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 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Web-приложение «Интернет-магазин настольных игр»

Выполнил студент Романчук Алексей Михайлович

(Ф.И.О.)

Руководитель проекта ст.преп. Дубовик М.В.

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

И.о. заведующего кафедрой ст.преп. Блинова Е.А.

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

Курсовой проект защищен с оценкой

Минск 2024

**Содержание**

[Введение 5](#_Toc166244641)

[1 Постановка задачи 6](#_Toc166244642)

[1.1 Обзор аналогичных решений 6](#_Toc166244643)

[1.2 Спецификация требований 8](#_Toc166244644)

[2 Проектирование web-приложения 9](#_Toc166244645)

[2.1 Проектирование вариантов использования 9](#_Toc166244646)

[2.2 Структура web-приложения 10](#_Toc166244647)

[2.3 Проектирование базы данных 11](#_Toc166244648)

[2.4 Проектирование сервера web-приложения 14](#_Toc166244649)

[3 Разработка web-приложения 15](#_Toc166244650)

[3.1 Разработка серверной части приложения 15](#_Toc166244651)

[3.2 Разработка клиентской части приложения 20](#_Toc166244652)

[4 Тестирование web-приложения 22](#_Toc166244653)

[5 Руководство пользователя 28](#_Toc166244654)

[Заключение 34](#_Toc166244655)

[Список используемых источников 35](#_Toc166244656)

[Приложение А 36](#_Toc166244657)

[Приложение Б 37](#_Toc166244658)

[Приложение В 40](#_Toc166244659)

# **Введение**

В современном мире, где технологии развиваются с невероятной скоростью, web-приложения играют ключевую роль во многих сферах жизни, включая торговлю и розничную продажу. Это касается и магазинов настольных игр, которые стремятся привлечь большее количество клиентов, улучшить их опыт покупки и повысить эффективность своей работы. Одним из основных преимуществ web-приложений является возможность предоставления удобного и доступного интерфейса для покупателей, что позволяет им легко ознакомиться с ассортиментом товаров, сравнить различные варианты и совершить покупку в любое время.

Web-приложения используются в том числе для продажи настольных игр. Он не только представляют собой источник развлечения и отдыха, но и способствуют укреплению социальных связей, развитию коммуникативных навыков и обучению стратегическому мышлению. В эпоху цифровизации настольные игры остаются важным элементом культуры, предлагая альтернативу электронным развлечениям и способствую общению в реальном времени. Таким образом, сочетание актуальности настольных игр и возможности их продвижения через web-приложения даёт возможность магазинам настольных игр расширить свою аудиторию, улучшить взаимодействие с клиентами и увеличить продажи.

Целью данного курсового проекта является разработка распределённого web-приложения для магазина настольных игр. Такое приложение должно обеспечивать гибкость и масштабируемость, позволяя не только обрабатывать запросы от пользователей, но и обеспечивать надёжное хранение данных и конфиденциальность информации пользователей.

Для достижения цели курсового проекта необходимо выполнить следующие задачи:

* проанализировать существующие решения, определить их преимущества и недостатки;
* изучить требования к разрабатываемому приложению, определить варианты использования;
* выполнить проектирование архитектуры web-приложения;
* выполнить проектирование структуры базы данных;
* реализовать web-приложение;
* выполнить тестирование разработанного приложения;
* разработать руководство пользователя.

Распределённое web-приложение для магазина настольных игр, разработанное в рамках этого курсового проекта, будет способствовать улучшению качества обслуживания клиентов, увеличению продаж и оптимизации работы магазина.

# **1 Постановка задачи**

## **Обзор аналогичных решений**

В связи с расширением возможностей и удобства использования интернета, всё больше людей предпочитают заказывать настольные игры в онлайн-магазинах, используя их web-сайты, мобильные и десктопные приложения. Для составления подробной спецификации требований к разрабатываемому приложению далее рассматриваются web-сайты интернет-магазинов настольных игр Hobby Games (hobbygames.by) и Nastolki.by (nastolki.by).

Вид страницы сайта hobbygames.by при выборе настольной игры из каталога представлен на рисунке 1.1.

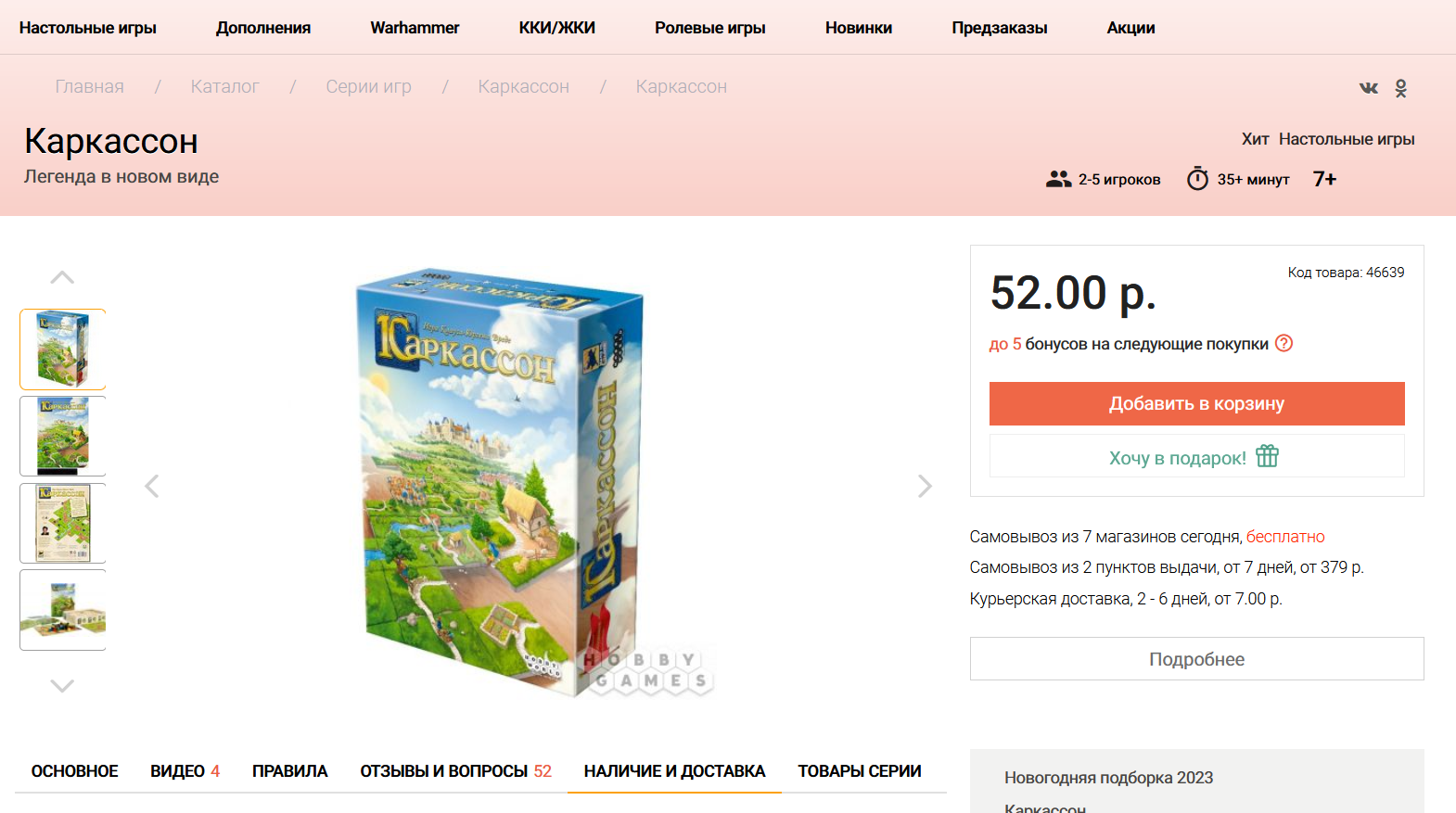


Рисунок 1.1 – Страница сайта hobbygames.by с информацией о настольной игре

Сайт позволяет просматривать каталог товаров, выполнять по нему поиск, фильтровать его по категориям, диапазону цен, количеству игроков, производителю, а также предлагаются различные варианты сортировки, такие как по возрастанию или убыванию цены, названию, популярности и другие.

Для настольных игр указана следующая информация: название, описание, изображения, цена, средняя продолжительность игры, возрастное ограничение, категория, правила, видео-обзоры, отзывы, производитель, год выпуска и другое.

Для заказа необходимо сначала добавить товары в корзину. Заказ могут оформлять как авторизованные, так и неавторизованные пользователи. Для авторизованных пользователей есть система бонусов и заданий.

Полнота предоставляемой информации и широкие возможности для неавторизованных пользователей являются большими преимуществами данного сайта. Недостатком же следует указать отсутствие возможности добавлять товары в коллекцию избранного за неимением таковой.

Сайт nastolki.by позволяет просматривать каталог настольных игр, фильтровать по ключевым фразам, цене, количеству игроков, категориям товаров и по производителю. Также на сайте есть возможность сортировки по цене и названию. Большое количество категорий товаров упрощает их поиск для потенциальных клиентов. Список категорий показан на рисунке 1.2.

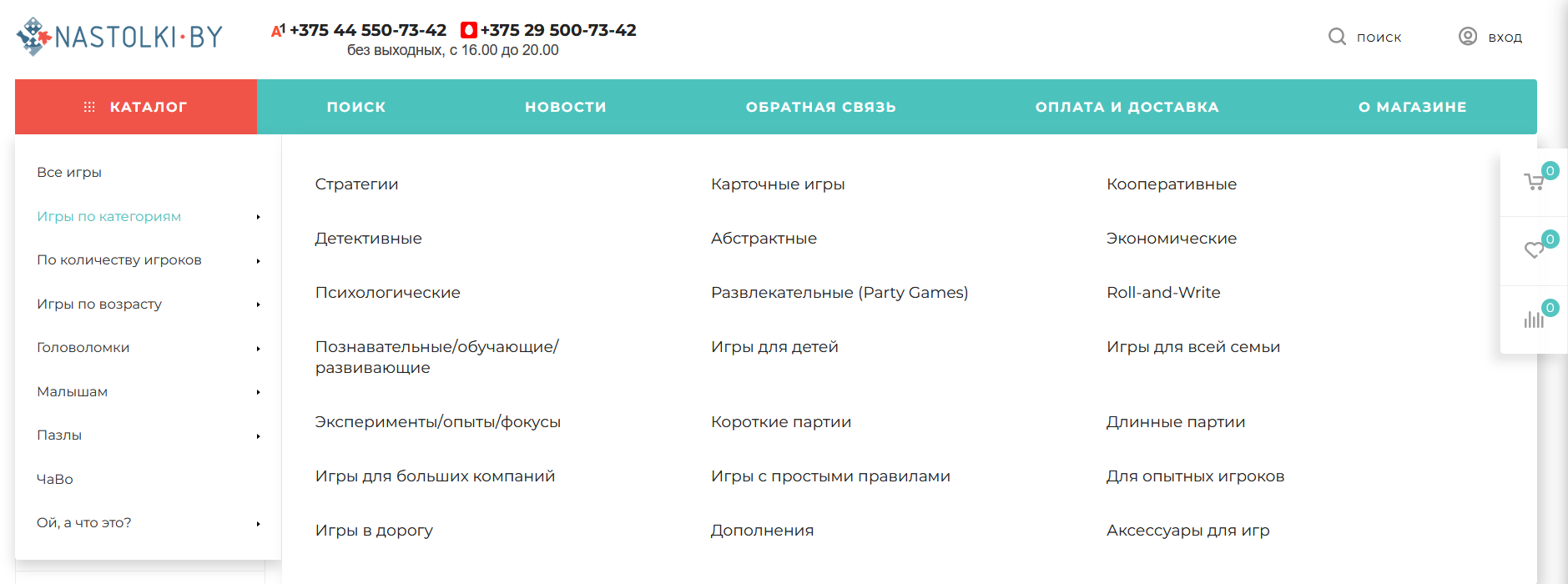


Рисунок 1.2 – Категории товаров на сайте nastolki.by.

Для товаров указана цена, общая информация, включающая количество игроков, время партии, возрастное ограничение. Также для товаров указывается скидка. Вид сайта nastolki.by при переходе к конкретному товару представлен на рисунке 1.3. Кроме того, сайт позволяет управлять коллекцией избранного, корзиной, даёт возможность оформить заказ. Недостатком является нестабильная работа клиентской части сайта и небольшой набор предоставляемой информации.



Рисунок 1.3 – Вид страницы сайта nastolki.by с информацией о товаре

Таким образом, оба сайта предоставляют широкие возможности манипуляции каталогом, управления корзиной, позволяют оформить заказ и изменить личную информацию пользователя. Подробная информация о товарах и широкие функциональные возможности могут выделить разрабатываемое приложение.

## **Спецификация требований**

После рассмотрения достоинств и недостатков аналогичных решений были определены требования к разрабатываемому приложению, в соответствии с которыми должны обеспечиваться следующие основные возможности для клиентов:

* просматривать товары в каталоге, фильтровать их и сортировать;
* добавлять и исключать товары из коллекции избранного;
* добавлять, исключать, изменять количество товаров в корзине;
* оформлять заказы товаров;
* просматривать историю заказов с информацией об их текущем статусе.

