МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных

технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

WEB-приложение «Торговая площадка»

Выполнил студент Борисов Антон Андреевич

(Ф.И.О.)

Руководитель проекта преп.-стаж. Сенюк В.К.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты преп.-стаж. Сенюк В.К.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер преп.-стаж. Сенюк В.К.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2021

# **Введение**

В данном курсовом проекте разработано WEB-приложение «Торговая площадка». Логически оно разделено на две части: серверную, написанную на NestJS, и клиентскую написанную на ReactJS. Серверная часть взаимодействует с базой данных и размещена на платформе «Heroku». Клиентская часть приложения взаимодействует с сервером через REST API и размещена на платформе «Netlify».

В современном мире информационные технологии позволяют автоматизировать и упростить работу во многих сферах деятельности человека. Базы данных, интернет, быстрый поиск, автоматическое вычисление данных позволяют упростить продажу вещей в интернете. Это и является целью моего курсового проекта ­– создание простого и удобного программного средства, позволяющего пользователям ознакомиться с ассортиментом торговой площадки, выставить товар на продажу, выбрать товар, положить его в корзину и заказать. Также продавцы на сайте получают доступ к заказам их продуктов, а значит есть возможность организовать доставку товаров.

Я выбрал в качестве темы курсового проекта «Торговая площадка» в связи с активным использованием онлайн площадок для приобретения и продажи товаров. Данное приложение содержит тот функционал, который поможет вам осуществлять покупки, не выходя из дома, тем самым экономить ваше время. А также продавать вещи, которыми вы уже не пользуетесь.

Основная цель курсового проекта: разработка web-приложения «Торговая площадка».

В первом разделе рассматриваются основные технологии, которые использовались в разработке данного приложения, а также его прототипы и актуальность задачи.

Во втором разделе описана архитектура курсового проекта.

В третьем разделе предоставлена информация о разработанных объектах базы данных.

В четвертом разделе представлены результаты тестирования приложения.

Пятый раздел содержит руководство пользователя для разработанного клиентского приложения.

В заключении описывается результат курсового проектирования и задачи, которые были решены в ходе разработки приложения.

1. **Постановка задачи**

Главная задача курсового проекта является разработка WEB- приложения, позволяющего пользователю просматривать каталог товаров, выставлять товары на торговую площадку, добавлять товары в корзину, а также удалять их из корзины, оформлять заказ, выставлять товары на аукцион, ставить ставки на определенный лот. В данном курсовом проекте требовалось реализовать следующие задачи:

* сохранение рабочей информации в централизованной базе данных;
* организация регистрации и входа в систему со стороны пользователей;
* возможность просмотра, добавления, удаления товарных позиций;
* осуществление заказа;
* возможность просмотра, добавления, удаления товаров на аукцион;
* возможность поставить ставку на аукцион;

## **Алгоритмы решения**

При реализации курсового проекта использовались технологии Node.js, фреймворк NestJs, ORM библиотека Sequelize-typescript, Socket.IO для реализации back-end. Для front-end части Material-UI, JavaScript, ReactJS, Redux, Redux-Saga, Sokect.IO-client. PostgreSQL в качестве базы данных.

Node.js –  программная платформа, основанная на движке [V8](https://ru.wikipedia.org/wiki/V8_(%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA_JavaScript)) (транслирующем [JavaScript](https://ru.wikipedia.org/wiki/JavaScript" \o "JavaScript) в [машинный код](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность [JavaScript](https://ru.wikipedia.org/wiki/JavaScript" \o "JavaScript) взаимодействовать с устройствами [ввода-вывода](https://ru.wikipedia.org/wiki/%D0%92%D0%B2%D0%BE%D0%B4-%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4) через свой [API](https://ru.wikipedia.org/wiki/API) (написанный на [C++](https://ru.wikipedia.org/wiki/C%2B%2B)), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль [веб-сервера](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80), но есть возможность разрабатывать на Node.js и десктопные оконные приложения и даже программировать микроконтроллеры. В основе Node.js лежит [событийно-ориентированное](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B9%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [асинхронное (или реактивное)](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) программирование с [неблокирующим вводом/выводом](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B1%D0%BB%D0%BE%D0%BA%D0%B8%D1%80%D1%83%D1%8E%D1%89%D0%B0%D1%8F_%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F).

