллеров веб-интерфейса приложения, определения моделей для обмена данными с клиентом и класс для преобразования доменных моделей в DTO модели, класс для работы с узлом подключения к приложению для обмена данными между различными клиентами в реальном времени, а также точка входа в приложения. Рассмотрим отдельно каждую из этих групп классов.

Классы контроллеров веб-интерфейса приложения предоставляют публичное API (программный интерфейс приложения) для организации доступа к данным приложения клиентам. Вся внутренняя реализация методов данных классов взаимодействует только со слоем приложения, к каждому из них через внедрение зависимостей добавлен соответствующий сервис для решения определенных задач.

- `AuthController` – данный класс содержит в себе только два метода: `register()` и `login()`, которые соответственно дают доступ к регистрации и входу в аккаунт пользователя.

- `UserController` – отвечает за обработку запросов, связанных с пользователями, и использует `IUserService` для взаимодействия с бизнес-логикой и `IMapper` для преобразования доменных моделей в модели для предоставления данных пользователю.

- `CategoryController` – управляет категориями в приложении аукциона и использует `ICategoryService` для взаимодействия с бизнес-логикой и `IMapper` для преобразования доменных моделей в модели для предоставления данных пользователю.

- `ItemController` – управляет товарами в приложении аукциона и использует `IItemService` для взаимодействия с бизнес-логикой, `IHubContext` для отправки сообщений в реальном времени и `IMapper` для преобразования доменных моделей в модели для предоставления данных пользователю.

- `BidController` – управляет ставками в приложении аукциона и использует `IBidService` для взаимодействия с бизнес-логикой, `IHubContext` для отправки сообщений в реальном времени и `IMapper` для преобразования доменных моделей в модели для предоставления данных пользователю.

Для уведомления пользователей об изменениях состояния системы аукциона используется класс `AuctionHub`, который является частью приложения аукциона и отвечает за управление соединениями клиентов и отправку сообщений в реальном времени. Данный класс наследует класс `Hub`, который является частью библиотеки `SignalR`.

Далее рассмотрим класс преобразования доменных моделей в модели для предоставления данных пользователю. Данный процесс реализован с помощью библиотеки `AutoMapper`.

В точке входа в приложении происходит подключение всех слоев приложения друг другу через внедрение зависимостей, подключение внешних зависимостей, подключение приложения к базе данных и непосредственно запуск веб-сервера.

## **4 ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРИЛОЖЕНИЯ**

Осуществлялось функциональное тестирование, ниже рассмотрим тестирование каждой функции.

1 Тестирование функции регистрации – ожидаемый результат, появление нового пользователя в таблице базы данных при передаче уникального имени, соответствует полученному.

2 Тестирование функции авторизации – ожидаемый результат, предоставление валидного токена авторизации, соответствует полученному.

3 Тестирование функции просмотра категорий – ожидаемый результат, получения списка категорий, соответствует полученному.

4 Тестирование функции просмотра информации о пользователе – ожидаемый результат, получение объекта, описывающего пользователя, соответствует полученному.

5 Тестирование функции просмотра лотов определенной категории – ожидаемый результат, получения списка объектов данной категории, соответствует полученному.

6 Тестирование функции удаления категории – ожидаемый результат, удаление из таблицы базы данных категории пользователем в роли администратора, соответствует полученному.

7 Тестирование функции просмотра лотов определенного пользователя – ожидаемый результат, получения списка объектов лотов данного пользователя, соответствует полученному.

8 Тестирование функции просмотра собственных лотов – ожидаемый результат, получения списка объектов лотов данного авторизованного пользователя, соответствует полученному.

9 Тестирование функции просмотра ставок определенного пользователя – ожидаемый результат, получение списка объектов ставок, созданных данным пользователем, соответствует полученному.

10 Тестирование функции изменение статуса лота – ожидаемый результат, изменение поля ItemStatus в таблице базы данных, при выполнении запроса авторизованного пользователя с ролью администратора, соответствует полученному.

