ГУАП

КАФЕДРА №42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент, кандидат тех. наук |  |  |  | А. В. Бржезовский |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| ПРОГРАММИРОВАНИЕ ИНТЕРФЕЙСА |
| по курсу: МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4121 |  |  |  | А.М.Антонов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

1. **Цель работы**

Изучить методы программирования интерфейсов пользователей.

1. **Задание**

Для варианта информационной системы интернет-магазина спроектировать слой доступа к данным информационной системы с использованием шаблонов проектирования. Слой доступа к данным должен обеспечивать, выборку, вставку, модификацию и удаление записей в таблицах базы данных.

1. **Результаты работы**

**3.1 Обоснование выбора шаблонов проектирования для реализации доступа к БД**

1) Шаблон "Репозиторий" (Repository):

Применение: В коде server.js функции взаимодействия с базой данных разделены на обработчики запросов (например, app.post, app.get) и конкретные SQL-запросы. Это позволяет отделить логику приложения от доступа к данным.

Обоснование: Репозиторий помогает упорядочить работу с базой данных, обеспечивая повторное использование кода и упрощая замену источника данных (например, смену MySQL на другую базу данных).

2)Шаблон "Слой абстракции данных" (Data Access Layer):

Применение: SQL-запросы оформлены в виде отдельных функций в обработчиках, таких как db.query, для извлечения, добавления, изменения и удаления данных.

Обоснование: Слой абстракции упрощает взаимодействие с базой данных, скрывая детали реализации и обеспечивая возможность масштабирования.

3)Шаблон "Фабрика объектов" (Factory):

Применение: Функции, такие как loadProducts в shop.js, создают и добавляют элементы товаров динамически на основе данных из базы.

Обоснование: Фабрика упрощает создание объектов, таких как карточки товаров, абстрагируя детали их формирования.

4)Шаблон "Одиночка" (Singleton):

Применение: Объект подключения к базе данных mysql.createConnection является единственным экземпляром в приложении.

Обоснование: Одиночка гарантирует единое управление подключением к базе данных, избегая лишних операций соединения.

5)Шаблон "Наблюдатель" (Observer):

Применение: В клиентской части (например, shop.js), слушатели событий используются для обновления интерфейса корзины при добавлении или удалении товаров.

Обоснование: Наблюдатель упрощает реакцию интерфейса на изменения данных, синхронизируя визуальное представление с действиями пользователя.

6)Шаблон "Фасад" (Facade):

Применение: Методы API, такие как /api/products или /api/order, предоставляют упрощенный интерфейс для работы с данными на стороне клиента.

Обоснование: Фасад упрощает взаимодействие с серверной частью, скрывая сложные SQL-запросы и структурируя доступ к данным.

**3.2 Описание основных принципов работы выбранных шаблонов**

При входе в интернет-магазин пользователю предлагается войти в аккаунт или зарегистрировать новой. На рисунке 8 представлена страница для регистрации нового пользователя и входа в существующий аккаунт.

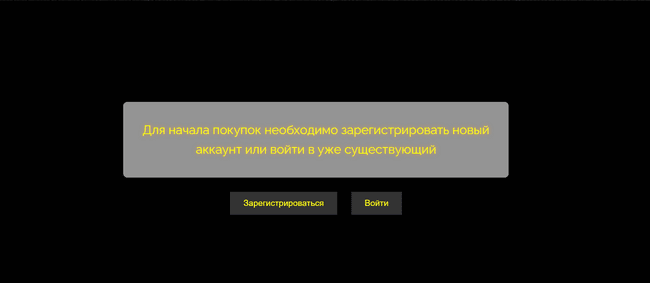


Рисунок 1 – Страница регистрации и авторизации

Пользователь нажимает на кнопку “Зарегистрироваться” и переходит на страницу регистрации (register.html), где представлена форма для ввода следующих данных:

* Имя
* Фамилия
* Электронная почта
* Пароль

После заполнения формы и нажатия кнопки "Зарегистрироваться", данные отправляются на сервер с помощью HTTP POST-запроса (рисунок 2).