NestJS - это платформа для создания эффективных масштабируемых серверных приложений Node.js. Он использует прогрессивный JavaScript, построен на TypeScript и полностью поддерживает его (но при этом позволяет разработчикам кодировать на чистом JavaScript) и сочетает в себе элементы ООП (объектно-ориентированное программирование), FP (функциональное программирование) и FRP (функциональное реактивное программирование).

Под капотом Nest использует надежные фреймворки HTTP-серверов, такие как Express (по умолчанию), и при желании также может быть настроен на использование Fastify!

Sequelize – это ORM-библиотека для приложений на Node.js, которая осуществляет сопоставление таблиц в базе данных и отношений между ними с классами. При использовании Sequelize мы можем не писать SQL-запросы, а работать с данными как с обычными объектами. Для работы с Typescript необходимо установить пакет sequelize-typescript, который добавляет возможность объявлять классы моделй с помощью декораторов

WebSoket – это передовая технология, которая позволяет создавать интерактивное соединение между клиентом (браузером) и сервером для обмена сообщениями в режиме реального времени. Веб-сокеты, в отличие от HTTP, позволяют работать с двунаправленным потоком данных, что делает эту технологию совершенно уникальной. Socket.IO - это библиотека, которая упращает работу с веб сокетами на серверной стороне. Socket.IO-client на клиентской.

JavaScript – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией стандарта ECMAScript (стандарт ECMA-262). JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Material-UI – это библиотека, которая включает компоненты React, реализующие Material Design от Google.

React — это декларативная, эффективная и гибкая JavaScript библиотека для создания пользовательских интерфейсов. Она позволяет собирать сложный UI из маленьких изолированных кусочков кода, называемых «компонентами».

Redux является предсказуемым контейнером состояния для JavaScript приложений. Это позволяет создавать приложения, которые ведут себя одинаково в различных окружениях (клиент, сервер и нативные приложения), а также просто тестируются.

Redux-Saga — это библиотека, которая призвана упростить и улучшить побочные эффекты асинхронных запросов (т.е. такие действия, как асинхронные операции, например, загрузки данных, и "грязные" действия, такие, как доступ к браузерному кешу), сделать лёгкими в тестировании и лучше справляться с ошибками.

## **Обзор прототипа**

В наше время в сети Интернет можно найти любую информацию, поэтому было решено поискать аналоги проектируемого приложения в иных ведущих торговых площадок страны.

Одной из них является площадка Куфар. Эта площадка предназначена для размещения объявлений, покупок интересующих товаров и доставки. Достаточно удобная система поиска товаров и размещения объявления. Приложение имеет довольно понятный любому пользователю интерфейс и функционал.