11 Тестирование функции завершения аукциона – ожидаемый результат, изменение статуса лота в таблице базы данных и уведомление пользователей в режиме реального времени, соответствует полученному.

12 Тестирование функции создания лота – ожидаемый результат, со-

здание новой сущности в таблице базы данных при передаче корректных данных и извещение администратора, соответствует полученному. Ниже на рисунке 4.1 показан результат, полученный в программном средстве Postman.



Рисунок 4.1 – Тестирование уведомления администратора при создании нового лота

13 Тестирование функции удаления лота – ожидаемый результат, удаление соответствующей сущности из таблицы в базе данных , соответствует полученному.

14 Тестирование функции изменения лота – ожидаемый результат, изменение сущности в таблице базы данных при передаче корректных данных и извещение администратора об изменении, соответствует полученному. Ниже на рисунке 4.2 показан результат, полученный в программном средстве Postman.



Рисунок 4.2 – Тестирование уведомления администратора при изменении лота

Таким образом, на основе выполненных тестов можно сделать вывод о том, что программное средство работает исправно и готово к дальнейшему использованию.

## 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Требуется убедиться, что серверная часть системы настроена. Для начала работы необходимо запустить приложение на заранее известном ip адресе и порту.

После запуска приложения пользователь получит доступ к интерактивной документации. Чтобы получить доступ к интерактивной документации системы, введите в адресной строке браузера URL /swagger. Например, если адрес сервера – `http://localhost:8080`, то введите `http://localhost:8080/swagger`. Нажмите клавишу Enter или выполните запрос, чтобы перейти на страницу с интерактивной документацией.

В интерактивной документации показано описание API-интерфейсов, которые могут использоваться для взаимодействия с системой. Это набор методов, которые клиентские приложения могут вызывать для выполнения различных операций в системе.

Описание API-интерфейсов включает информацию о доступных методах, параметрах, возвращаемых значениях и кодах состояния HTTP.

На рисунках 5.1, 5.2, 5.3, 5.4, 5.5 показана интерактивная документация системы управления авторизацией, пользователями, категориями лотов, лотами, ставками соответственно.

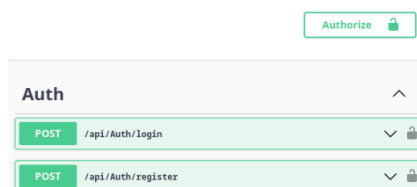


Рисунок 5.1 – Интерактивная документация системы авторизации



Рисунок 5.2 – Интерактивная документация системы управления пользователями

Category			^
GET	/api/Category/all	▼	🔒
GET	/api/Category/{categoryname}	▼	🔒
POST	/api/Category/create	▼	🔒
PUT	/api/Category/{catid}	▼	🔒
DELETE	/api/Category/{catid}	▼	🔒

Рисунок 5.3 – Интерактивная документация системы управления категориями лотов

Item			^
GET	/api/Item/bycategory/{catname}	▼	🔒
GET	/api/Item/{itemid}	▼	🔒
GET	/api/Item/postedby/{username}	▼	🔒
GET	/api/Item/myposted	▼	🔒
POST	/api/Item/create	▼	🔒
PUT	/api/Item/{itemid}/editdetails	▼	🔒
PUT	/api/Item/{itemid}/editstatus/{itemstatus}	▼	🔒
PUT	/api/Item/{itemid}/finish	▼	🔒
DELETE	/api/Item/{itemid}/delete	▼	🔒

Рисунок 5.4 – Интерактивная документация системы управления лотами

Bid			^
GET	/api/Bid/allmybids	▼	🔒
GET	/api/Bid/item/{itemid}	▼	🔒
GET	/api/Bid/user/{username}	▼	🔒
POST	/api/Bid/makebid/item/{itemid}	▼	🔒
DELETE	/api/Bid/{bidid}	▼	🔒

Рисунок 5.5 – Интерактивная документация системы управления ставками



## ЗАКЛЮЧЕНИЕ

В рамках данного проекта была разработана серверная часть системы управления онлайн-аукционом. Основной целью проекта было создание эффективной и удобной платформы для проведения аукционов, которая позволит продавцам и покупателям легко взаимодействовать и совершать сделки.