Администраторы должны иметь возможность управления каталогом товаров, включая:

* добавление новых товаров в каталог с их подробными характеристиками и изображениями;
* изменение параметров существующих товаров;
* регулирование доступности товаров для заказа и их количества на складе.

Также администраторам должно быть доступно изменение статусов заказов пользователей, добавление новых администраторов. Исходя из принципа разделения ролей у администраторов не должно быть собственной корзины товаров, коллекции избранного, а также они не должны иметь возможность оформлять свои заказы.

Для гостей должна быть обеспечена возможность использования каталога товаров без обязательной регистрации.

Приложение должно поддерживать механизм авторизации пользователей. Все соединения между клиентом и сервером должны быть защищены. Для обеспечения отзывчивости приложения должны использоваться асинхронные запросы.

2 Проектирование web-приложения

2.1 Проектирование вариантов использования

В приложении должны поддерживаться роли гостя, клиента и администратора. Для описания их полномочий в приложении была спроектирована диаграмма вариантов использования, представленная на рисунке 2.1.

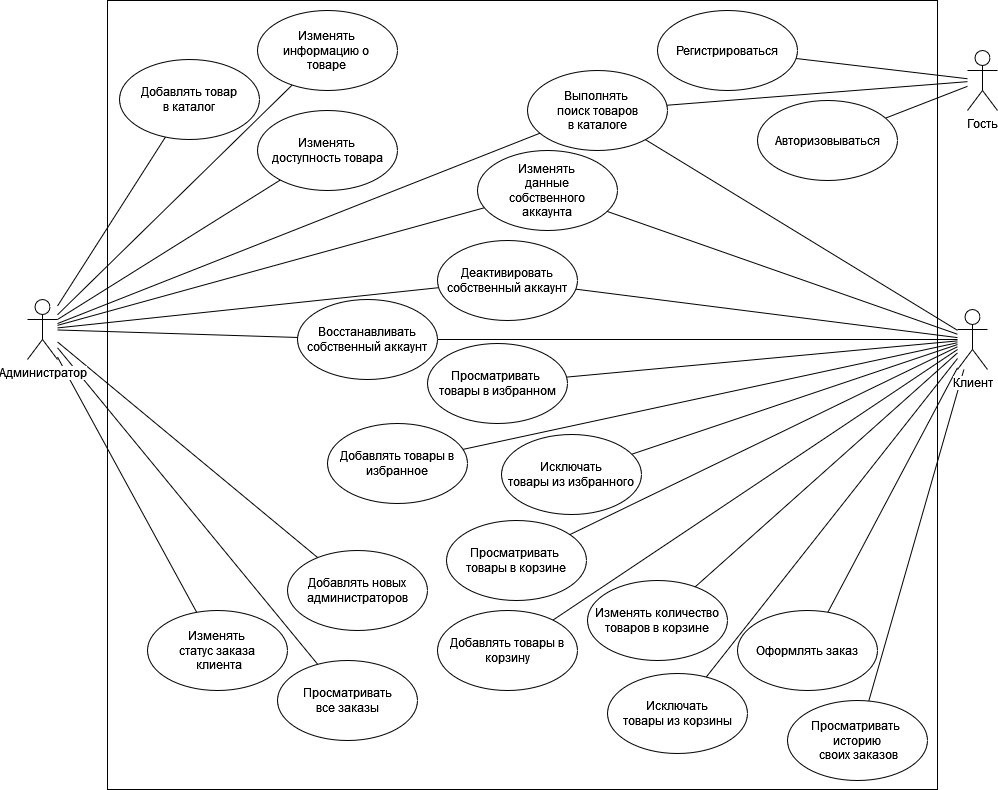


Рисунок 2.1 – Диаграмма вариантов использования

Согласно диаграмме, возможности гостя ограничиваются просмотром товаров каталога, регистрацией и авторизацией.

Авторизованный клиент может управлять данными своего аккаунта, деактивировать его и восстановить. Также клиент может добавлять, исключать товары из своей коллекции избранного и корзины. Кроме того, клиент может изменять количество товаров в своей корзине, оформлять их заказ и просматривать историю своих заказов с их текущим статусом.

Администратор может добавлять товары в каталог, изменять информацию о них, изменять их доступность. Удаление товаров нежелательно, так как может повлечь за собой нарушение целостности базы данных. Также администраторы могут изменять статусы заказов клиентов, просматривать заказы всех клиентов добавлять новых администраторов (регистрация в приложении подразумевает назначение роли клиента по умолчанию).

2.2 Структура web-приложения

Диаграмма развёртывания разрабатываемого распределённого web-приложения представлена на рисунке 2.2. Она включает в себя web-сервер, сервер базы данных и клиента.

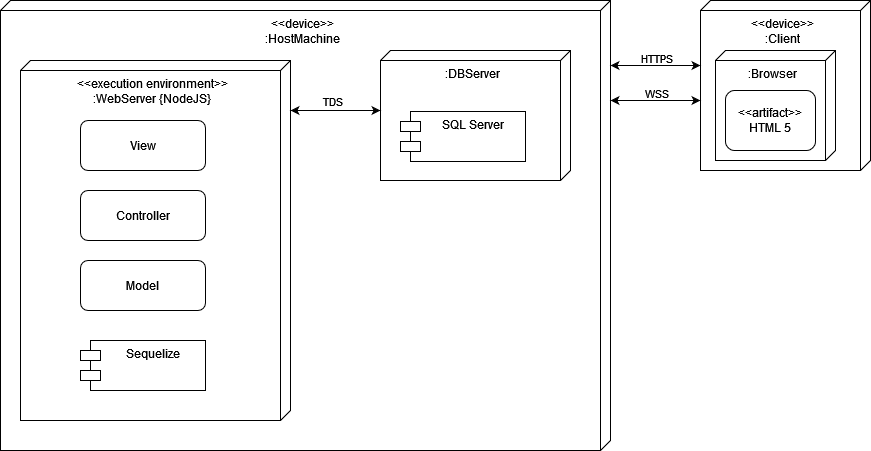


Рисунок 2.2 – Диаграмма развёртывания web-приложения

Web-сервер реализуется с использованием платформы Node.js. Для хранения данных была выбрана СУБД Microsoft SQL Server, а соединение с ней осуществляется по протоколу Tabular Data Stream (TDS). Выполнение запросов к базе данных обеспечивается использованием ORM Sequelize.

Для реализации приложения было решено использовать паттерн MVC (Model – View – Controller) [1]. Это позволяет разделить приложение на логические компоненты, каждый из которых отвечает за определённый аспект его функциональности. Модель отвечает за управление данными приложения. Представление отвечает за отображение данных пользователю и взаимодействие с пользователем. Контроллер обрабатывает входящие запросы от пользователя, взаимодействует с моделью для получения необходимых данных и обновляет представление в соответствии с этими данными. Таким образом, контроллер является посредником между моделью и представлением, управляя потоком данных и логикой приложения. Такой подход упрощает разработку, тестирование и поддержку приложения, обеспечивает его гибкость и расширяемость.

Для обмена данными между клиентом и web-сервером используются протоколы HTTPS и WSS, обеспечивающие безопасность и шифрование передаваемой информации. Это гарантирует защиту конфиденциальности пользовательских данных и обеспечивает безопасное взаимодействие с приложением через интернет.

2.3 Проектирование базы данных

Проектирование базы данных приложения предусматривает создание девяти таблиц, каждая из которых хранит определенный тип информации, необходимой для функционирования приложения. В этих таблицах содержатся данные об аккаунтах пользователей, товарах каталога, корзинах клиентов, их заказах, списках избранных товаров, персональные данные клиентов и другая информация.

Каждая запись в таблицах обладает собственным уникальным идентификатором, что обеспечивает возможность связывания данных между различными таблицами посредством внешних ключей. Это позволяет эффективно организовывать структуру базы данных и обеспечивать целостность.

Логическая схема базы данных представлена на рисунке 2.3.

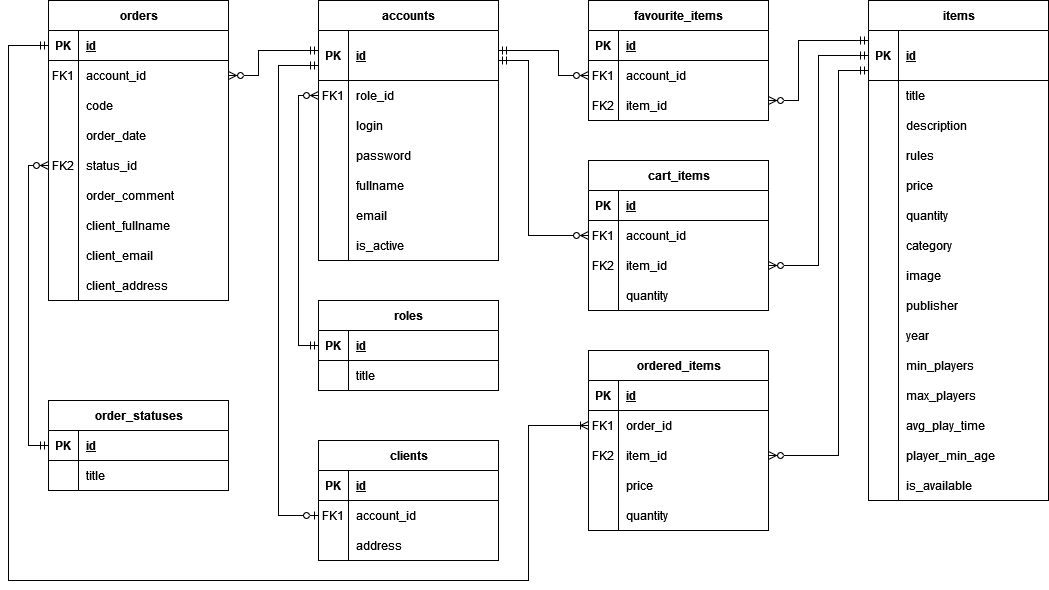


Рисунок 2.3 – Логическая схема базы данных

Таблица roles содержит список возможных ролей пользователей. Описание её столбцов представлено в таблице 2.1.

Таблица 2.1 – Описание столбцов таблицы roles

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | Идентификатор роли. |
| title | nvarchar(255) | Название роли. |

Таблица accounts содержит информацию об аккаунтах пользователей. Описание её столбцов представлено в таблице 2.2. Пароли пользователей хешируются, а имена и почтовые адреса шифруются. Хранение ключей пользователей для зашифрования и расшифрования личных данных в этой базе данных не предусматривается. Для этой цели можно использовать СУБД Redis.

Таблица 2.2 – Описание столбцов таблицы Accounts

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | Идентификатор аккаунта. |
| role\_id | int | Идентификатор роли пользователя. |
| login | nvarchar(255) | Логин. |
| password | nvarchar(MAX) | Пароль. |
| fullname | nvarchar(MAX) | Полное имя пользователя. |
| email | nvarchar(MAX) | Адрес электронной почты пользователя. |
| is\_active | bit | Флаг активности аккаунта. |

Таблица clients содержит специфическую информацию о клиентах. Описание её столбцов представлено в таблице 2.3. Адреса клиентов также шифруются.

Таблица 2.3 – Описание столбцов таблицы clients

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | Идентификатор клиента. |
| account\_id | int | Идентификатор аккаунта клиента. |
| address | nvarchar(MAX) | Адрес клиента. |

Таблица items содержит информацию о товарах (настольных играх) в каталоге. Описание её столбцов представлено в таблице 2.4.

Таблица 2.4 – Описание столбцов таблицы items

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | Идентификатор товара. |
| title | nvarchar(255) | Название товара. |
| description | nvarchar(MAX) | Описание товара. |
| rules | nvarchar(MAX) | Правила игры. |
| price | money | Цена товара. |
| quantity | int | Количество товара (штук). |
| category | nvarchar(255) | Категория товара. |
| image | nvarchar(MAX) | Путь к изображению товара. |
| publisher | nvarchar(225) | Издатель товара. |
| year | int | Год издания. |
| min\_players | int | Минимальное количество игроков. |
| max\_players | int | Максимальное количество игроков. |
| avg\_play\_time | int | Среднее время игры (минут). |
| player\_min\_age | int | Ограничение по возрасту. |
| is\_available | bit | Флаг доступности товара. |

Таблица cart\_items содержит информацию о корзинах клиентов. Описание её столбцов представлено в таблице 2.5.

Таблица 2.5 – Описание столбцов таблицы cart\_items

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | Идентификатор записи о товаре в корзине. |
| account\_id | int | Идентификатор аккаунта. |
| item\_id | int | Идентификатор товара. |
| quantity | int | Количество товара в корзине. |

Таблица order\_statuses содержит список возможных статусов заказа. Описание её столбцов представлено в таблице 2.6. Как и таблица ролей, таблица статусов заказа представляет собой перечисление допустимых значений.