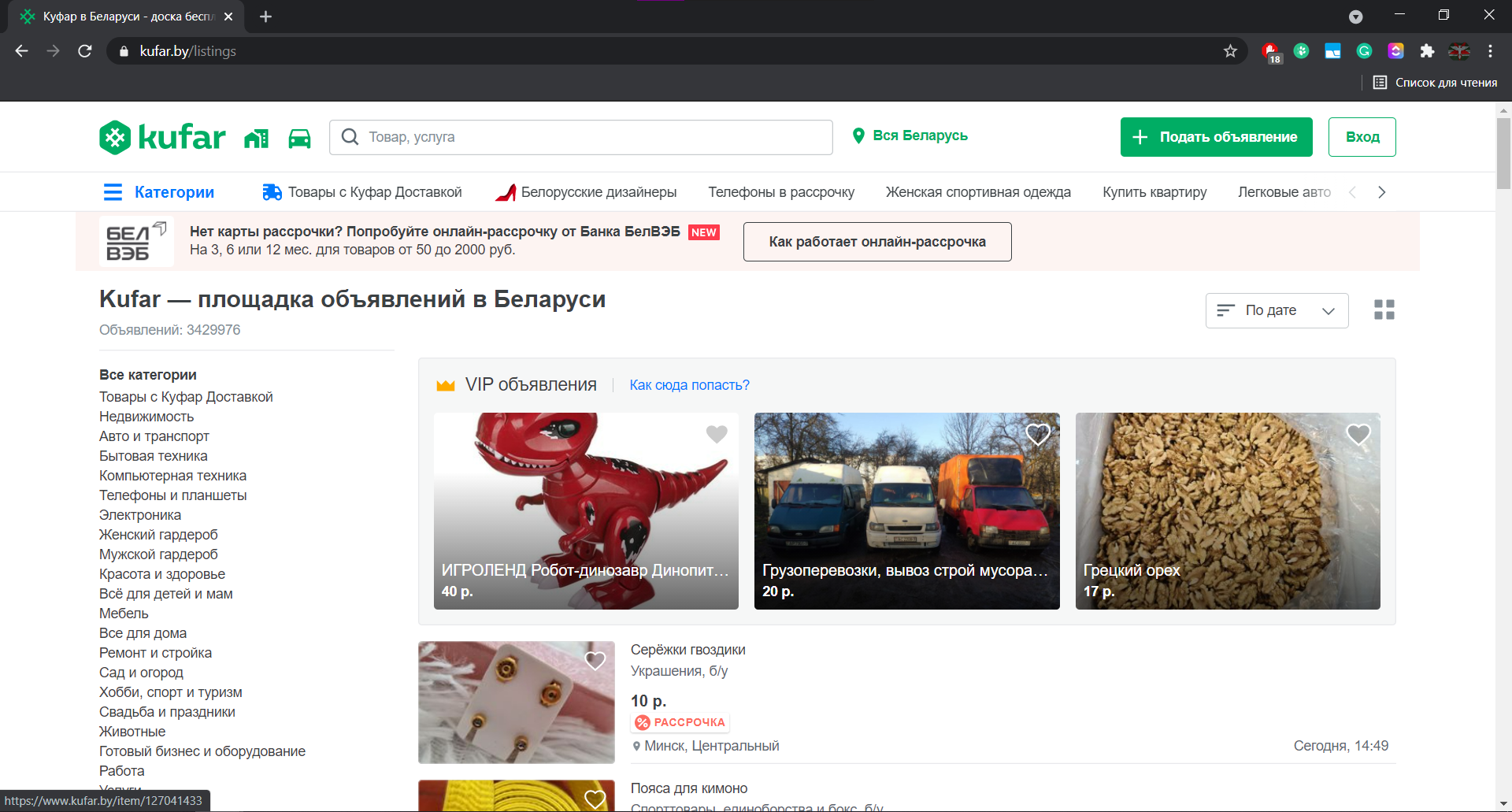


Рисунок 1.1 – Торговая площадка «Куфар»

Еще один интернет-магазин помог мне определиться с функциональностью моего WEB-приложения – это магазин «Wildberries» (рисунок 1.2).