Реализация проекта состояла из нескольких этапов: разработка требований и проектирование, реализация поставленных задач и тестирование работоспособности приложения.

В ходе тестирования были протестированы все функции системы, включая регистрацию, авторизацию, просмотр категорий, просмотр информации о пользователе, просмотр лотов, удаление категории, просмотр лотов определенного пользователя, просмотр собственных лотов, просмотр ставок определенного пользователя, изменение статуса лота, завершение аукциона, создание лота, удаление лота и изменение лота. В результате тестирования было подтверждено, что все функции системы работают корректно и в соответствии с ожидаемыми результатами.

В результате реализации данного проекта были достигнуты все поставленные цели. Разработанная система управления онлайн-аукционом позволяет повысить эффективность и прозрачность проведения торгов, а также упростить взаимодействие между продавцами и покупателями. Она обеспечивает удобный и интуитивно понятный интерфейс, позволяющий легко ориентироваться в каталоге лотов, делать ставки и отслеживать статус аукционов.

В целом, разработанная система управления онлайн-аукционом является эффективным инструментом для проведения аукционов, который может быть использован в различных областях, включая коммерцию, недвижимость и искусство.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] GitHub [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://github.com/>.

[2] Dotnet guide [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://learn.microsoft.com/en-us/dotnet/csharp/>.

[3] ASP.NET Core [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://dotnet.microsoft.com/en-us/apps/aspnet>.

[4] EF Core Overview [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://learn.microsoft.com/en-us/ef/core/>.

[5] SignalR | Microsoft Learn [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://learn.microsoft.com/en-us/aspnet/signalr/>.

# ПРИЛОЖЕНИЕ А

## (обязательное)

### Код программы

```
public class UserService : IUserService
{
    private readonly IUnitOfWork _unitOfWork;
    public UserService(IUnitOfWork unitOfWork)
    {
        _unitOfWork = unitOfWork;
    }
    public async Task<User> GetUser(string username)
    {
        return await _unitOfWork.UserRepository.FirstOrDefaultAsync(user => user.Name ==
            username);
    }
    public async Task<User> GetUserById(Guid id)
    {
        return await _unitOfWork.UserRepository.FirstOrDefaultAsync(u => u.Id == id);
    }
    public async Task<string> GetUserName(Guid id)
    {
        var user = await GetUserById(id);
        return user.Name;
    }
}

public class CategoryService : ICategoryService
{
    private readonly IUnitOfWork _unitOfWork;
    public CategoryService(IUnitOfWork unitOfWork)
    {
        _unitOfWork = unitOfWork;
    }
    public async Task<IEnumerable<Category>> GetAllAsync()
    {
        return await _unitOfWork.CategoryRepository.ListAllAsync();
    }
    public async Task<Category> GetCategoryAsync(Guid id)
    {
        return await _unitOfWork.CategoryRepository.FirstOrDefaultAsync(c => c.Id == id);
    }
    public async Task<(bool, Category)> CreateCategoryAsync(string name)
    {
        var category = new Category { Name = name };
        var categoryId = await _unitOfWork.CategoryRepository.AddAsync(category);
        await _unitOfWork.SaveAllAsync();
        return (categoryId != Guid.Empty, category);
    }
    public async Task<bool> DeleteCategoryAsync(Guid id)
    {
        var category = await _unitOfWork.CategoryRepository.FirstOrDefaultAsync(c => c.Id
            == id);
        if (category != null)
```

```

        {
            await _unitOfWork.CategoryRepository.DeleteAsync(category);
            await _unitOfWork.SaveAllAsync();
            return true;
        }
        return false;
    }
}

public async Task<Category> GetCategoryAsync(string name)
{
    string lowername = name.ToLower();
    return await _unitOfWork.CategoryRepository.FirstOrDefaultAsync(c => c.Name.
        ToLower() == lowername);
}

public async Task<string> GetCategoryName(Guid id)
{
    var cat = await GetCategoryAsync(id);
    return cat.Name;
}

public async Task<(bool, Category)> ChangeCategoryAsync(Guid id, Category category)
{
    var foundcategory = await _unitOfWork.CategoryRepository.FirstOrDefaultAsync(c =>
        c.Id == id);
    if (foundcategory != null)
    {
        foundcategory.Name = category.Name;
        await _unitOfWork.CategoryRepository.UpdateAsync(foundcategory);
        await _unitOfWork.SaveAllAsync();
        return (true, foundcategory);
    }
    return (false, null)!;
}
}

public class ItemService : IItemService
{
    private readonly IUnitOfWork _unitOfWork;
    public ItemService(IUnitOfWork unitOfWork)
    {
        _unitOfWork = unitOfWork;
    }

    public async Task<Item> GetItemAsync(Guid id)
    {
        return await _unitOfWork.ItemRepository.FirstOrDefaultAsync(i => i.Id == id);
    }

    public async Task<IEnumerable<Item>> GetItemsByCategoryAsync(string categoryName)
    {
        var lowcatname = categoryName.ToLower();
        var category = await _unitOfWork.CategoryRepository.FirstOrDefaultAsync(c => c.
            Name.ToLower() == lowcatname);
        if (category == null)
            return null;

        return await _unitOfWork.ItemRepository.ListAsync(i => i.CategoryId == category.Id
        );
    }
}