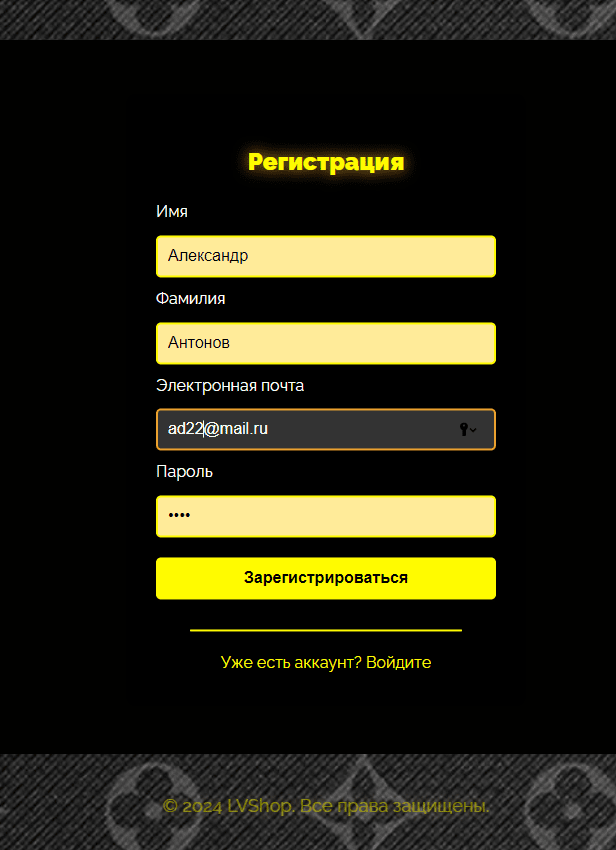


Рисунок 2 - Форма регистрации

Сервер обрабатывает запрос, получая данные из тела (req.body). Проверка существующего пользователя: Сервер выполняет SQL-запрос для проверки, существует ли пользователь с указанным email в базе данных. Если пользователь с таким email уже зарегистрирован, сервер возвращает сообщение об ошибке (рисунок 3).

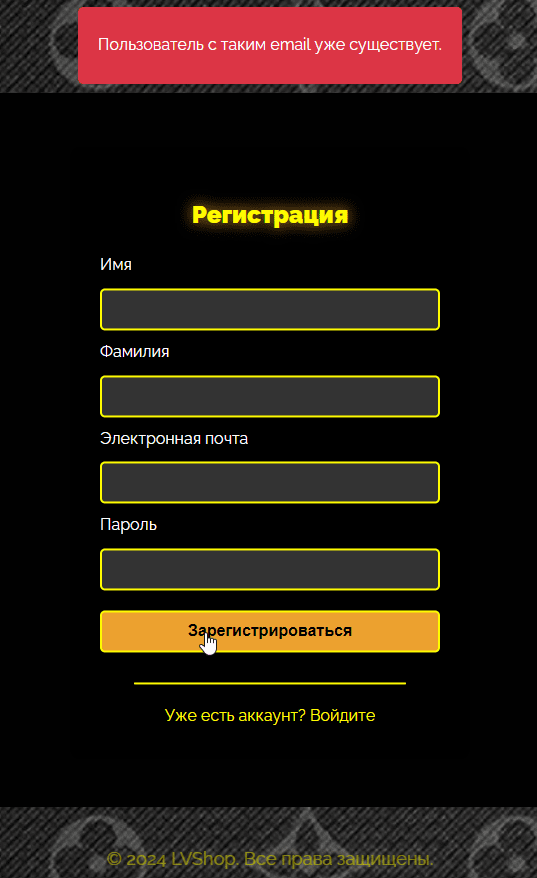


Рисунок 3 – Ошибка при регистрации

Если email уникален, сервер добавляет нового пользователя в базу данных. Если регистрация успешна, сервер перенаправляет пользователя на страницу с авторизацией. Если возникла ошибка (например, проблема с базой данных), сервер возвращает сообщение об ошибке (рисунок 4).

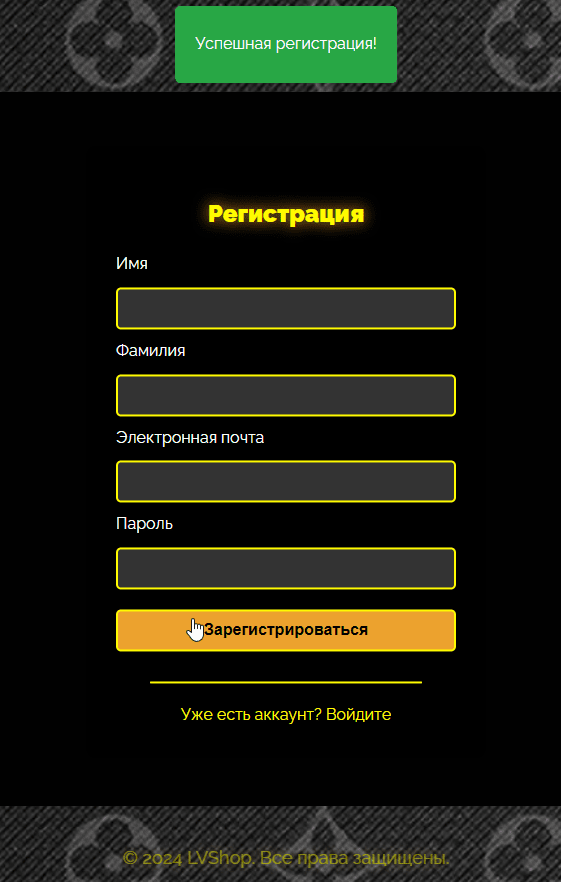


Рисунок 4 – Успешная регистрация

Пользователь нажимает на кнопку “Войти” и переходит на страницу авторизации (login.html) и заполняет форму:

* Электронная почта
* Пароль

После нажатия кнопки "Войти", данные отправляются на сервер с помощью HTTP POST-запроса (рисунок 5).