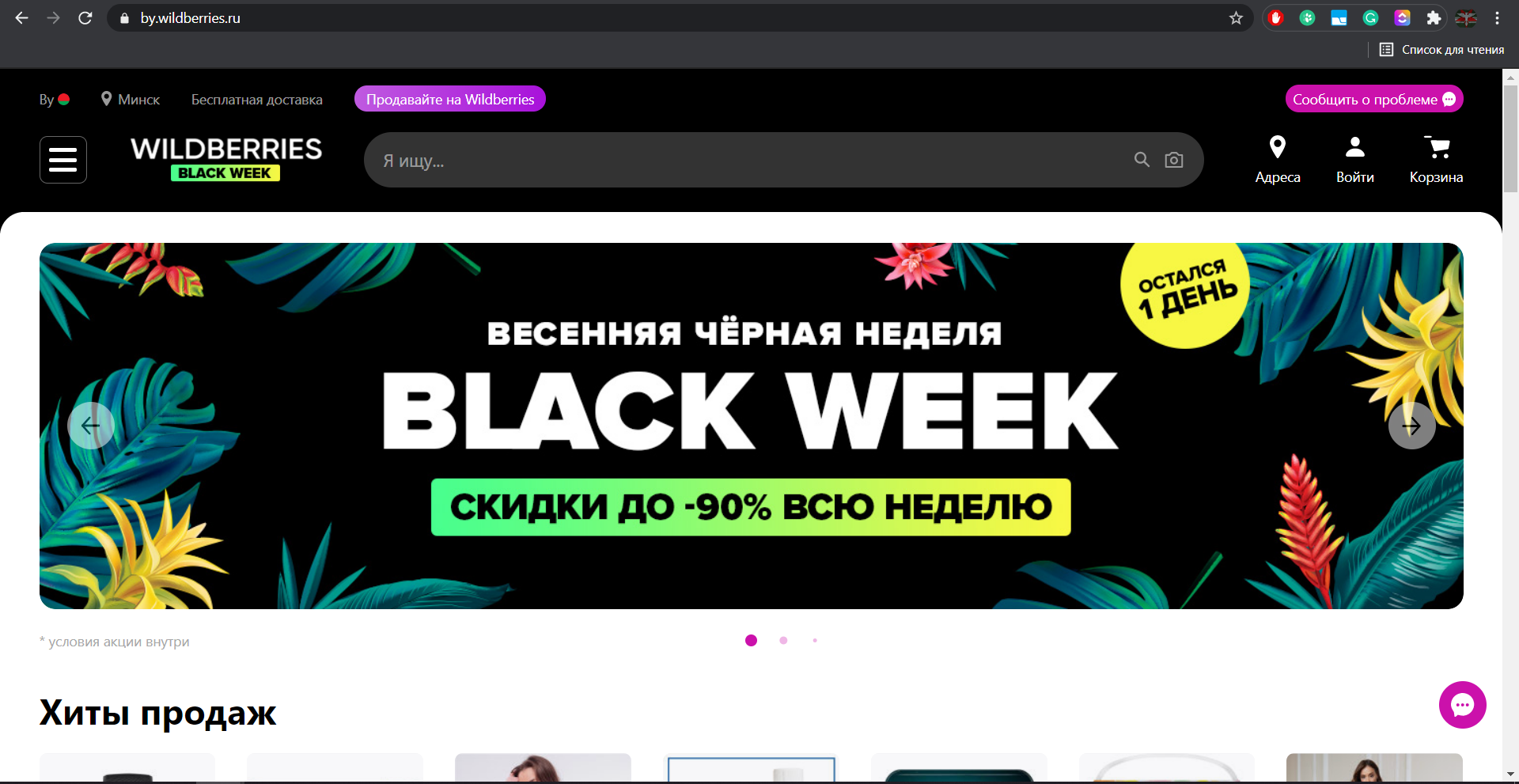


Рисунок 1.2 – Интернет-магазин «Wildberries»

# **Проектирование приложения**

Разработка архитектуры проекта – одна из важнейших задач в процессе работы над приложением, так как в зависимости от неё определяется уровень зависимости компонентами приложения, и насколько легко расширяемы её составные части.

Архитектура проекта – это его строение как оно видно (или должно быть видно) из вне его, т.е. представление программного средства как системы, состоящей из некоторой совокупности взаимодействующих подсистем. В качестве таких подсистем выступают обычно отдельные программы. Разработка архитектуры является первым этапом борьбы со сложностью программного средства, на котором реализуется принцип выделения относительно независимых компонент.

Основные задачи разработки архитектуры проекта:

Выделение программных подсистем и отображение на них внешних функций (заданных по внешнем описании) программного средства;

Определение способов взаимодействия между выделенными программными подсистемами.

С учетом принимаемых на этом этапе решений производится дальнейшая конкретизация и функциональных спецификаций.

## **Архитектура**

Для разработки подобного типа проекта было решено использовать стандартную трёхуровневую архитектуру, так как такая архитектура позволяет добиться низкой связности уровней, что, в свою очередь, обеспечивает для приложения надёжность, простоту долгосрочной поддержки, а также оперативное устранение неисправностей при необходимости.

Клиентская часть приложения является Single-Page Application, то есть приложение, использует единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемый HTML, CSS, JavaScript код, генерируемый на основе ответов, поступивших на AJAX запросы. Также на клиенте реализовывается логика обработки полученных от сервера сообщений об ставках на определенный лот на аукционе.

Сервер будет выполнять мероприятия по поддержке аутентификации и авторизации, а также будет ответственным за всё взаимодействие с базой данных.