```

```

    }
    public async Task<IEnumerable<Item>> GetItemsByUserIdAsync(Guid userId)
    {
        return await _unitOfWork.ItemRepository.ListAsync(i => i.UserId == userId);
    }
    public async Task<IEnumerable<Item>> GetItemsByUserAsync(string username)
    {
        var user = await _unitOfWork.UserRepository.FirstOrDefaultAsync(u => u.Name ==
            username);
        if (user == null)
            return null;
        return await GetItemsByUserIdAsync(user.Id);
    }
    public async Task<(bool, Item)> ChangeItemStatusAsync(Guid itemId, ItemStatus status)
    {
        if(status > ItemStatus.Finished || status < ItemStatus.WaitingApprove)
            return (false, null);
        var item = await GetItemAsync(itemId);
        if (item == null)
            return (false, null);
        item.ItemStatus = status;
        await _unitOfWork.ItemRepository.UpdateAsync(item);
        await _unitOfWork.SaveAllAsync();
        return (true, item);
    }
    public async Task<(bool, Item)> ChangeItemDetailsAsync(Guid itemId, Item updatedItem,
        string username)
    {
        var item = await GetItemAsync(itemId);
        if (item == null)
            return (false, null);
        var user = await _unitOfWork.UserRepository.FirstOrDefaultAsync(u => u.Name ==
            username);
        if(user == null)
            return (false, null);
        if(item.UserId != user.Id && user.Role != UserRole.Admin && user.Role != UserRole.
            Staff)
            return (false, null);
        if (updatedItem.StartTime <= DateTime.UtcNow || updatedItem.EndTime <= DateTime.
            UtcNow)
            return (false, item);
        if (updatedItem.EndTime <= updatedItem.StartTime)
            return (false, item);
        if (updatedItem.StartingPrice < 0 || updatedItem.MinIncrease < 0)
            return (false, item);
        item.Title = updatedItem.Title;
        item.Description = updatedItem.Description;
        item.StartingPrice = updatedItem.StartingPrice;
        item.MinIncrease = updatedItem.MinIncrease;
        item.StartTime = updatedItem.StartTime;
        item.EndTime = updatedItem.EndTime;
        item.ItemStatus = ItemStatus.Updated;
        await _unitOfWork.ItemRepository.UpdateAsync(item);
    }