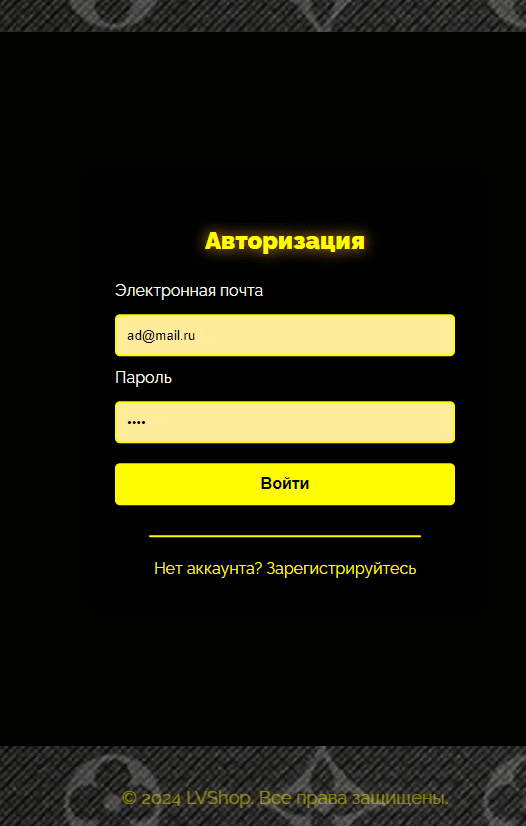


Рисунок 5 - Форма авторизации

Сервер обрабатывает запрос, извлекая email и пароль из тела запроса. Проверка пользователя: Сервер выполняет SQL-запрос для поиска пользователя по email. Если пользователь с таким email не найден, сервер возвращает сообщение "Пользователь не найден" (рисунок 6).

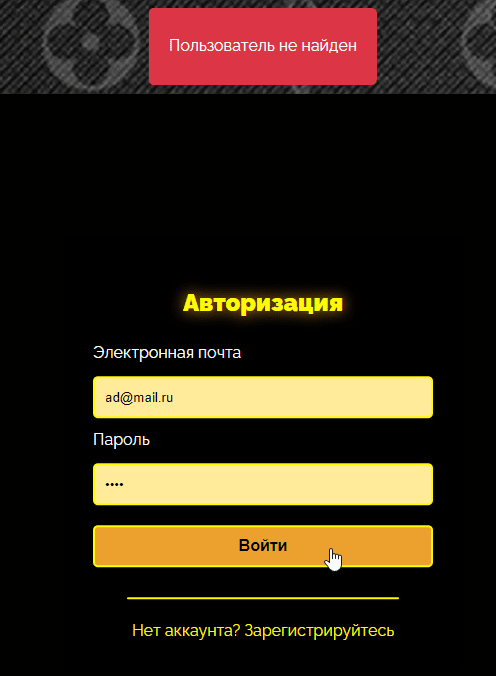


Рисунок 6 – Ошибка при авторизации

Если пользователь найден, сервер сравнивает введенный пароль с паролем из базы данных. Если пароль не совпадает, возвращается сообщение об ошибке (рисунок 7).

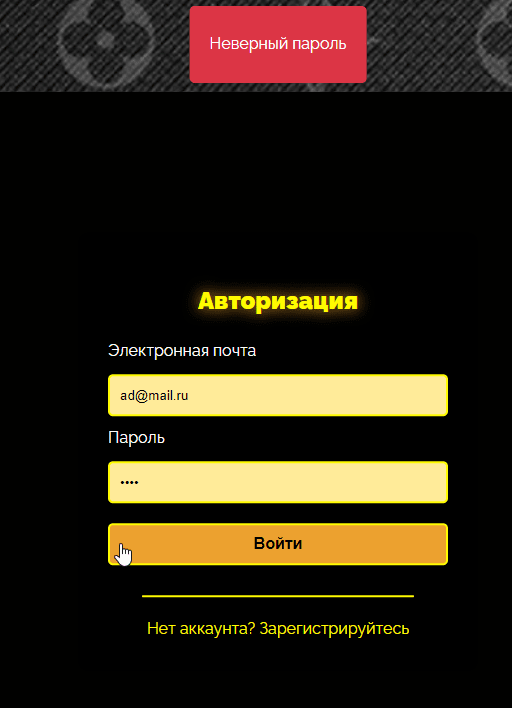


Рисунок 7 – Ошибка не правильного пароля

При успешной авторизации данные пользователя (например, ID, роль) сохраняются в сессии и идёт перенаправление в зависимости от роли:

* Если роль пользователя покупатель, он перенаправляется на страницу выбора пола (/gender).
* Если роль руководитель, он попадает в панель управления руководителя (/pyk).
* Если роль администратор, пользователь перенаправляется в административную панель (/admin).