## **Серверная часть приложения**

При проектировании классов и интерфейсов для серверной части приложения необходимо стараться держать низкий уровень избыточности и связности между составляющими частями, а также усиленно следить за корректностью кода, поскольку от этого будет зависеть работа всех подключенных клиентов. Добиться всего этого можно с помощью грамотного распределения логики сервера по разным уровням, определяя формат объектов, которые будут использоваться на каждом уровне, и преобразования объектов при передаче с уровня на уровень, а также используя подходящие паттерны проектирования.

Для того, чтобы обмениваться данными между клиентами и сервисами был выбран архитектурный стиль REST. По своему определению он широко использует возможности протокола HTTP и поэтому не имеет состояния (stateless). Это современный подход, использующийся в большинстве веб-сервисов и поэтому имеющий множество примеров реализаций различных функций на его основе, таких как аутентификация, выполнение CRUD-операций и т.д. В REST-приложениях в подавляющем большинстве случаев используется JSON для передачи данных, его также было решено взять как формат данных при передаче между клиентами и сервером.

## **Уровни. API, Service, Repository**

Приняв во внимание вышенаписанное, лучше всего разбить серверную логику на следующие уровни API, Service и Repository.

API-уровень. Это самый верхний уровень сервера, и на нём будут располагаться классы-контроллеры, методы которых будут описывать URL-адреса, на которые можно будет сделать вызов, и их характеристику. Также эти методы будут преобразовывать данные из формата, пригодного для сериализации, в формат, пригодный для сохранения данных в БД и работы с ними как с бизнес-объектами.

Service-уровень. На этом уровне находятся классы, ответственные за бизнес-логику, среди них как те, которые просто делегируют вызов к репозиторию, так и те, которые выполняют более сложную логику.

Repository-уровень. Он содержит в себе классы-репозитории, использующиеся, когда необходимо совершить какие-либо операции с базой данных. Методы этих классов могут обращаться к БД как с помощью автоматически сгенерированных SQL-запросов, так и с помощью написанных непосредственно программистом. Стоит упомянуть, что создание данных классов в NestJS не требуется, так как для нас эти классы предоставляет пакет sequelize.

## **Взаимосвязь всех компонентов**

В решении курсового проекта была использована REST архитектура. Рассмотрим компоненты, используемые в данной архитектуре.

Клиент – это программа, использующая API. Клиент делает запросы к API, чтобы получить некоторую информацию или что-то изменить в приложении. Веб-браузер является клиентом - он взаимодействует с API-интерфейсом, чтобы получить от него содержимое страницы. Запрошенная информация отправляется обратно в браузер и отображается на экране.

Ресурс - это любая информация, которую API может предоставить клиенту. Например, ресурсом в данном API может быть пользователь, фотография или товар. Каждый ресурс имеет уникальное имя, называемое идентификатором ресурса.

Сервер используется приложением, которое принимает запросы клиентов и содержит ресурсы, которые требуются клиенту. Сервер имеет API для взаимодействия с клиентами, не предоставляя им прямого доступа к контенту, хранящемуся в базе данных.

Для реализации серверной части была использована N-Layer архитектура. Эта архитектура выбрана, так как трехуровневая архитектура обеспечивает множество преимуществ для производственной среды и среды разработки за счет модульного разделения пользовательского интерфейса, бизнес-логики и уровней хранения данных. Это дает большую гибкость, позволяя обновлять определенную часть приложения независимо от других частей. Эта дополнительная гибкость может улучшить общее время вывода продукта на рынок и сократить время цикла разработки, давая возможность заменять или обновлять независимые уровни, не затрагивая другие части системы. Данный курсовой проект построен по примеру ниже приведенной схемы классической трехуровневой системы. Ее схематическое представление продемонстрировано на рисунке 2.1.

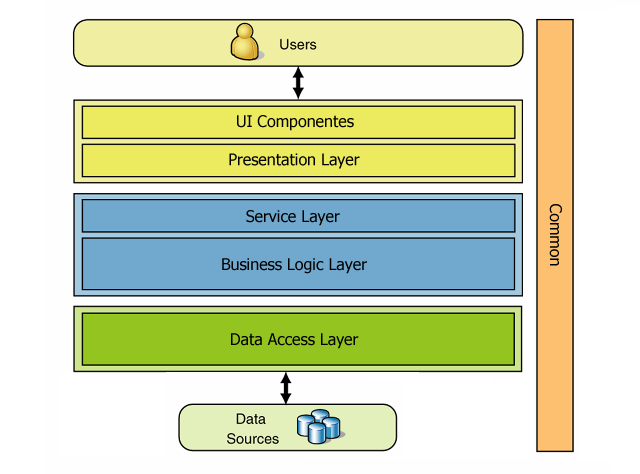


Рисунок 2.1 – Общая схема N-Layer архитектуры

Data Access Layer (уровень доступа к данным) содержит сущности, соответствующие таблицам базы данных, а также схемы взаимодействия друг с другом.