Таблица 2.6 – Описание столбцов таблицы order\_statuses

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | Идентификатор статуса. |
| title | nvarchar(255) | Название статуса. |

Таблица orders содержит информацию о заказах клиентов. Описание её столбцов представлено в таблице 2.7. Комментарий к заказу, имя клиента, его почта и адрес шифруются ключом аккаунта. Номер заказа генерируется случайно.

Таблица 2.7 – Описание столбцов таблицы orders

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | Идентификатор записи о заказе. |
| account\_id | int | Идентификатор клиента. |
| code | nvarchar(255) | Номер заказа. |
| order\_date | datetime | Дата оформления заказа (UTC). |
| status\_id | int | Идентификатор статуса заказа. |
| order\_comment | nvarchar(MAX) | Комментарий к заказу. |
| client\_fullname | nvarchar(MAX) | Полное имя клиента. |
| client\_email | nvarchar(MAX) | Электронная почта клиента. |
| client\_address | nvarchar(MAX) | Адрес клиента (адрес доставки). |

Таблица favourite\_items содержит информацию об избранных товарах. Описание её столбцов представлено в таблице 2.8.

Таблица 2.8 – Описание столбцов таблицы favourite\_items

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | Идентификатор записи об избранном товаре. |
| account\_id | int | Идентификатор аккаунта. |
| item\_id | int | Идентификатор избранного товара. |

Таблица ordered\_items содержит информацию о заказанных товарах. Описание её столбцов представлено в таблице 2.9.

Таблица 2.9 – Описание столбцов таблицы ordered\_items

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | Идентификатор записи о заказанном товаре. |
| order\_id | int | Идентификатор заказа. |
| item\_id | int | Идентификатор заказанного товара. |
| price | decimal | Цена товара на момент оформления заказа. |
| quantity | int | Количество экземпляров товара в заказе. |

Данные таблицы позволяют хранить всю основную информацию о товарах и пользователях приложения.

2.4 Проектирование сервера web-приложения

Для разработки сервера была выбрана многослойная архитектура, которая позволяет разделить функциональность на отдельные слои, каждый из которых выполняет свою специфическую задачу. Это упрощает поддержку и развитие системы, так как изменения в одном слое не влияют на другие слои.

Первый слой в этой архитектуре – это middleware, которое обрабатывает входящие запросы к серверу. Middleware отвечает за аутентификацию и авторизацию пользователей, проверяя их учётные данные и определяя, имеют ли они право доступа к определенным ресурсам или функциям приложения.

После прохождения middleware запросы направляются к маршрутизаторам. Каждый маршрутизатор отвечает за определенное представление (view) в приложении и возвращает соответствующую страницу при получении GET-запроса. Маршрутизаторы обеспечивают навигацию по приложению, перенаправляя запросы к соответствующим контроллерам.

Контроллеры являются центром обработки запросов. Они соответствуют представлениям и содержат основную логику приложения. В случае, если разные представления требуют одинаковой обработки запроса, используется один контроллер, что позволяет избежать дублирования кода и упрощает поддержку работоспособности web-приложения.

Сервисы используются контроллерами для взаимодействия с моделями и выполнения бизнес-логики. Большинство сервисов связаны с моделями базы данных через ORM Sequelize для выполнения SQL-запросов, но также могут использоваться и несвязанные с базой данных сервисы для таких целей, как:

* работа с cookies;
* шифрование, расшифрование, хеширование данных;
* работа с JSON Web Tokens (JWT) для аутентификации;
* управление личными ключами пользователей;
* работа с базой данных Redis.

Такая многослойная архитектура обеспечивает гибкость, масштабируемость и лёгкость поддержки и развития серверного приложения. Каждый слой имеет четко определенные обязанности, что позволяет разработчикам эффективно организовывать код и легко вносить изменения без нарушения работы других компонентов системы.

# **3 Разработка web-приложения**

3.1 Разработка серверной части приложения

Серверная часть web-приложения реализует его бизнес-логику. Для разработки сервера был выбран фреймворк Express.js – минималистичный и гибкий web-фреймворк для приложений Node.js, предоставляющий обширный набор функций для web-приложений [2].

Для доступа к данным СУБД Microsoft SQL Server используется ORM Sequelize – это инструмент для организации взаимодействия между платформой Node.js и реляционными базами данными без использования специального языка запросов SQL [3].

В структуре сервера были выделены следующие модули:

* controllers – набор контроллеров сервера, обрабатывающих запросы;
* database – набор моделей Sequelize для выполнения SQL-запросов;
* middleware – набор middleware для аутентификации, авторизации, проверки наличия необходимых привилегий с использованием библиотеки CASL;
* routes – набор маршрутизаторов запросов;
* security – наборы возможных ролей, привилегий, SSL-сертификат;
* services – набор сервисов, выполняющих запросы к базе данных с использованием моделей Sequelize;
* собственно файл server.js, инициализирующий сервер.

Для инициализации HTTPS-сервера используется следующий код, представленный в листинге 3.1.

|  |
| --- |
| const app = express();  const server = await (async () => {  await redisService.connect();  console.log('Redis connection established');  await dbService.connect();  console.log('Database connection established');  await dbService.ensureDefaultRolesExist();  console.log('Default roles ensured to exist');  await dbService.ensureDefaultOrderStatusesExist();  console.log('Default order statuses ensured to exist');  await dbService.ensureAdminAccountExist();  console.log('Admin account ensured to exist');  return createServer({  key: readFileSync(process.env.SSL\_SERVER\_KEY\_PATH),  cert: readFileSync(process.env.SSL\_SERVER\_CRT\_PATH),  }, app);  })().catch(err => {  console.error(err);  console.log(`Server didn't start due to error`);  process.exit(1);  }); |

Листинг 3.1 – Инициализация HTTPS-сервера

В приведенном выше листинге проверяется установление соединения с СУБД Redis, затем с SQL Server. Redis необходим для хранения refresh-токенов и ключей пользователей. Затем проверяется наличие необходимых ролей в базе данных, статусов заказов и хотя бы одного аккаунта администратора. Если их недостаёт, они создаются со значениями, определёнными в файле конфигурации .env. После этого инициализируется HTTPS-сервер. Для работы данного сервера используется самоподписанный SSL-сертификат.

Для инициализации WebSocket-сервера используется следующий код, представленный в листинге 3.2. Данный сервер предназначен для выполнения SELECT-запросов к базе данных SQL Server, которые администратор может ввести в соответствующую панель представления query. Данная панель доступна только администраторам. WebSocket-сервер возвращает результаты выполнения запросов представлению query, как успешные, так и ошибочные.

|  |
| --- |
| const wss = new WebSocketServer({ server: server });  wss.on('connection', (ws) => {  console.log('New WebSocket connection!');  ws.on('message', async (message) => {  try {  const results = await dbService.executeSelectQuery(message.toString());  ws.send(JSON.stringify(results));  } catch (error) {  console.error(error);  ws.send(JSON.stringify({ error: `Error: ${error.message}` }));  }  });  ws.on('close', () => { console.log('WebSocket connection closed.'); });  }); |

Листинг 3.2 – Инициализация WebSocket-сервера

В приложении используется JWT-аутентификация. JSON Web Token (JWT) представляет собой формат структурированного токена безопасности, используемый для кодирования данных JSON [4].

При аутентификации пользователь получает access-токен, который содержит информацию о его аутентификации и предоставляет доступ к защищенным ресурсам приложения. Также пользователю выдается refresh-токен, который используется для обновления access-токена после его истечения срока действия.

Белый список refresh-токенов позволяет приложению контролировать список разрешенных токенов, что повышает безопасность системы. При обновлении access-токена приложение проверяет, есть ли переданный refresh-токен в белом списке в Redis. Если токен есть в списке, то выполняется обновление access-токена, иначе пользователю назначается роль гостя. Этот механизм частично защищает от возможных атак с перехватом токенов и подделкой запросов. Нагрузка токенов содержит идентификатор текущего пользователя, его роль и UUID для разграничения сохранённых токенов для разных браузеров и устройств.

Аутентификация обеспечивается middleware-функцией authenticate, представленной в листинге 3.3.

|  |
| --- |
| export default async function authenticate(req, res, next) {  const accessToken = cookieService.getAccessToken(req);  const decodedAccess = accessToken && jwtService.verifyAccessToken(accessToken);  if (decodedAccess) {  req.account = decodedAccess.account;  return next();  }  const refreshToken = cookieService.getRefreshToken(req);  const decodedRefresh = refreshToken && jwtService.verifyRefreshToken(refreshToken);  if (decodedRefresh && refreshToken === await jwtService.getRefreshToken(decodedRefresh.account?.uuid)) {  const { account } = decodedRefresh;  cookieService.setAccessToken(res, jwtService.generateAccessToken(account.id, account.role, account.uuid));  req.account = account;  return next();  }  cookieService.clearAccessToken(res);  cookieService.clearRefreshToken(res);  req.account = { id: null, role: Role.Guest, uuid: null };  return next();  } |

Листинг 3.3 – Функция authenticate

Для обеспечения авторизации используется система правил библиотеки CASL [5]. В приложении определяются различные объекты (например, товар и заказ) и действия, которые над ними можно выполнять (например, управлять и создавать). Задание привилегий в зависимости от используемой роли происходит в middleware-функции authorize, листинг которой приведён в приложении А. Проверка наличия нужных привилегий для выполнения запросов сервером происходит при помощи вызова другой middleware-функции – checkAbility. В случае ограничения доступа к ресурсу клиент получает сообщение о об ошибке доступа.

После прохождения middleware запрос рассматривается корневым маршрутизатором (routes/root.js), который в зависимости от адреса запрашиваемого ресурса передаёт управление маршрутизаторам следующего уровня. Каждому представлению соответствует свой маршрутизатор, который обрабатывает запросы к адресам, начинающимся на название своего представления. Маршрутизатор страницы добавления товара в каталог представлен в листинге 3.4.

|  |
| --- |
| const router = new Router({ mergeParams: true });  router.get('/', asyncHandler(checkAbility(Action.Manage, new Subject.Item())),  asyncHandler(async (req, res) => res.render('add-item', { account: req.account })));  const apiRouter = new Router({ mergeParams: true });  router.use('/api', apiRouter);  apiRouter.post('/add-item', asyncHandler(checkAbility(Action.Manage, new Subject.Item())),  upload.single('image'), asyncHandler(addItemController.handleAddItem)); next();  } |

Листинг 3.4 – Маршрутизатор страницы добавления товара в каталог

Для загрузки изображений на сервер используется middleware multer [6]. При этом проверяется расширение загружаемых файлов. Конфигурация этого middleware представлена в листинге 3.5.

|  |
| --- |
| const upload = multer({  storage: multer.diskStorage({  destination: (req, file, cb) => {  cb(null, path.join(process.env.STATIC\_DIR, process.env.UPLOADED\_ITEM\_IMAGE\_DIR));  },  filename: (req, file, cb) => {  cb(null, `${new Date().getTime()}-${cryptoService.generateUuid()}${path.extname(file.originalname)}`);  }  }),  fileFilter: function (req, file, cb) {  var ext = path.extname(file.originalname);  const allowedExts = ['.png', '.jpg', '.jpeg', '.gif'];  if (!allowedExts.includes(ext)) {  return cb(new Error(`Калі ласка, абярыце выяву памерам да 1МБ. Дазволеныя пашырэнні: ${allowedExts.join(', ')}`));  }  cb(null, true);  },  limits: { fileSize: 1024 \* 1024 }  }); |

Листинг 3.5 – Middleware для загрузки изображений на сервер

Обработчики запросов на получение представлений прописаны непосредственно в маршрутизаторах. Однако все остальные запросы к серверу обрабатываются в контроллерах. Контроллеры, как и маршрутизаторы, соответствуют их использующим представлениям, но для обработки повторяющихся на разных представлениях запросов есть контроллер commonApi, что предотвращает дублирование кода. Пример реализации обработки контроллером корзины запроса на изменение количества товара в корзине представлен в листинге 3.6. Листинг контроллера account представлен в приложении Б.

|  |
| --- |
| export async function handleUpdateCartItemQuantity(req, res, next) {  const { item\_id, cart\_quantity } = req.body;  let item = await itemService.getById(item\_id);  let new\_cart\_quantity;  if (item.quantity > 0 && cart\_quantity > 0) {  item = await cartItemService.updateCartItemQuantityByAccountId(item\_id, req.account.id, cart\_quantity);  new\_cart\_quantity = item?.quantity || 0;  } else {  await cartItemService.deleteByItemIdAndAccountId(item\_id, req.account.id);  new\_cart\_quantity = 0;  }  return res.json({ new\_cart\_quantity });  } |

Листинг 3.6 – Обработчик изменения количества товара в корзине

Логика контроллеров сравнительно высокоуровневая – здесь нет непосредственного взаимодействия с моделями базы данных. Вместо этого контроллеры обращаются к слою сервисов, где формируются запросы к базе данных через модели Sequelize. Сервисы имеют узкую специализацию: так, есть отдельные сервисы для работы с моделью аккаунта, для выполнения криптографических операций и так далее. Сервисы могут обращаться к функциональности друг друга. Реализация функции сервиса коллекции избранного для перемещения доступных товаров из избранного в корзину представлена в листинге 3.7.

|  |
| --- |
| export async function moveAllToCartByAccountId(accountId) {  const favItemsToMove = await db.favourite\_items.findAll({  where: { account\_id: accountId },  include: [{  model: db.items, as: 'item', required: true,  where: { is\_available: true, quantity: { [Op.gt]: 0 } }  }] });  await Promise.all(favItemsToMove.map(async favItem => {  await db.cart\_items.findOrCreate({  where: { item\_id: favItem.item.id, account\_id: accountId },  defaults: { item\_id: favItem.item.id, account\_id: accountId, quantity: 1 }  });  await db.favourite\_items.destroy({ where: { id: favItem.id } });  }));  } |

Листинг 3.7 – Функция сервиса для перемещения избранных товаров в корзину

Сервисы взаимодействуют с базой данных через модели Sequelize. При реализации приложения сначала были созданы необходимые таблицы при помощи SQL-скрипта как более удобного для этих целей средства. Затем с использованием пакета sequelize-auto были автоматически сгенерированы все модели с их отношениями между собой.