На рисунке 8 представлена страница с панелью управления администратора. Она служит центральной точкой доступа к различным функциям административного управления. Страница предлагает три основные опции управления, представленные кнопками:

• Управление товарами: Переход на страницу для добавления, редактирования или удаления товаров из каталога.

• Управление клиентами: Переход на страницу для работы с данными клиентов, такими как просмотр и удаление учетных записей.

• Загрузка отчетов: Функция для загрузки и управления аналитическими отчетами.

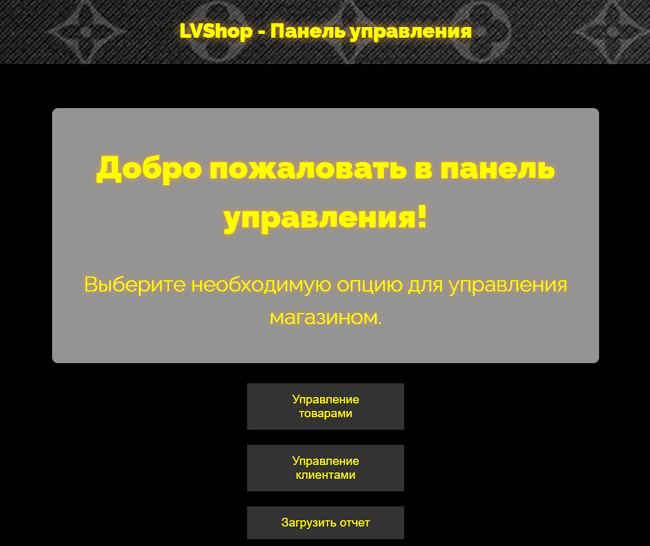


Рисунок 8 - Панель управления администратора

На рисунке 9 представлена страница для добавления, редактирования и удаления товаров из каталога. При выборе пола и категории товара администратору отображается таблица товаров по заданным фильтрам, для последующего редактирования.

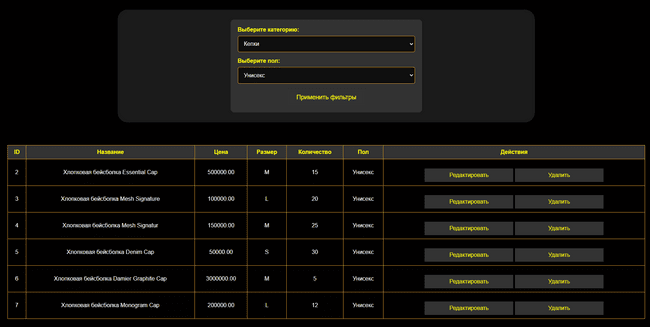


Рисунок 9 - Функция управления товарами

Под таблицей с товарами расположена форма добавления нового товара (рисунок 10). После нажатия кнопки "Добавить", данные отправляются на сервер через POST-запрос и новый товар отобразиться в списке товаров. На рисунках 11-12 представлен процесс добавления нового товара.