Business Logic Layer (уровень бизнес-логики) при тесном взаимодействии с Service Layer (уровень сервисов) являются посредниками между сущностями базы данных и моделями для UI, сочетая в себе сервисы и классы, реализующий основной функционал всего разрабатываемого приложения.

Presentation Layer (уровень представления), поддерживаемый UI Components (User Interface Components – компоненты пользовательского интерфейса), представляет собой генерирующуюся на сервере разметку или получаемые с уровня бизнес-логики данные для REST API в формате JSON, преобразующиеся впоследствии в разметку на стороне клиента. Уровень представления состоит состоящее из контроллеров, координирующих пользовательские запросы, и представлений, формирующееся в разметку HTML**.**

## **Диаграмма UML**

ДиаграммаUML–это графическое представление набора элементов,изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями).

В языке UML вариант использования изображается в виде овала, помеченного именем представляемого варианта. Варианты использования могут быть связаны с участвующими в них действующими лицами (actors), изображаемыми в виде человечков и представляющими различные роли пользователей системы или внешние системы, взаимодействующие с ней.

Варианты использования могут быть связаны друг с другом тремя видами связей: обобщением (generalization), расширением (extend relationship) и включением (include relationship). Действующие лица также могут быть связаны друг с другом с помощью связей обобщения (generalization).

Для каждой роли были разработаны UML-диаграммы вариантов использования. При первом открытии сайта пользователь не аутентифицирован. Диаграмма вариантов использование неаутентифицированного пользователя отображена на рисунке 2.2.



Рисунок 2.2 – Диаграмма использования неаутентифицированный пользователя

Как видно из диаграммы, неаутентифицированный пользователь может зарегистрироваться, авторизироваться, просмотреть каталог товаров, а также лоты на аукционе, производить поиск по продуктам, а также добавлять и удалять товары в корзине. При попытке совершения заказа пользователю будет выведено сообщение о том что ему необходимо авторизироваться.

Итак, зарегистрировавшись и войдя в аккаунт, пользователь получает роль аутентифицированного пользователя. Диаграмма вариантов использования приложения в зависимости от роли отображена на Рисунке 2.3.

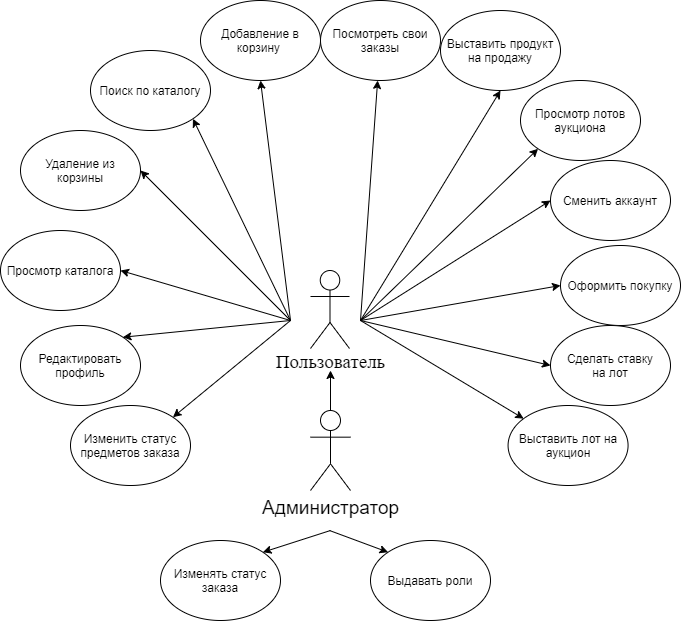


Рисунок 2.3 – Диаграмма использования аутентифицированного пользователя

Из диаграммы видно, что администратор имеет тот же функционал, как и обычный пользователь за исключением того, что присутствуют дополнительные доступные действия для подтверждения статуса заказов и выдачи роли.