Среди прочих, следует отметить сервис dbService, предназначенный для выполнения специфических операций с базой данных. Так, при каждом запуске сервера происходит проверка наличия в базе данных необходимых ролей пользователей, статусов заказа, хотя бы одного аккаунта с ролью администратора (если необходимых данных недостаёт, они автоматически добавляются с применением предопределённых значений). Реализация последней из перечисленных функций представлена в листинге 3.8.

|  |
| --- |
| export async function ensureAdminAccountExist() {  let adminAccount = await db.accounts.findOne({  include: [{ model: db.roles, as: 'role', where: { title: Role.Admin } }]  });  if (!adminAccount) {  await accountService.createAdmin(  process.env.DATABASE\_DEFAULT\_ADMIN\_LOGIN,  process.env.DATABASE\_DEFAULT\_ADMIN\_PASSWORD,  process.env.DATABASE\_DEFAULT\_ADMIN\_FULLNAME,  process.env.DATABASE\_DEFAULT\_ADMIN\_EMAIL  );  }  } |

Листинг 3.8 – Функция проверки наличия хотя бы одного аккаунта администратора

Для запуска сервера должны работать сервисы Redis и SQL Server. Сервер прослушивает предопределённый порт 443, а HTTP-запросы перенаправляются на путь HTTPS при помощи пакета express-redirect-to-https.

3.2 Разработка клиентской части приложения

В приложении используется движок представлений EJS.

Клиентская часть приложения включает в себя:

* файлы представлений EJS в папке views;
* файлы частичных представлений EJS в папке views/partials;
* изображения;
* CSS-стили.

Всего на клиентской части есть одиннадцать представлений, которые отображаются на странице как вкладки меню, и одно представление для отображения ошибки приложения при попытке неавторизованного доступа к ресурсу, попытке доступа к несуществующей странице или возникновении внутренней ошибки сервера. В зависимости от роли текущего пользователя ему будут доступны разные представления.

Среди частичных представлений следует обратить внимание на menu.ejs, где определяется отображаемое пользователю меню, позволяющее перемещаться между страницами приложения. Листинг этого представления приведён в приложении В. Элементы меню заданы как объекты в массиве, у каждого из которых прописан параметр condition, определяющий, когда данный пункт меню должен быть отображён. Некоторые вкладки меню отображаются всегда (например, catalog). Другие отображаются только пользователям с определённой ролью (например, вкладка add-item доступна только администраторам). Третьи видны только тогда, когда они активные (например, вкладка item, которую можно открыть не из меню, а по нажатию на блок товара). Условия отображения могут быть комбинированные.

Для удобства проверки текущей роли пользователя в представлениях-страницах определяются переменные isAdmin, isClient, isGuest, которые вместе с названием активной страницы передаются во включаемые частичные шаблоны. В зависимости от роли пользователя ему могут отображаться разные частичные представления на одной и той же странице. Пример выбора видимых форм на странице аккаунта представлен в листинге 3.9.

|  |
| --- |
| <div class="forms-container">  <% if (isGuest) { %>  <%- include('./partials/sign-up-form.ejs', { ...globals }) %>  <%- include('./partials/sign-in-form.ejs', { ...globals }) %>  <% } %>  <% if (isClient) { %>  <%- include('./partials/update-account-info-form.ejs', { ...globals }) %>  <%- include('./partials/update-client-account-info-form.ejs', { ...globals }) %>  <% } %>  <% if (isAdmin) { %>  <%- include('./partials/update-account-info-form.ejs', { ...globals }) %>  <%- include('./partials/create-new-admin-account-form.ejs', { ...globals }) %>  <% } %>  </div> |

Листинг 3.9 – Фрагмент представления account с условным включением частичных представлений

Для каждой страницы определён свой файл CSS-стилей, но для повторно используемых стилей есть файл common.css и файлы в папке style/utils.

Загружаемые изображения товаров содержатся в папке images/uploaded.

Также в каждую страницу включается файл common.js, который содержит универсальные JavaScript-функции.

Большинство запросов с клиента на сервер отправляются по Fetch API. В случае возникновения ошибки её текст выводится клиенту в модальном окне.

4 Тестирование web-приложения

Было проведено ручное тестирование приложения с использованием браузера Mozilla Firefox, которое включало в себя проверку выполнения основных сценариев использования приложения, проверку обработки вводимых невалидных данных, проверку доступа к различным частям приложения для пользователей с разными ролями и другое.

При открытии приложения происходит перенаправление на страницу каталога. По строке адреса приложения видно, что соединение защищено сертификатом, как показано на рисунке 4.1.

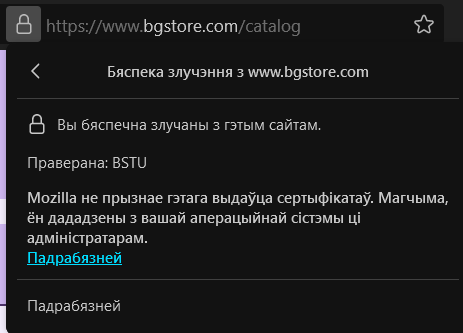


Рисунок 4.1 – Проверка применения SSL-сертификата

При попытке перейти на несуществующую страницу выводится сообщение о том, что такой страницы нет, как показано на рисунке 4.2.

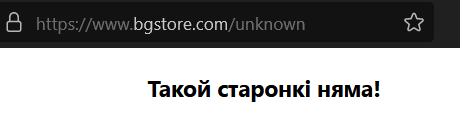


Рисунок 4.2 – Проверка недоступности несуществующей страницы

При регистрации клиента должна быть проверка на уникальность логина в базе данных. При конфликте выводится соответствующее сообщение об ошибке, представленное на рисунке 4.3.

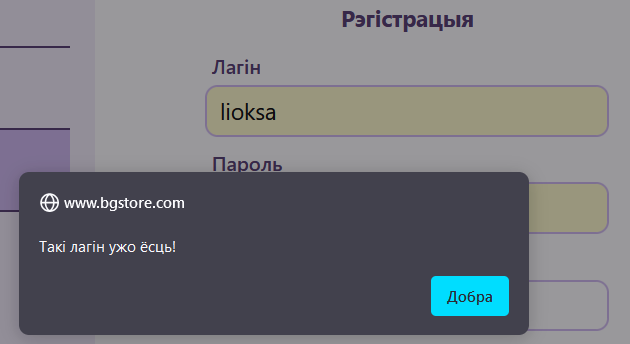


Рисунок 4.3 – Проверка уникальности логина

При попытке регистрации с некорректным адресом электронной почты выводится соответствующее сообщение, как показано на рисунке 4.4.

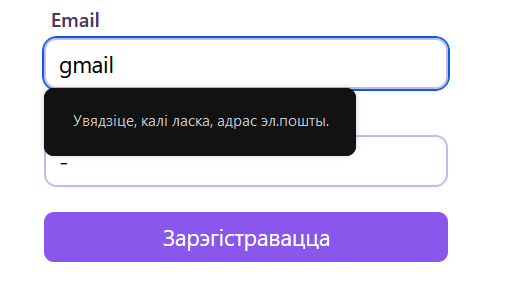


Рисунок 4.4 – Проверка корректности электронной почты при регистрации

При попытке неавторизованного доступа к странице, которая существует, но недоступна из-за ограничений роли пользователя, выводится соответствующее сообщение, как показано на рисунке 4.5. В данном тесте гость пытается получить доступ к защищённой странице с заказами клиентов.

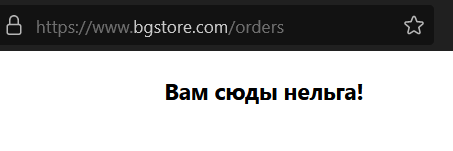


Рисунок 4.5 – Проверка недоступности страниц вследствие ограничений роли

Проверено, что при удалении cookie с JWT-токенами пользователь теряет доступ к защищённым страницам сайта.

При авторизации в качестве клиента на странице каталога можно добавлять товары в корзину и коллекцию избранного по нажатию на соответствующие кнопки блоков товаров, как показано на рисунке 4.6.

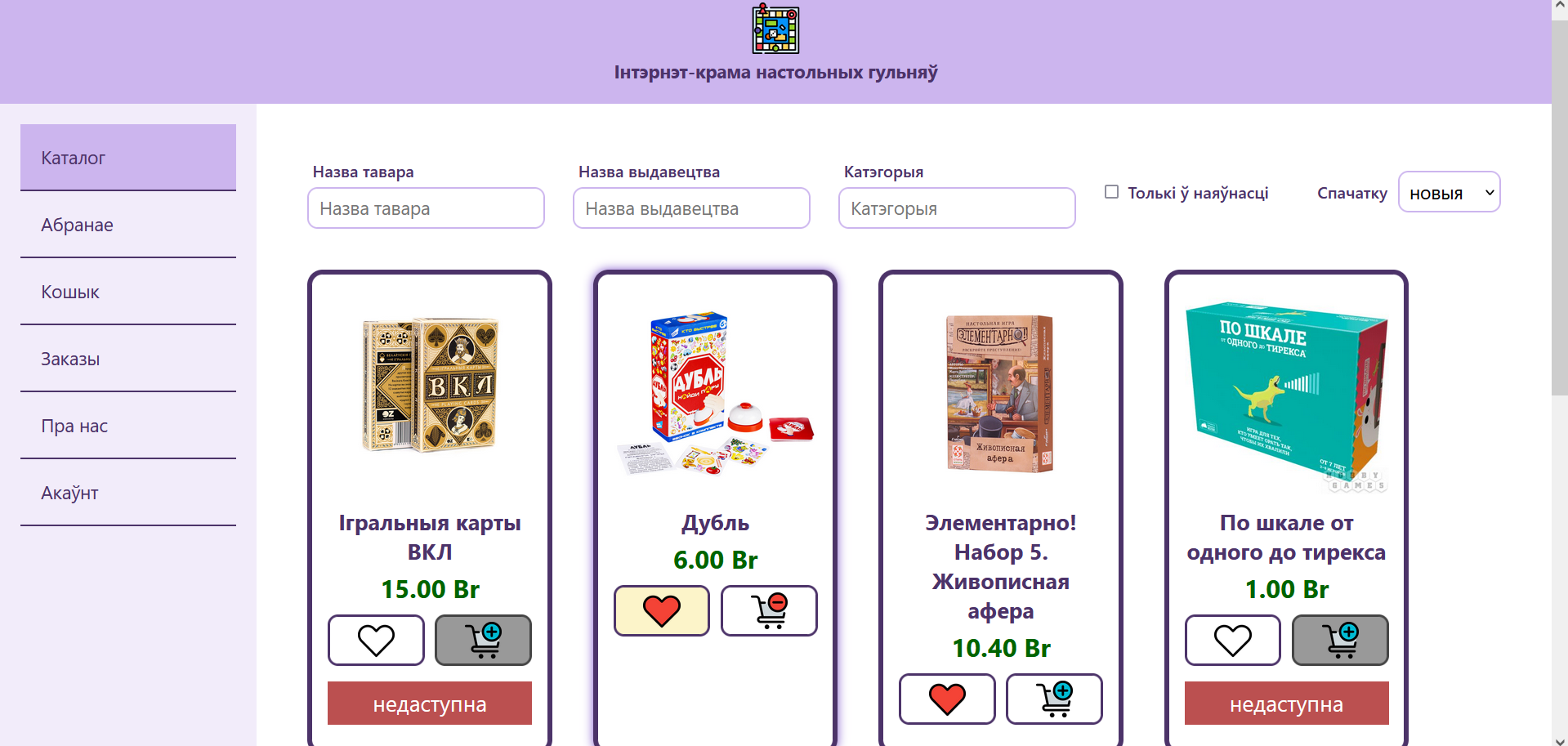


Рисунок 4.6 – Проверка корректного отображения состояния каталога для клиента

Были проведены проверки корректного поиска в каталоге, фильтрации по доступности товаров, сортировки.

На странице избранного было проверено, что при переносе избранных товаров в корзину недоступные товары не переносятся. Проверена очистка избранного.