```

```

        await _unitOfWork.SaveAllAsync();
        return (true, item);
    }
    public async Task DeleteItemAsync(Guid itemId)
    {
        var item = await GetItemAsync(itemId);
        if (item != null)
        {
            await _unitOfWork.ItemRepository.DeleteAsync(item);
            await _unitOfWork.SaveAllAsync();
        }
    }
    public async Task<(bool, Item)> CloseItemAsync(Guid itemId, string username)
    {
        var item = await GetItemAsync(itemId);
        var user = await _unitOfWork.UserRepository.FirstOrDefaultAsync(u => u.Name ==
            username);
        if (item == null)
            return (false, null);
        if (user.Id != item.UserId && user.Role != UserRole.Admin && user.Role != UserRole.
            Staff)
            return (false, null);
        item.ItemStatus = ItemStatus.Finished;
        await _unitOfWork.ItemRepository.UpdateAsync(item);
        await _unitOfWork.SaveAllAsync();
        return (true, item);
    }
    public async Task<bool> DeleteItemAsync(Guid itemId, string username)
    {
        var item = await GetItemAsync(itemId);
        var user = await _unitOfWork.UserRepository.FirstOrDefaultAsync(u => u.Name ==
            username);
        if (item == null)
            return false;
        if (user.Id != item.UserId && user.Role != UserRole.Admin && user.Role != UserRole.
            Staff)
            return false;
        await _unitOfWork.ItemRepository.DeleteAsync(item);
        await _unitOfWork.SaveAllAsync();
        return true;
    }
}
public class BidService : IBidService
{
    private readonly IUnitOfWork _unitOfWork;
    public BidService(IUnitOfWork unitOfWork)
    {
        _unitOfWork = unitOfWork;
    }
    public async Task<IEnumerable<Bid>> GetBidsByItemAsync(Guid itemId)
    {
        return await _unitOfWork.BidRepository.ListAsync(b => b.ItemId == itemId);
    }
}

```

```

public async Task<IEnumerable<Bid>> GetBidsByUserAsync(string username)
{
    var user = await _unitOfWork.UserRepository.FirstOrDefaultAsync(u => u.Name ==
        username);
    if (user == null)
        return Enumerable.Empty<Bid>();
    return await _unitOfWork.BidRepository.ListAsync(b => b.UserId == user.Id);
}

public async Task<(bool, Bid)> CreateBidAsync(decimal amount, string username, Guid
    itemId)
{
    var user = await _unitOfWork.UserRepository.FirstOrDefaultAsync(u => u.Name ==
        username);
    var item = await _unitOfWork.ItemRepository.FirstOrDefaultAsync(i => i.Id ==
        itemId);
    if (user == null || item == null || amount <= 0 || item.StartingPrice > amount)
        return (false, null);
    if (item.StartTime > DateTime.UtcNow || item.EndTime < DateTime.UtcNow)
        return (false, null);
    if (item.ItemStatus != ItemStatus.Active)
        return (false, null);
    var lastBid = await _unitOfWork.BidRepository
        .ListAsync(b => b.ItemId == itemId)
        .ContinueWith(task => task.Result.OrderByDescending(b => b.DateTime).
            FirstOrDefault());
    if (lastBid != null &&
        (lastBid.Amount >= amount
            || lastBid.Amount + item.MinIncrease > amount))
    {
        return (false, null);
    }
    var bid = new Bid
    {
        Amount = amount,
        UserId = user.Id,
        ItemId = item.Id,
        DateTime = DateTime.UtcNow
    };
    var bidId = await _unitOfWork.BidRepository.AddAsync(bid);
    await _unitOfWork.SaveAllAsync();
    if (bidId != Guid.Empty)
    {
        return (true, bid);
    }
    return (false, null);
}
}

```

## ПРИЛОЖЕНИЕ Б (обязательное)

### Алгоритмы, используемые в программном средстве

Схемы алгоритмов указаны на рисунках Б.1, Б.2, Б.3.