## **Сущности в базе данных и связи между ними**

Изучение предметной области и составленный впоследствии функционал показали, что можно выделить 8 таблиц, которые позволяют полностью охватить обозначенный функционал (рис. 2.1). Из них 7 основных:

* пользователь (users) – таблица, предназначенная для хранения основных данных о пользователе, таких как имя, телефон, электронная почта, пароль;
* продукт (products) – таблица, предназначенная для хранения данных о продукте;
* аукцион (auctions) – таблица, предназначенная для хранения данных о товарах выставленных на аукцион;
* аукцион (bids) – таблица, предназначенная для хранения ставок сделанных на аукционе;
* заказы (orders) – таблица, предназначенная для хранения информации о заказах пользователя;
* товары заказа(order\_items) – таблица, предназначенная для хранения информации о товарах в заказе;
* роль (roles) – таблица нужна для того, чтобы хранить данные о существующих ролях;

Оставшиеся таблица user\_roles вспомогательная, и нужна для того, чтобы поддерживать отношения типа «многие-ко-многим».

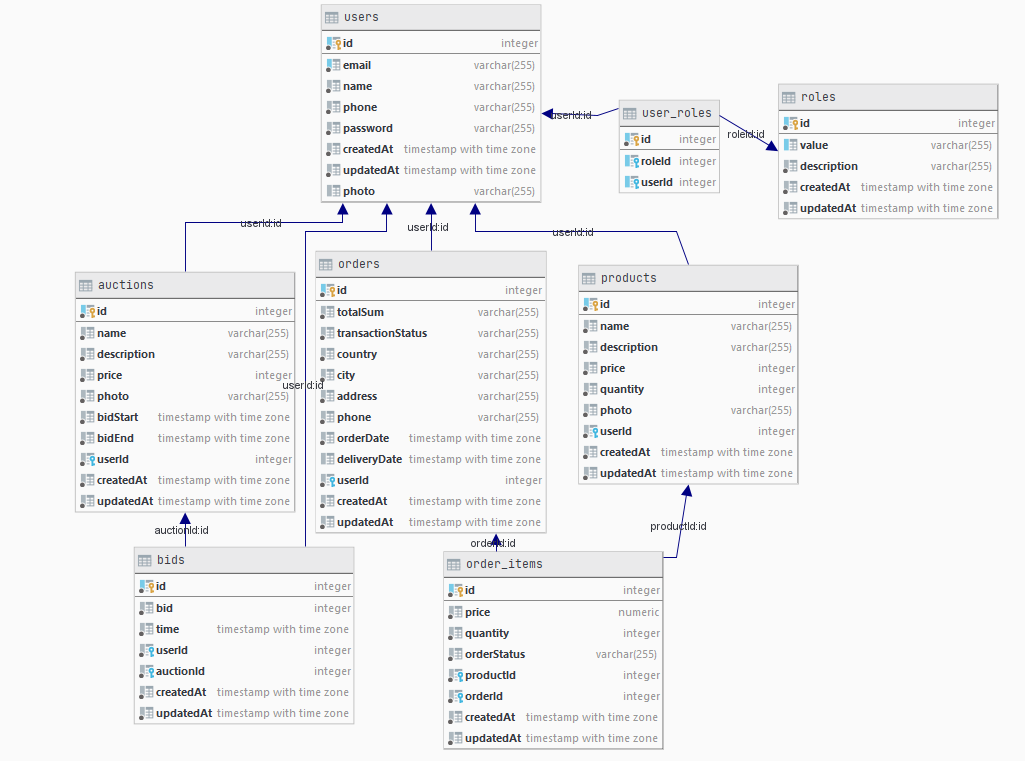


Рисунок 2.1 – Схема базы данных

Отдельно стоит упомянуть способ хранения картинок в базе данных. Там их хранение осуществляется в виде относительных ссылок на файловую систему внутри сервера, либо абсолютных ссылок на Интернет-ресурс, откуда её будет необходимо качать для просмотра.

# **Разработка приложения**

Для разработки и управления базой данных курсового проекта использовалась