На странице корзины было проверено, что нельзя установить значение количества товара в корзине больше количества товара на складе. Проверено, что при изменении количество товаров в корзине изменяется отображаемая стоимость товаров в корзине. Также было проверено, что при установлении администратором недоступности товара он исключается из корзин всех пользователей.

На странице аккаунта проверено, что корректно изменяются данные аккаунта и клиентские сведения.

На странице корзины было проверено, что клиенты не могут оформить заказ со слишком большим количеством товаров в корзине, о чём свидетельствует отображаемое сообщение, как показано на рисунке 4.7.

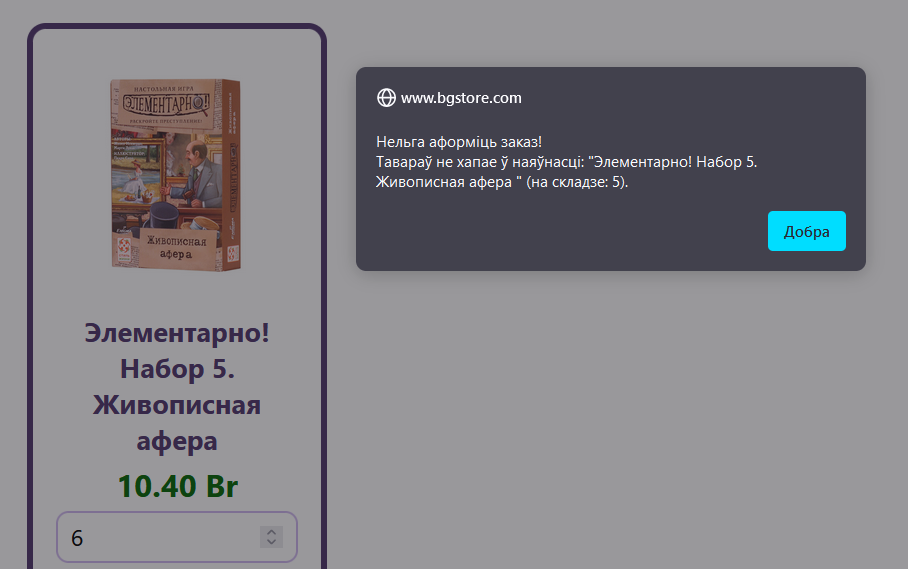


Рисунок 4.7 – Проверка запрета на оформление заказа с некорректным количеством товаров в корзине

На странице оформления заказа проверено, что с корректной корзиной клиент может оформить заказ, после чего отображается номер нового заказа, как показано на рисунке 4.8, и происходит переход на страницу истории заказов.

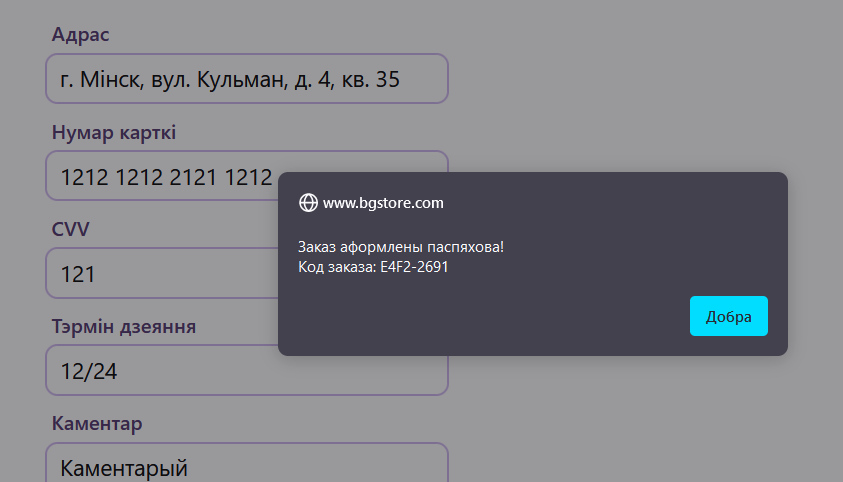


Рисунок 4.8 – Проверка возможности оформления заказа при корректной корзине

На странице заказов клиента проверено, что отображаются актуальные сведения о заказах.

На странице аккаунта проверено, что после деактивации при попытке входа с тем же логином и паролем аккаунт можно восстановить, как показано на рисунке 4.9.

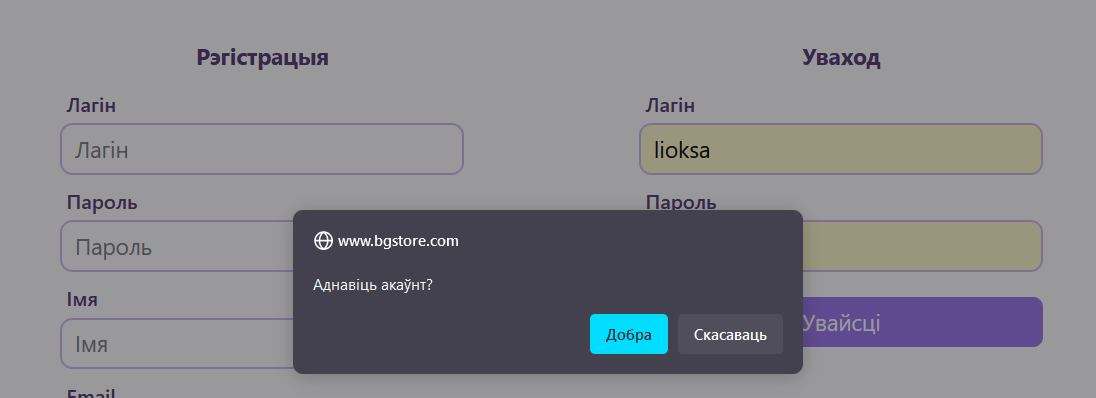


Рисунок 4.9 – Проверка возможности восстановления аккаунта

После авторизации в качестве администратора было проверено, что на странице аккаунта можно добавить новых администраторов и войти в приложение под их учётными записями.

На странице добавления товара была проверена возможность добавления товара в каталог.

На странице каталога проверено, что администратор может перейти к изменению товаров по нажатию на соответствующую кнопку блока товара.

На странице изменения товара было проверено, что администратор может изменять сведения о товаре, после чего они видны и другим пользователям.

На странице заказов было проверено, что администратор может изменить статус заказа, после чего он обновляется на странице, как показано на рисунке 4.10.

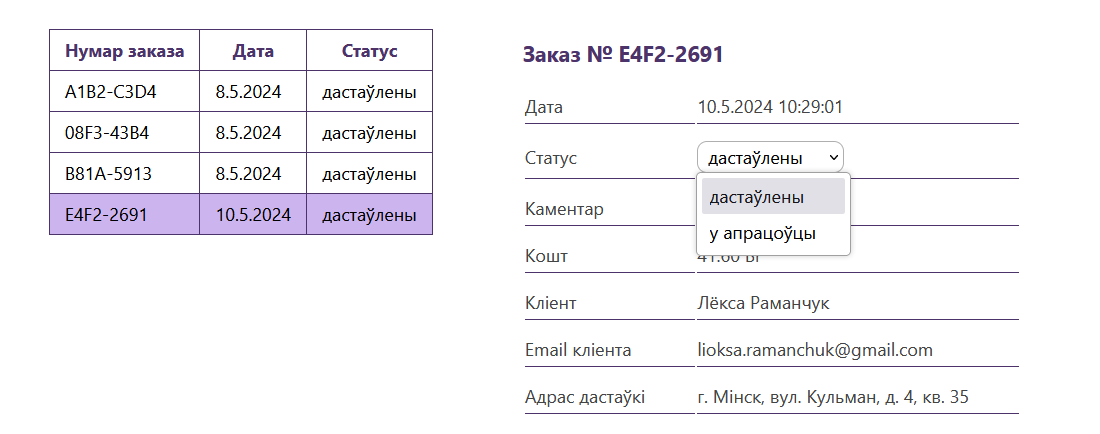


Рисунок 4.10 – Проверка обновления статуса заказа администратором

На странице выполнения SELECT-запросов к базе данных администратором было проверено, что исключается возможность использования в запросах ключевых слов «UPDATE», «DELETE», «DROP» и других, которые могли бы привести к повреждению базы данных, и что выводится соответствующее сообщение попытке их использования, как показано на рисунке 4.11.

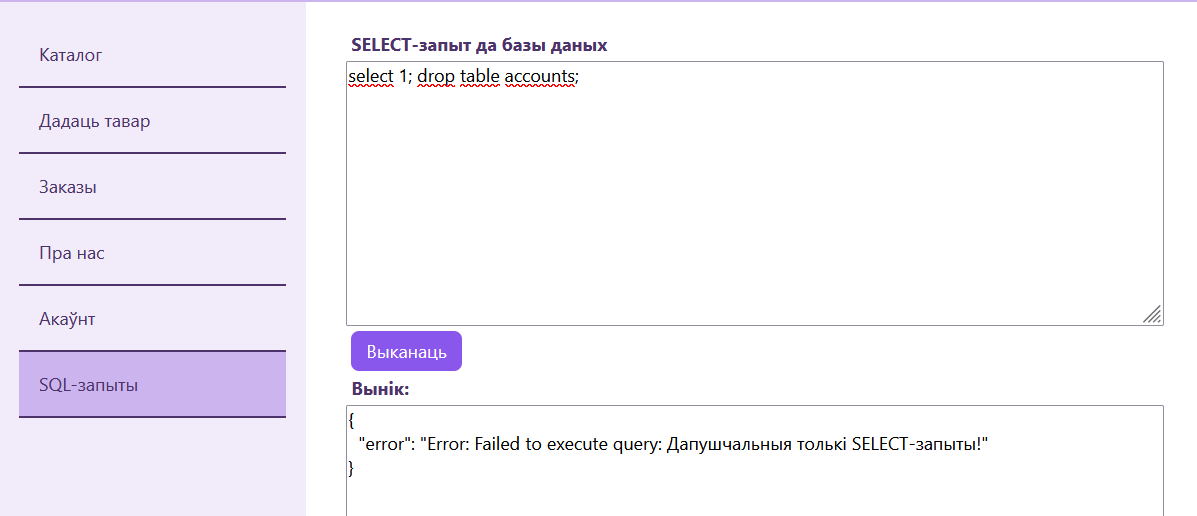


Рисунок 4.11 – Проверка запрета на использование ключевого слова «DROP» в поле ввода SQL-запроса

Также было проверено, что правильно составленные SELECT-запросы возвращают корректный результат, как показано на рисунке 4.12.

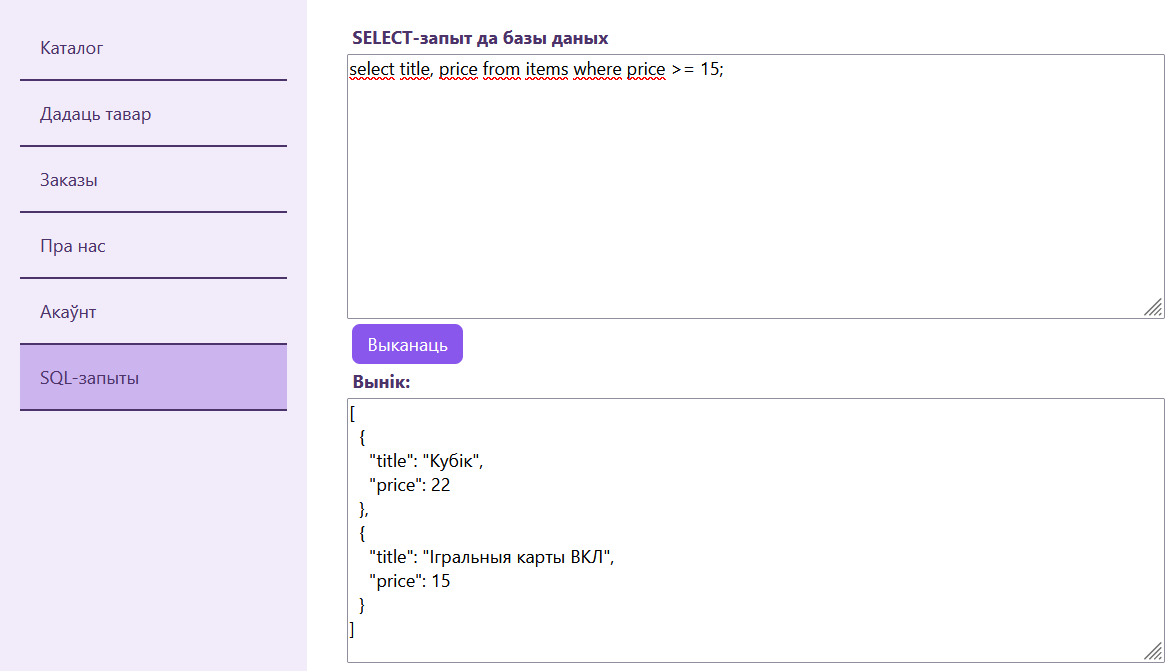


Рисунок 4.12 – Проверка корректности выполнения SELECT-запроса

Таким образом, в ходе тестирования приложения было успешно проверено, что все заявленные операции выполняются корректно.