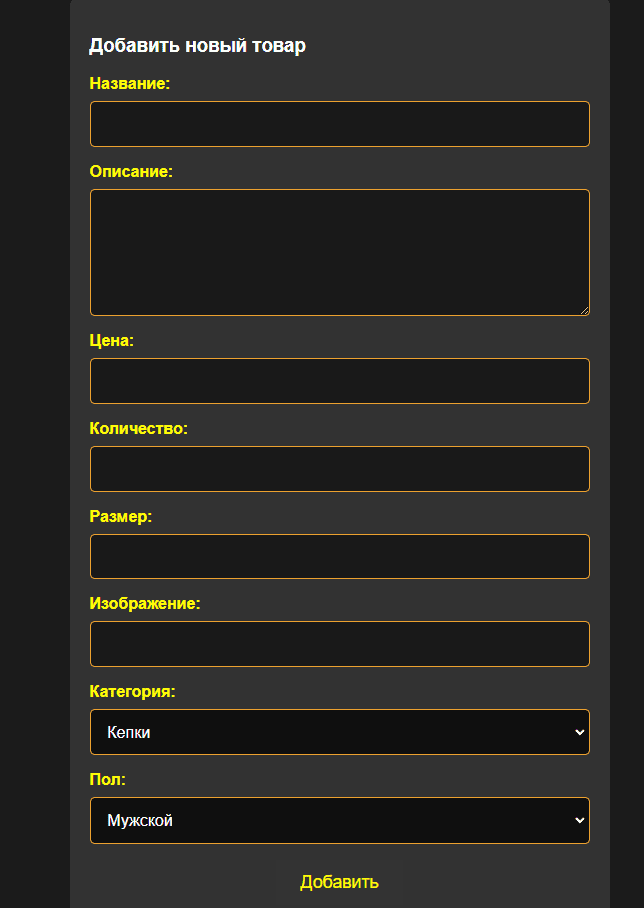


Рисунок 10 - Форма добавления нового товара

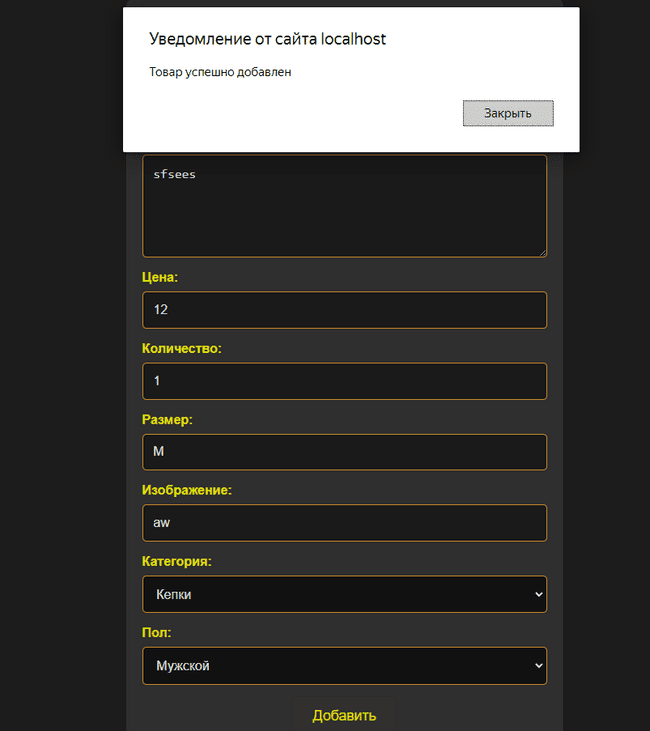


Рисунок 11 – Заполнение формы добавления товаров

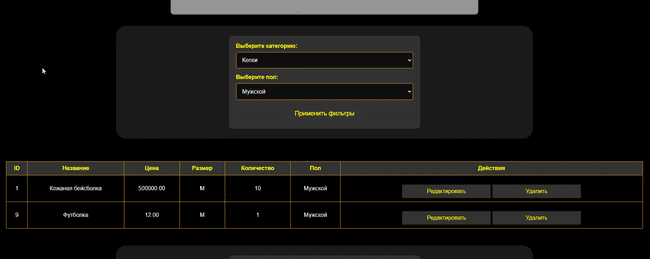


Рисунок 12 – Отображение нового товара в списках товаров

На рисунках 13-15 представлена функция редактирования товаров. Администратор выбирает товар из таблицы и нажимает кнопку "Редактировать". В модальном окне заполняются поля с текущими данными товара, которые можно изменить. При нажатии на кнопку “Сохранить изменения” измененные данные отправляются на сервер через POST-запрос.