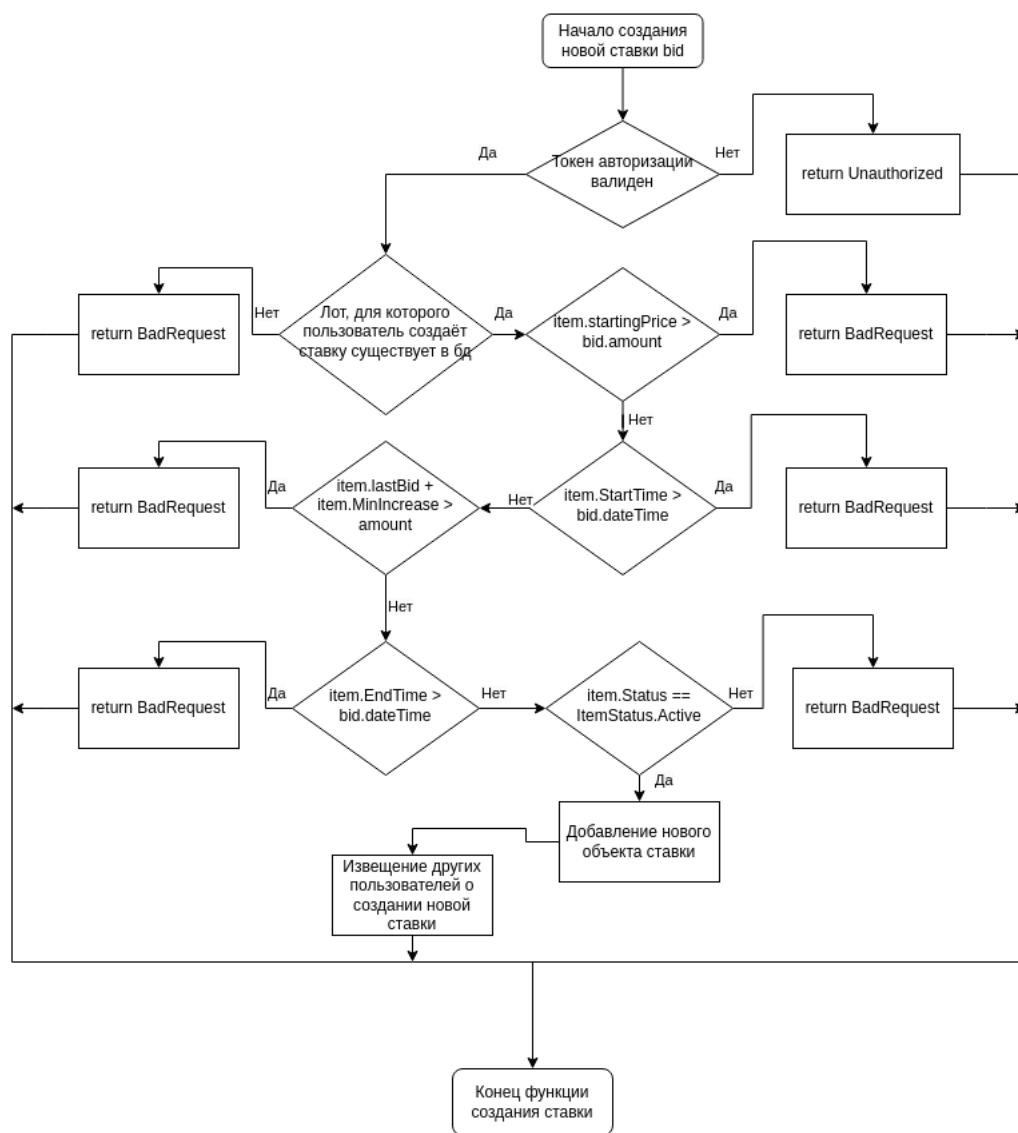


Рисунок Б.1 – Схема алгоритма создания новой ставки