# **5 Руководство пользователя**

После запуска web-сервера и открытия в браузере адреса web-приложения пользователь попадает на главную страницу – страницу каталога товаров, представленную на рисунке 5.1.

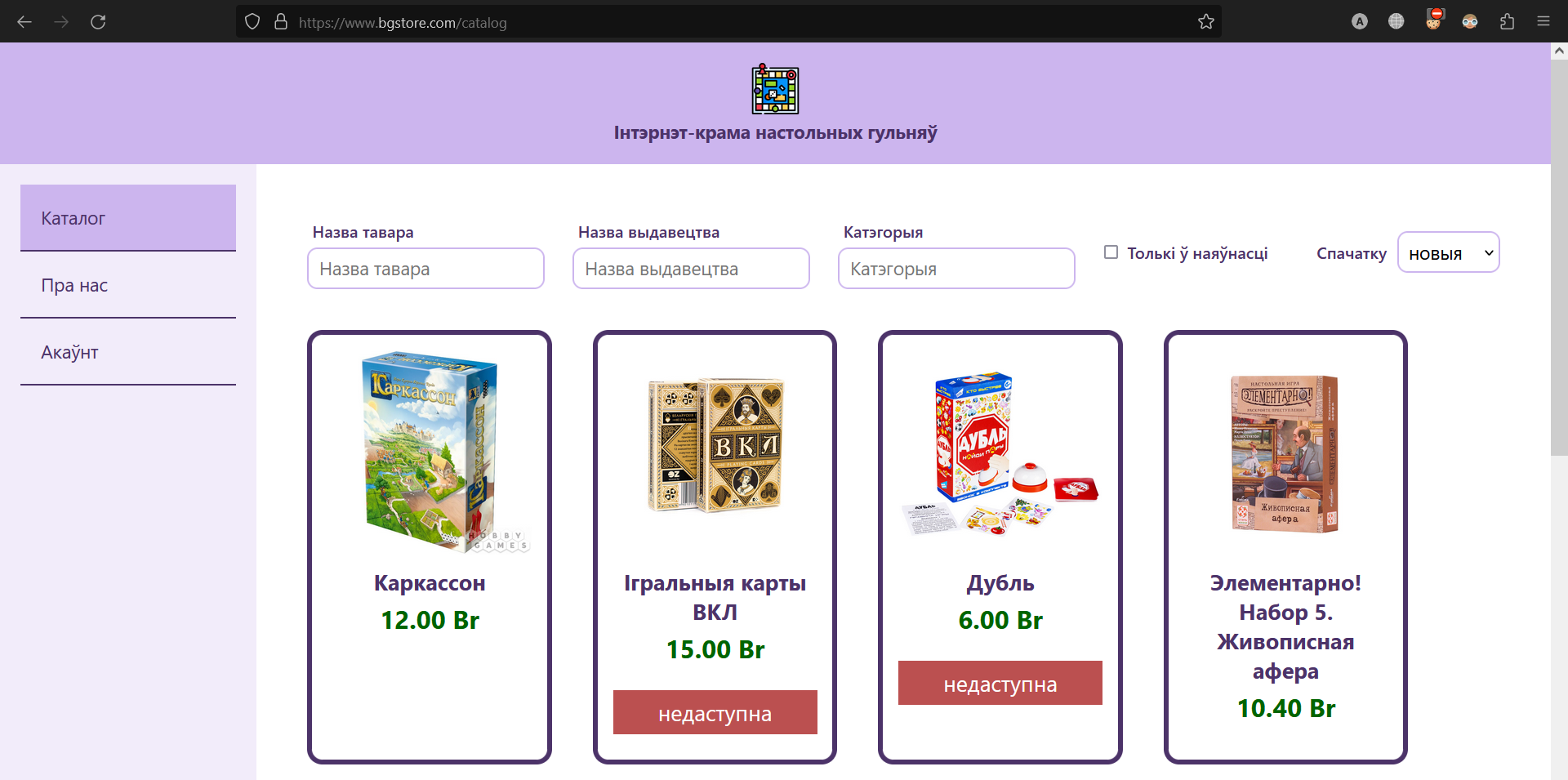


Рисунок 5.1 – Страница каталога для гостя

Каталог можно фильтровать по названию товара, названию издательства, категории, наличию товара на складе. Можно сортировать по дате добавления в каталог и по цене. По нажатию на блок товара происходит переход на страницу товара с подробной информацией о нём, как показано на рисунке 5.2.



Рисунок 5.2 – Страница с информацией о товаре

На вкладке аккаунта можно зарегистрироваться либо войти, если уже есть аккаунт. Страница аккаунта для гостя представлена на рисунке 5.3.

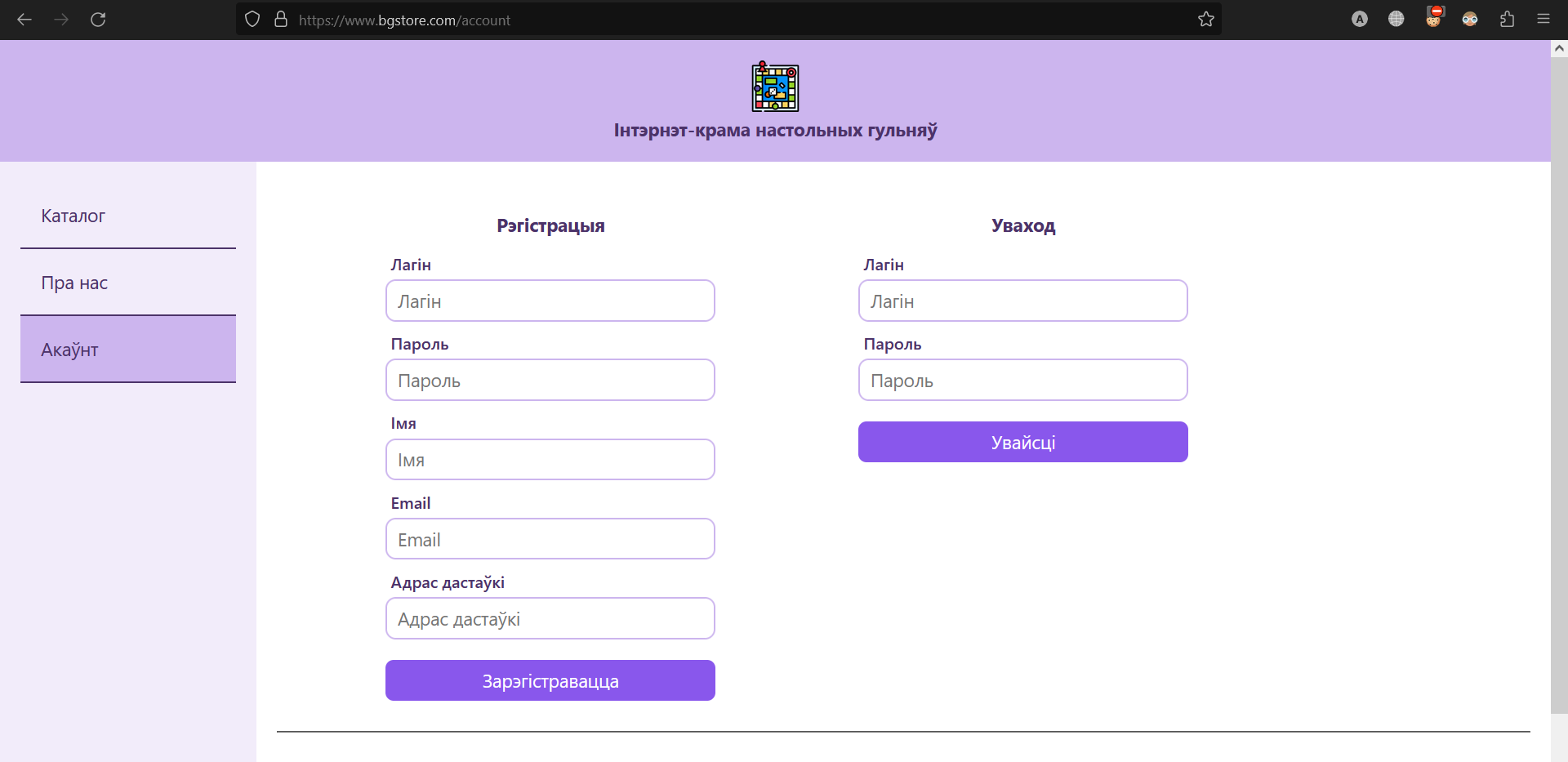


Рисунок 5.3 – Страница аккаунта для гостя

После входа в учётную запись администратора, на странице аккаунта можно изменить данные аккаунта и добавить нового администратора. Также можно выйти из аккаунта либо деактивировать его. После деактивации аккаунта его можно восстановить, если войти снова под своим логином и паролем. Вид страницы аккаунта для администратора представлен на рисунке 5.4.



Рисунок 5.4 – Страница аккаунта для администратора

Для администраторов блоки товаров содержат кнопки для перехода к изменению информации о выбранном товаре. Пример страницы изменения сведений о товаре представлен на рисунке 5.5.



Рисунок 5.5 – Страница изменения сведений о товаре

После изменения товара выводится соответствующее сообщение, как показано на рисунке 5.6.

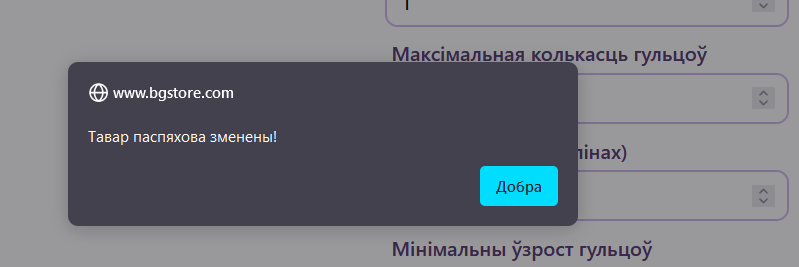


Рисунок 5.6 – Сообщение об успешном изменении товара

Страница добавления товара идентична странице изменения товара. На ней в соответствующие поля заносится информация о товаре, можно выбрать путь к файлу с изображением. По нажатию на кнопку добавления товара появляется сообщение об успешном завершении операции, как показано на рисунке 5.7.

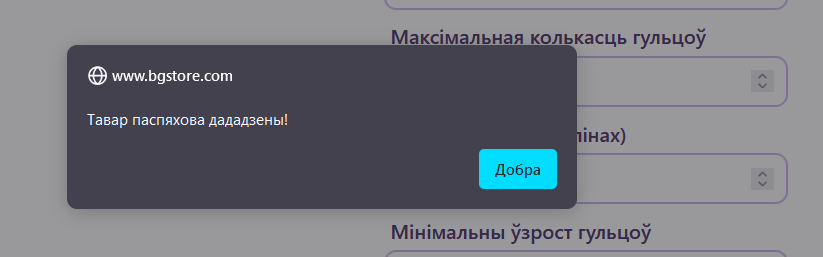


Рисунок 5.7 – Сообщение об успешном добавлении нового товара в каталог

На странице заказов администратор может просматривать информацию о заказах всех клиентов, в том числе список заказанных товаров, а также изменять статус заказа. Вид страницы заказов для администратора представлен на рисунке 5.8.



Рисунок 5.8 – Страница заказов для администратора

На странице SQL-запросов администратор может выполнять SELECT-запросы к базе данных. Исполнение других запросов запрещено вследствие потенциальной опасности повреждения базы данных. Вид страницы SQL-запросов для администратора представлен на рисунке 5.9.