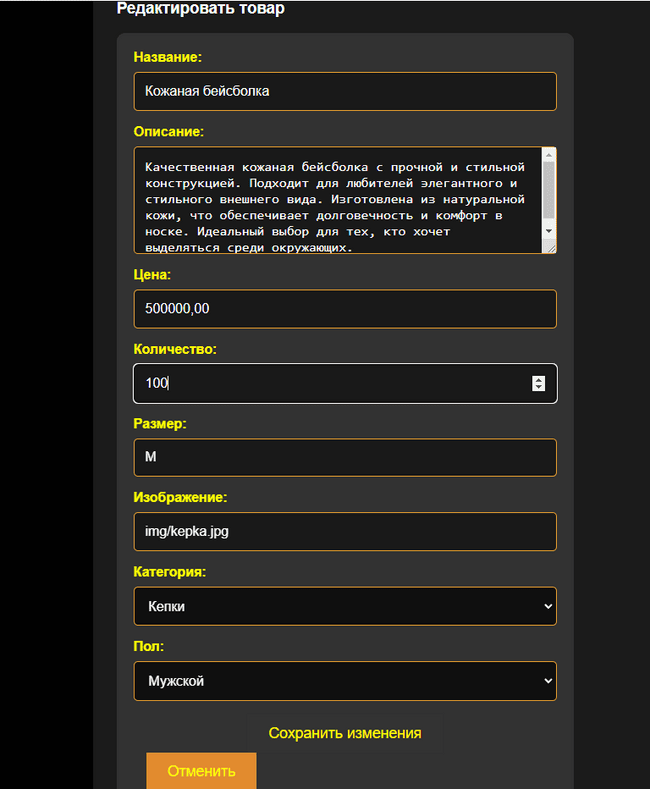


Рисунок 13 – Редактирование товара

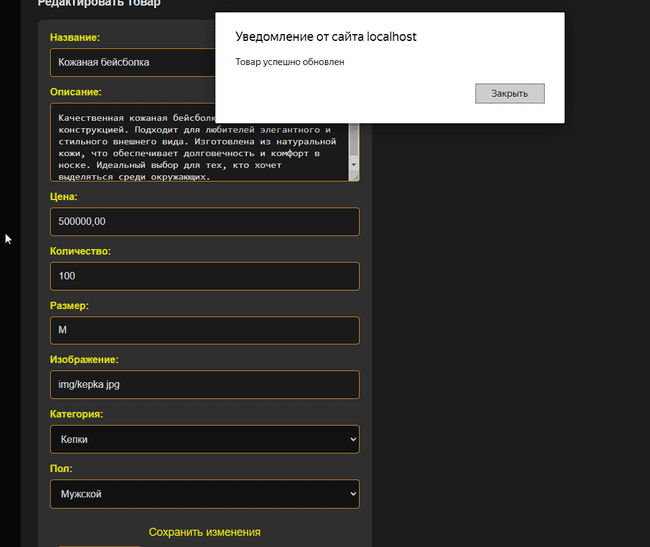


Рисунок 14 – Сообщение о успешном уведомлении

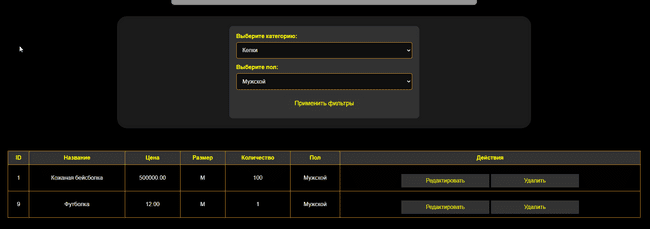


Рисунок 15 – Обновление информации о товаре

На рисунках 16-17 представлена функция удаления товара. В таблице товаров администратор нажимает кнопку "Удалить" рядом с конкретным товаром.

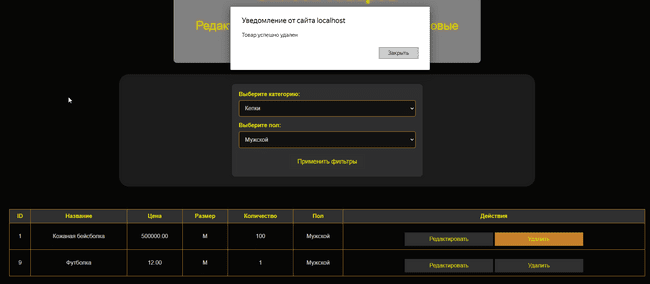


Рисунок 16 – Сообщение о удалении товара

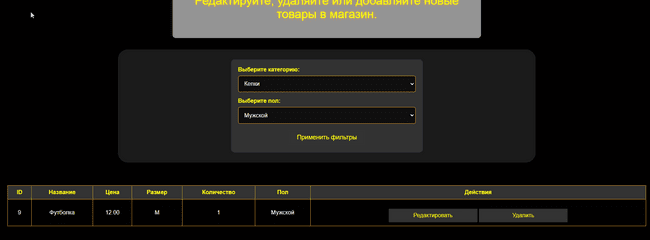


Рисунок 17 – Удаление товара

На рисунках 18-20 представлена функция администратора просмотра и удаления учетных записей пользователей. В таблице пользователей отображается список всех пользователей, загруженных с сервера. Администратор нажимает кнопку "Удалить" рядом с конкретным пользователем. Появляется окно подтверждения удаления.