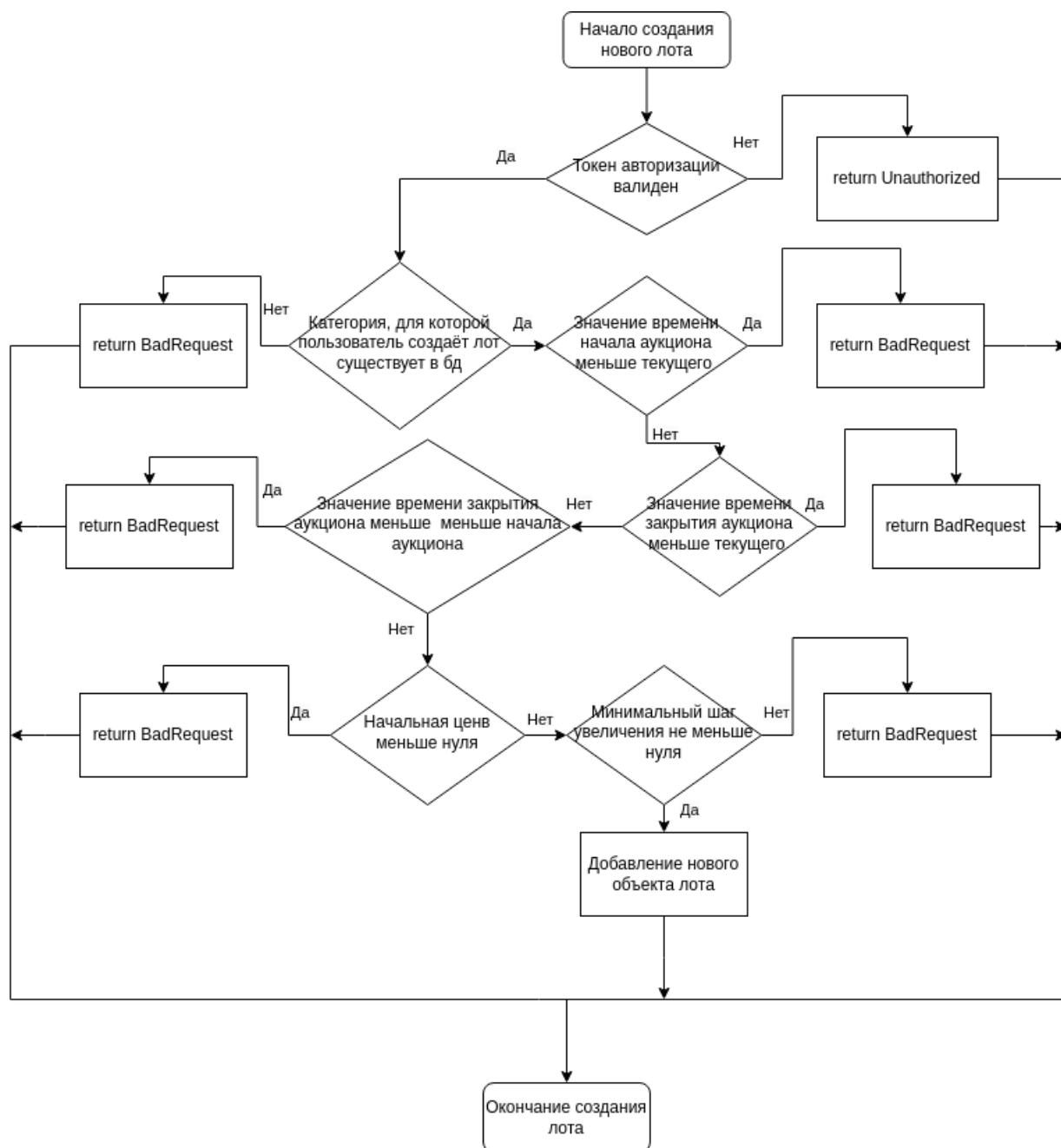


Рисунок Б.2 – Схема алгоритма создания нового лота