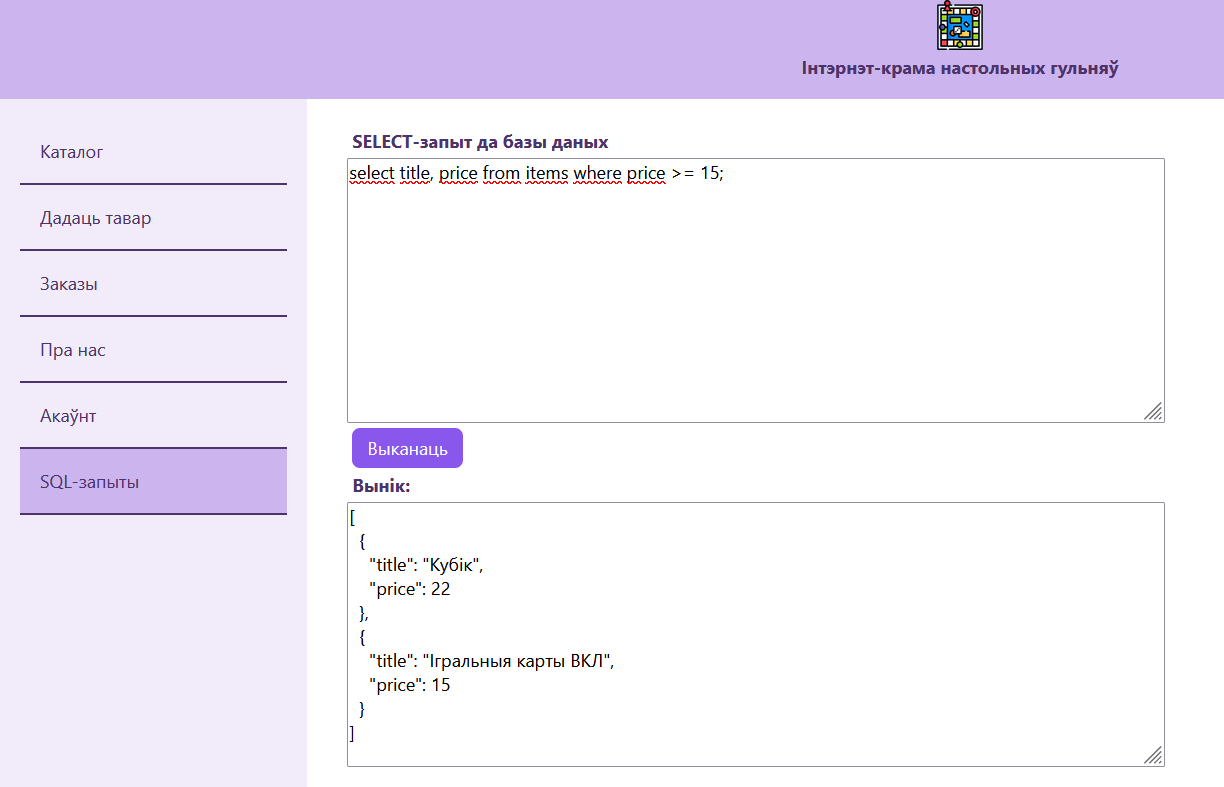


Рисунок 5.9 – Страница SQL-запросов для администратора

При входе в приложение в качестве клиента на странице аккаунта можно изменить персональные данные, изменить клиентскую информацию (адрес доставки), выйти из аккаунта, деактивировать его, затем восстановить.

На странице каталога клиент может добавлять и исключать товары из корзины и из своей коллекции избранного при нажатии на соответствующие кнопки. Вид страницы каталога для клиента показан на рисунке 5.10.

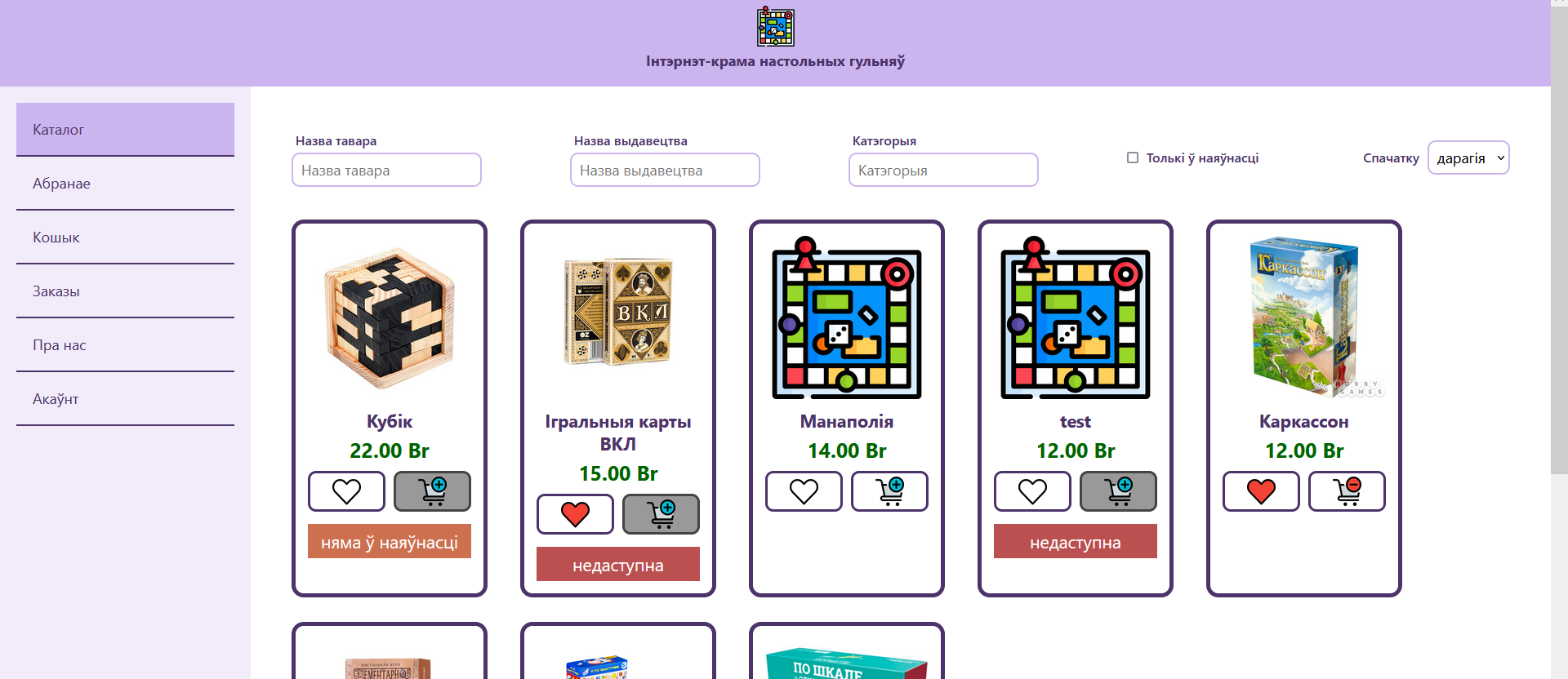


Рисунок 5.10 – Страница каталога для клиента

На странице избранного пользователь может исключать товары из избранного, добавлять и исключать товары из корзины. Также можно очистить коллекцию избранного либо переместить товары из неё в корзину. При этом товары, которых нет на складе или которые недоступны, не переносятся в корзину. Страница избранного показана на рисунке 5.11.

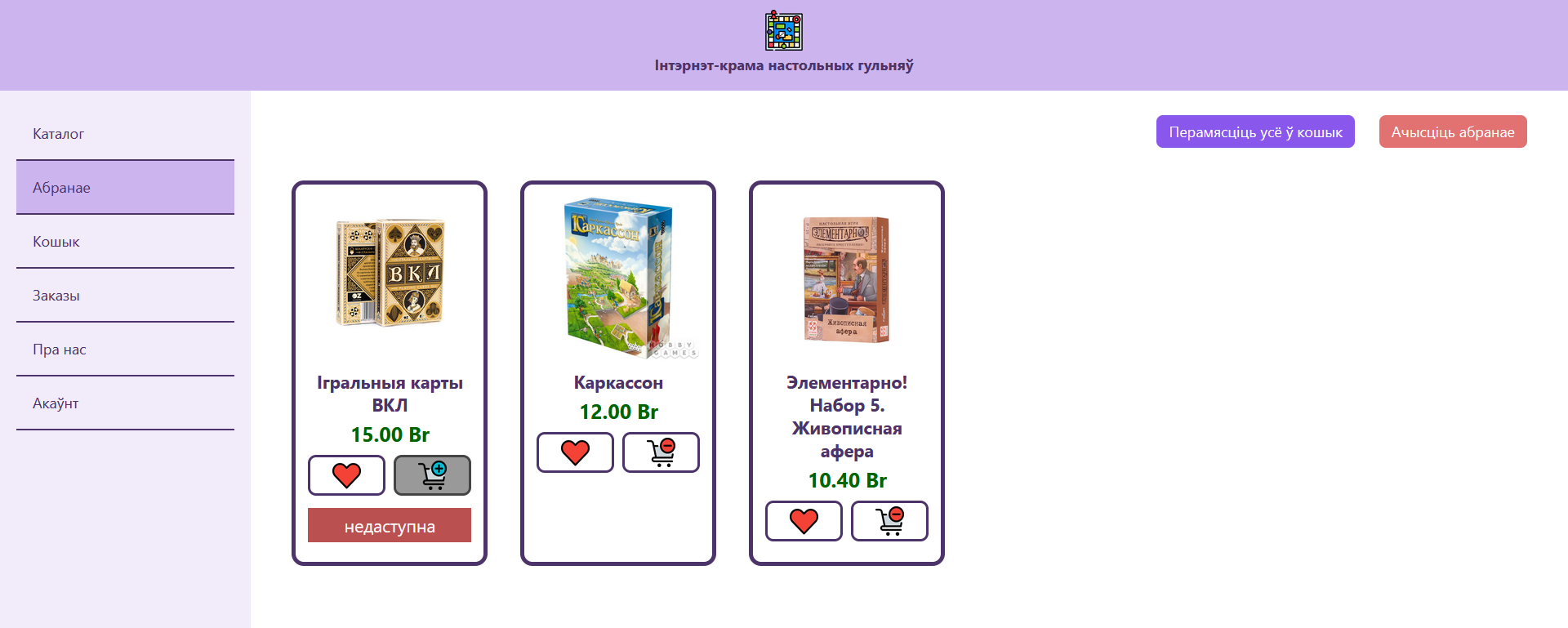


Рисунок 5.11 – Страница избранного

На странице корзины пользователь может изменять количество товаров в корзине, исключать их из корзины, добавлять и исключать из избранного. Также отображается стоимость корзины. Можно очистить корзину или перейти к оформлению заказа. Страница корзины показана на рисунке 5.12.

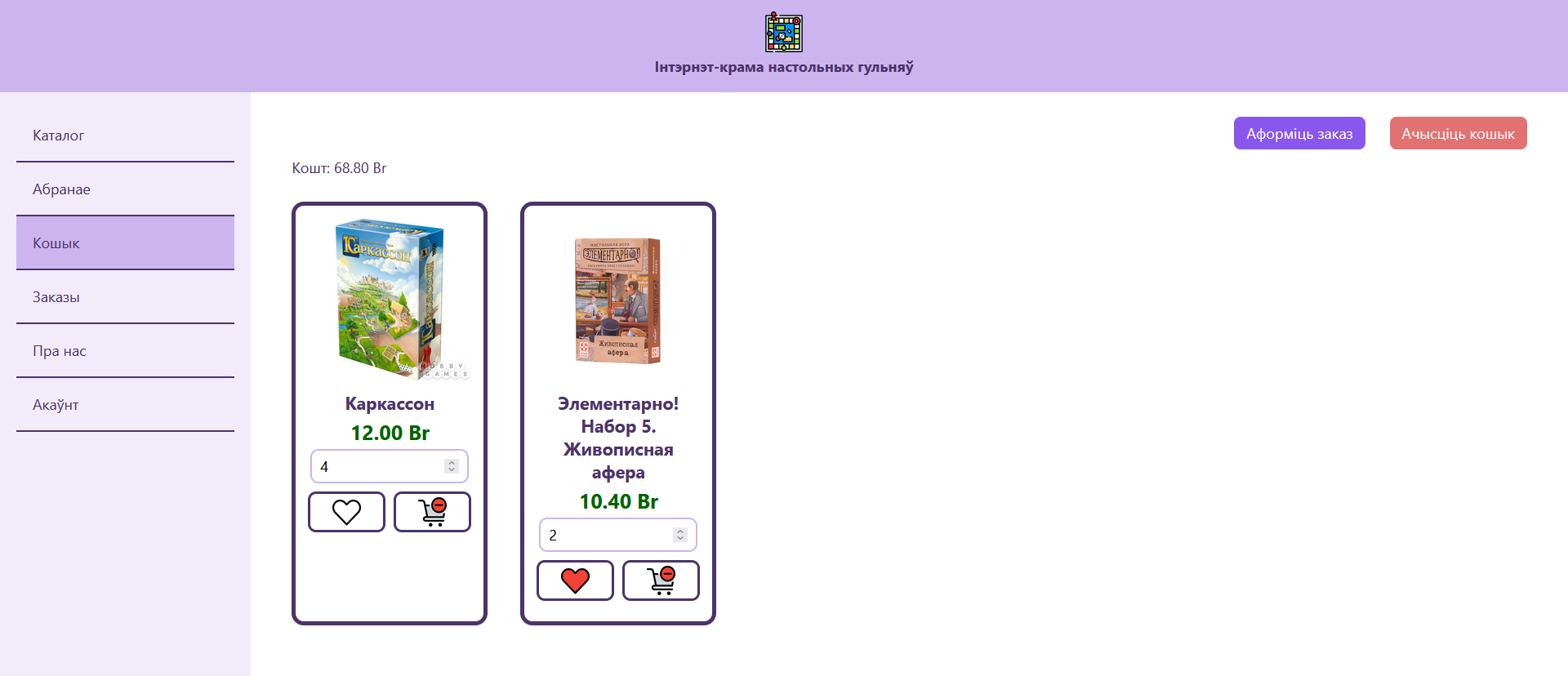


Рисунок 5.12 – Страница корзины

Страница заказов для пользователя такая же, как и для администратора, но в списке заказов отображаются заказы только этого пользователя и нет возможности изменить статус заказа, а только просмотреть.

При оформлении заказа необходимо указать данные карты, можно указать комментарий к заказу (необязательно). Если корзина не содержит недопустимых товаров, то по нажатию на соответствующую кнопку оформляется заказ. Страница оформления заказа показана на рисунке 5.13.