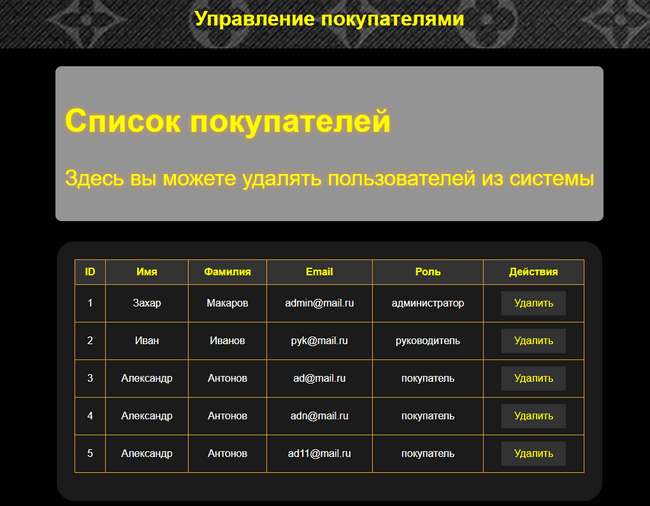


Рисунок 18 – Список пользователей

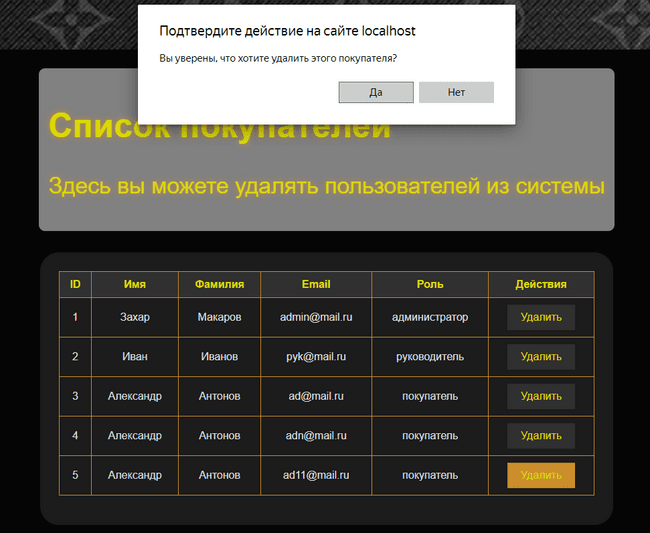


Рисунок 19 – Сообщение о удалении пользователя

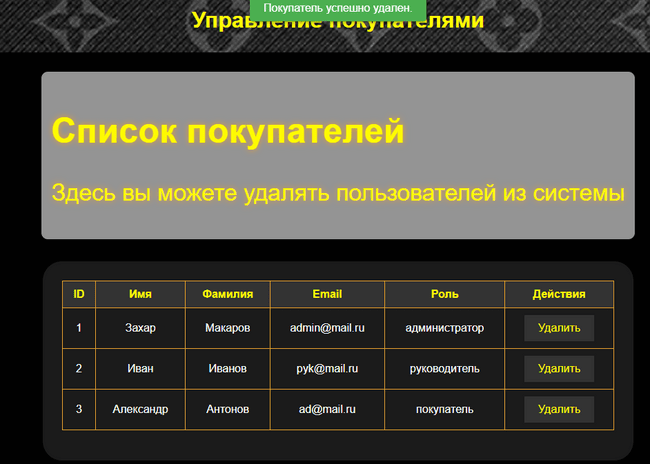


Рисунок 20 – Удаление пользователя

На рисунках 20-23 представлена функция администратора загрузки отчетов для анализа продаж руководителем. Администратор выбирает файл (Word или Excel) и нажимает кнопку "Загрузить". Файл отправляется на сервер с помощью multer. После загрузки отображается список загруженных отчетов с их статусами (например, "Принято", "Не принято"). Также администратор может удалить выбранный отчёт.