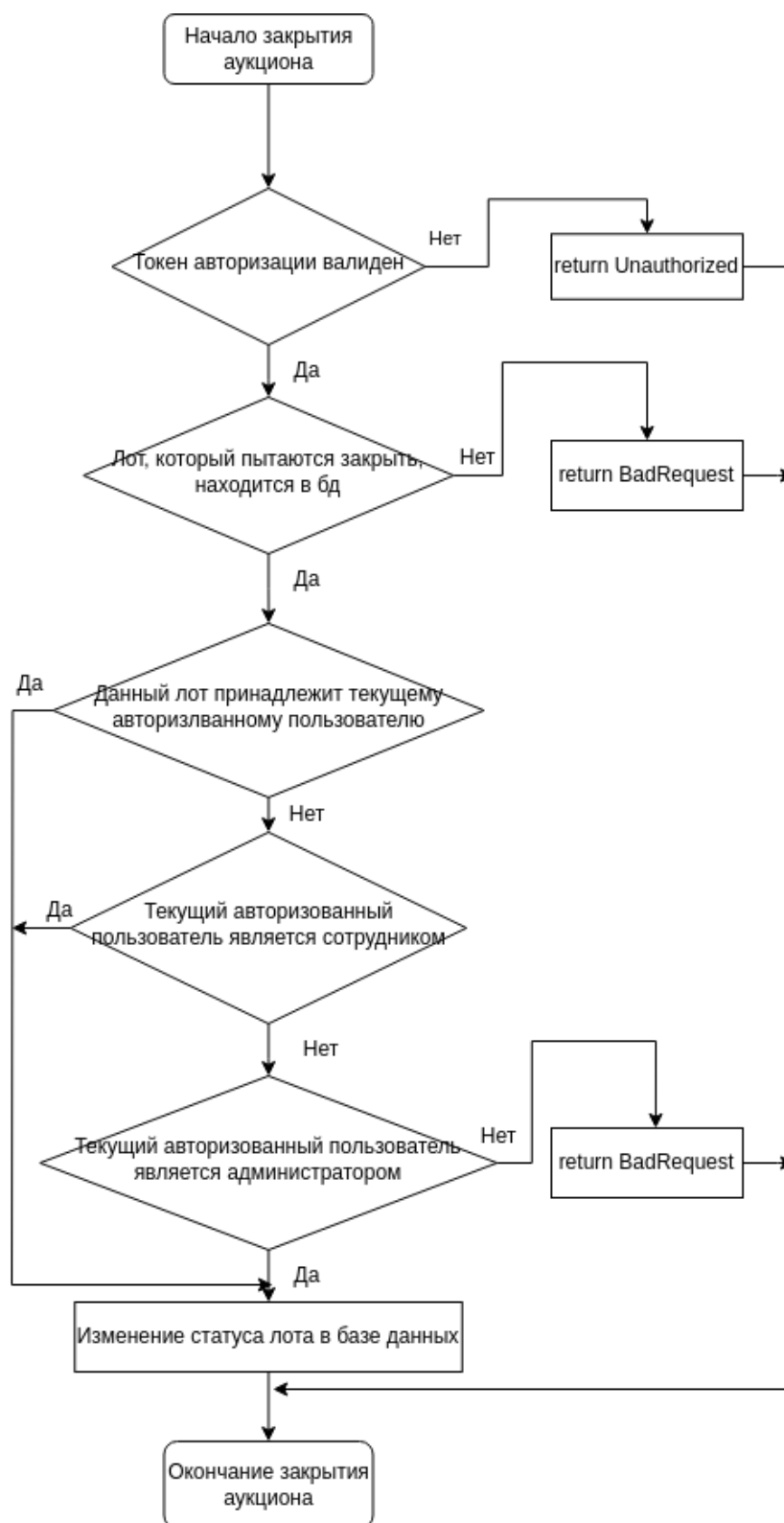


Рисунок Б.3 – Схема алгоритма завершения аукциона