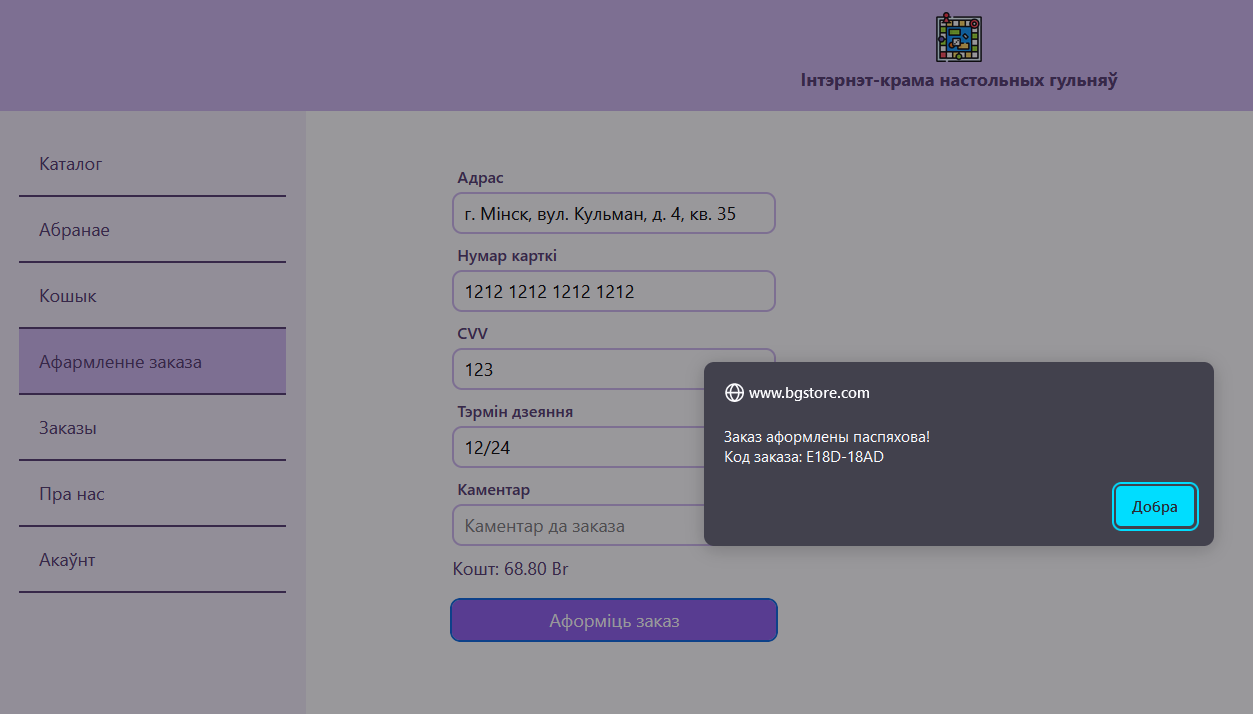


Рисунок 5.13 – Страница оформления заказа

Таким образом, приложение предоставляет различную функциональность для разных пользователей в зависимости от их роли согласно спецификации.

# **Заключение**

В результате выполнения курсового проекта было создано web-приложение для интернет-магазина настольных игр с использованием платформы Node.js, фреймворка Express, языка программирования JavaScript, ORM Sequelize, СУБД Microsoft SQL Server, СУБД Redis.

Были выполнены все функциональные требования, а именно:

* поддерживается возможность регистрации и авторизации в приложении;
* поддерживаются роли администратора, клиента (а также гостя);
* поддерживается поиск в каталоге товаров;
* поддерживается просмотр клиентом истории своих заказов, а администратором – истории заказов всех клиентов;
* поддерживается внесение изменений в данные своего аккаунта;
* поддерживается добавление администратором товаров в каталог, изменение информации о них;
* поддерживается изменение администратором доступности товаров в каталоге;
* поддерживается изменение администратором статусов заказов клиентов;
* поддерживается добавление администратором новых администраторов;
* поддерживается управление клиентом своей корзиной (добавление, исключение товаров);
* поддерживается управление клиентом своей коллекцией избранных товаров (добавление, исключение товаров);
* поддерживается оформление клиентом заказов.

Приложение использует протоколы HTTPS и WSS для передачи данных. Приложение было успешно протестировано. Было составлено руководство пользователя.

Таким образом, все задачи курсового проекта были выполнены с учётом поставленных требований, а цель по разработке web-приложения для интернет-магазина настольных игр была достигнута.

# **Список используемых источников**

1. Yanguren, J. MVC Pattern in Node.js and Express: Old but Gold [Электронный ресурс]. – Режим доступа: https://medium.com/@jonoyanguren/mvc-pattern-in-nodejs-and-express-old-but-gold-46c21bee365a. – Дата доступа: 05.05.2024.
2. Express. Быстрый, гибкий и минималистичный веб-фреймворк для Node.js [Электронный ресурс]. – Режим доступа: https://expressjs.com/ru/. – Дата доступа: 05.05.2024.
3. SkillFactory. Sequelize [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/glossary/sequelize/. – Дата доступа: 05.05.2024.
4. Okta. Beginner’s Guide to JWT [Электронный ресурс]. – Режим доступа: https://developer.okta.com/blog/2020/12/21/beginners-guide-to-jwt. – Дата доступа: 05.05.2024.
5. CASL. Introduction [Электронный ресурс]. – Режим доступа: https://casl.js.org/v6/en/guide/intro. – Дата доступа: 05.05.2024.
6. DigitalOcean. How To Handle File Uploads with Express [Электронный ресурс]. – Режим доступа: https://www.digitalocean.com/community/tutorials/nodejs-uploading-files-multer-express. – Дата доступа: 05.05.2024.

# **Приложение А**

Листинг функции authorize

|  |
| --- |
| export default async function authorize(req, res, next) {  const { rules, can } = casl.AbilityBuilder.extract();  const account = req.account;  switch (account.role) {  case Role.Guest:  break;  case Role.Client:  // Дазваляецца кіраванне звесткамі ўласнага акаўнта, уласным кошыкам, уласнай калекцыяй абранага  can(Action.Manage, [Subject.Account.name, Subject.ClientAccount.name, Subject.Cart.name, Subject.Favourites.name]);  // Дазваляецца стварэнне заказаў  can(Action.Create, Subject.Order.name);  // Дазваляецца прагляд сваіх заказаў  can(Action.View, Subject.Order.name, { orderer\_id: account.id });  // Дазваляецца прагляд спісу сваіх заказаў  can(Action.View, Subject.OrdersList.name);  break;  case Role.Admin:  // Дазваляецца кіраванне звесткамі ўласнага акаўнта, таварамі, заказамі кліентаў  can(Action.Manage, [Subject.Account.name, Subject.Item.name, Subject.Order.name]);  // Дазваляецца стварэнне новых адміністратараў  can(Action.Create, Subject.NewAdminAccount.name);  // Дазваляецца прагляд заказаў, прагляд зместу базы даных (SELECT-запыты)  can(Action.View, [Subject.Order.name, Subject.Db.name]);  // Дазваляецца прагляд спісу заказаў кліентаў  can(Action.View, Subject.OrdersList.name);  break;  default:  break;  }  req.ability = new casl.Ability(rules);  return next();  } |

# **Приложение Б**

Листинг контроллера account

|  |
| --- |
| import \* as accountService from '../services/accountService.js';  import \* as clientService from '../services/clientService.js';  import \* as cookieService from '../services/cookieService.js';  import \* as cryptoService from '../services/cryptoService.js';  import \* as jwtService from '../services/jwtService.js';  import { StatusCodes } from 'http-status-codes';  export async function handleGetAccountInfo(req, res, next) {  const account = await accountService.getFullById(req.account.id);  return res.json(account);  }  export async function handleSignIn(req, res, next) {  const { login, password } = req.body;  const account = await accountService.getByLogin(login);  if (!account || !await accountService.verifyPassword(account.id, password)) {  return res.status(StatusCodes.UNPROCESSABLE\_ENTITY).send('Няправільны лагін ці пароль!');  }  if (!account.is\_active) {  return res.status(StatusCodes.ACCEPTED).send('Аднавіць акаўнт?');  }  const accountUuid = cryptoService.generateUuid();  const accessToken = jwtService.generateAccessToken(account.id, account.role.title, accountUuid);  const refreshToken = jwtService.generateRefreshToken(account.id, account.role.title, accountUuid);  await jwtService.storeRefreshToken(accountUuid, refreshToken);  cookieService.setAccessToken(res, accessToken);  cookieService.setRefreshToken(res, refreshToken);  return res.status(StatusCodes.OK).end('Уваход у акаўнт выкананы паспяхова.');  }  export async function handleSignUp(req, res, next) {  const { login, password, fullname, email, address } = req.body;  if (await accountService.getByLogin(login)) {  return res.status(StatusCodes.CONFLICT).send('Такі лагін ужо ёсць!');  }  const clientAccount = await clientService.createClient(login, password, fullname, email, address);  const accountUuid = cryptoService.generateUuid();  const accessToken = jwtService.generateAccessToken(clientAccount.id, clientAccount.role.title, accountUuid);  const refreshToken = jwtService.generateRefreshToken(clientAccount.id, clientAccount.role.title, accountUuid);  await jwtService.storeRefreshToken(accountUuid, refreshToken);  cookieService.setAccessToken(res, accessToken);  cookieService.setRefreshToken(res, refreshToken);  return res.status(StatusCodes.OK).send('Рэгістрацыя выкананая паспяхова.');  }  export function handleSignOut(req, res, next) {  cookieService.clearAccessToken(res);  cookieService.clearRefreshToken(res);  return res.status(StatusCodes.OK).send('Выхад з акаўнта выкананы паспяхова.');  }  export async function handleUpdateAccountInfo(req, res, next) {  const { login, password, fullname, email } = req.body;  await accountService.updateById(req.account.id, login, password, fullname, email);  return res.status(StatusCodes.OK).send('Звесткі акаўнта абноўленыя паспяхова.');  }  export async function handleUpdateClientAccountInfo(req, res, next) {  const { address } = req.body;  await clientService.updateByAccountId(req.account.id, address);  return res.status(StatusCodes.OK).send('Кліенцкія звесткі абноўленыя паспяхова.');  }  export async function handleCreateNewAdminAccount(req, res, next) {  const { login, password, fullname, email } = req.body;  if (await accountService.getByLogin(login)) {  return res.status(StatusCodes.CONFLICT).send('Такі лагін ужо ёсць!');  }  const admin = await accountService.createAdmin(login, password, fullname, email);  return res.status(StatusCodes.OK).send(`Новы адміністратар ${admin.fullname} дададзены паспяхова.`);  }  export async function handleDeactivateAccount(req, res, next) {  await accountService.deactivateById(req.account.id);  cookieService.clearAccessToken(res);  cookieService.clearRefreshToken(res);  return res.status(StatusCodes.OK).send('Акаўнт дэактывіяваны паспяхова.');  }  export async function handleReactivateAccount(req, res, next) {  const { login, password } = req.body;  const account = await accountService.getByLogin(login);  if (!account || !await accountService.verifyPassword(account.id, password)) {  return res.status(StatusCodes.UNPROCESSABLE\_ENTITY).send('Няправільны лагін ці пароль!');  }  if (!account.is\_active) {  await accountService.activateById(account.id);  }  const accountUuid = cryptoService.generateUuid();  const accessToken = jwtService.generateAccessToken(account.id, account.role.title, accountUuid);  const refreshToken = jwtService.generateRefreshToken(account.id, account.role.title, accountUuid);  await jwtService.storeRefreshToken(accountUuid, refreshToken);  cookieService.setAccessToken(res, accessToken);  cookieService.setRefreshToken(res, refreshToken);  return res.status(StatusCodes.OK).send('Акаўнт адноўлены паспяхова.');  } |

# **Приложение В**

Листинг частичного представления menu.ejs

|  |
| --- |
| <%  const menuItems = [  { text: 'Каталог', href: '/catalog', condition: () => true },  { text: 'Тавар', href: `/item/${locals.item\_id}`, condition: () => pageName === 'item' },  { text: 'Дадаць тавар', href: '/add-item', condition: () => isAdmin },  { text: 'Змяніць тавар', href: `/update-item/${locals.item\_id}`, condition: () => isAdmin && pageName === 'update-item' },  { text: 'Абранае', href: '/favourites', condition: () => isClient },  { text: 'Кошык', href: '/cart', condition: () => isClient },  { text: 'Афармленне заказа', href: '/ordering', condition: () => isClient && pageName === 'ordering' },  { text: 'Заказы', href: '/orders', condition: () => isAdmin || isClient },  { text: 'Пра нас', href: '/about', condition: () => true },  { text: 'Акаўнт', href: '/account', condition: () => true },  { text: 'SQL-запыты', href: '/query', condition: () => isAdmin },  ];  %>  <nav class="menu">  <ul>  <% menuItems.forEach(item => { %>  <% if (item.condition()) { %>  <li><a href="<%= item.href %>"><%= item.text %></a></li>  <% } %>  <% }); %>  </ul>  </nav>  <script>  $(`.menu a[href^="/<%= pageName %>"`).classList.add('active');  </script> |