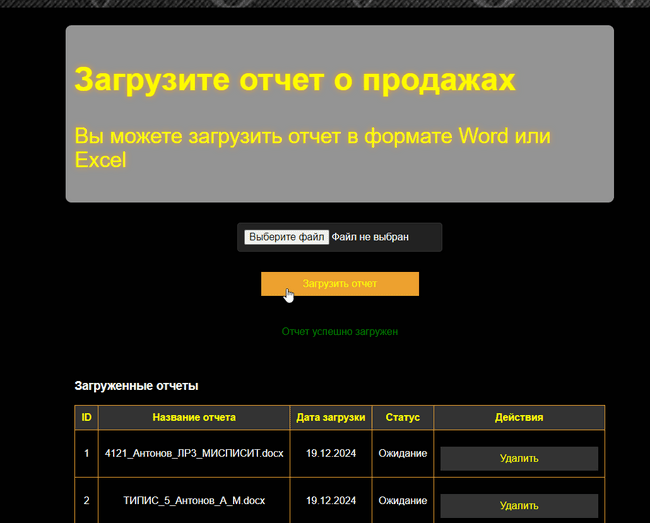


Рисунок 20 – Сообщение об успешной загрузки

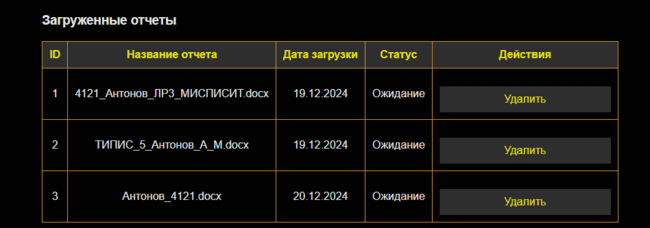


Рисунок 21 – Список загруженных отчётов

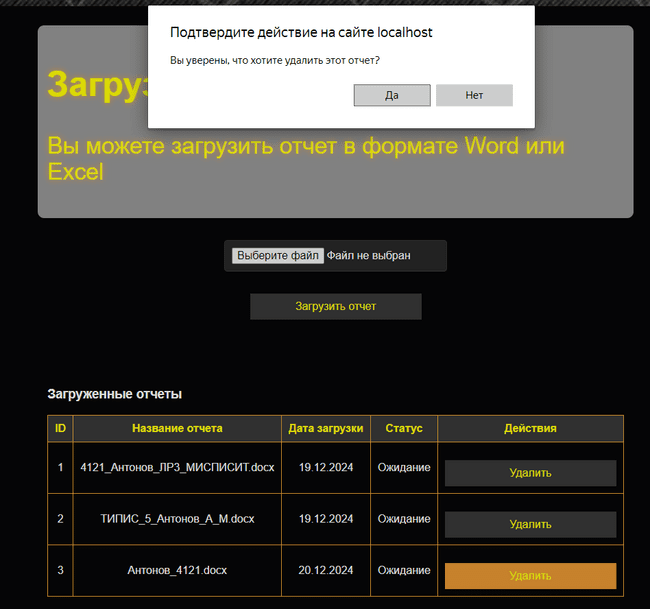


Рисунок 22 – Сообщение об удалении отчёта

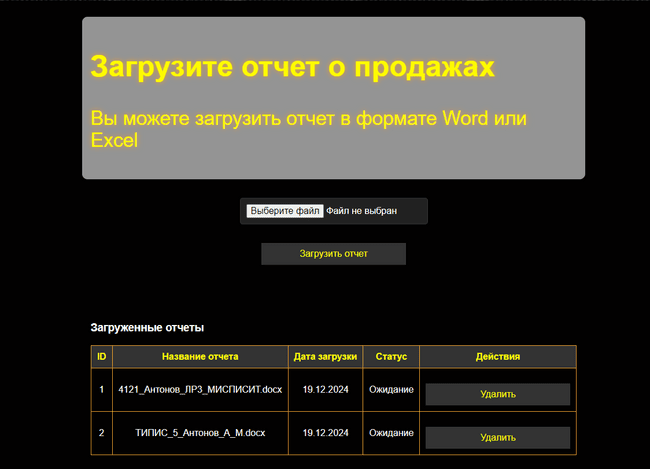


Рисунок 23 – Список загруженных отчётов

На рисунке 24 представлена страница панели управления руководителя. Эта страница предназначена для руководителя магазина и предоставляет доступ к ключевым функциям для анализа работы магазина. На странице представлены две основные опции для управления и анализа данных магазина: просмотр отчетов и просмотр статистики.

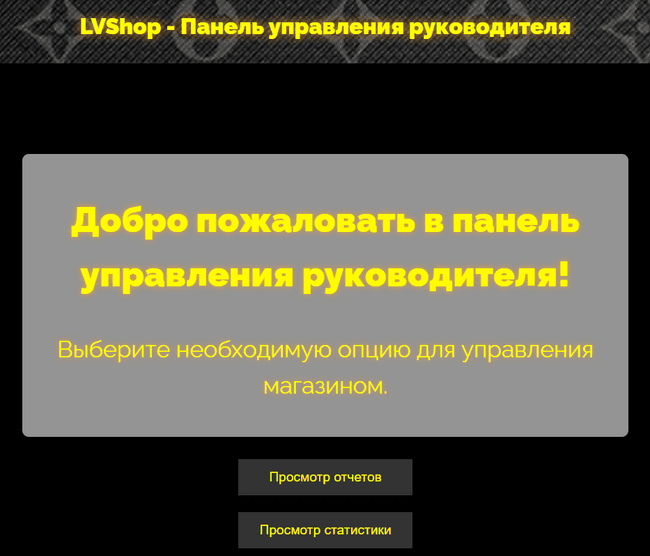


Рисунок 24 - Панель управления руководителя

На рисунках 25-28 представлена функия просмотра отчётов, загруженных администратором. Руководитель переходит на страницу просмотра отчетов через панель управления руководителя (pyk.html) с кнопкой "Просмотр отчетов". На странице view-reports.html отображается таблица с отчетами. Руководитель может скачать любой отчет, нажав кнопку "Скачать" в таблице. Руководитель может изменить статус отчета, выбрав "Принято" или "Не принято".

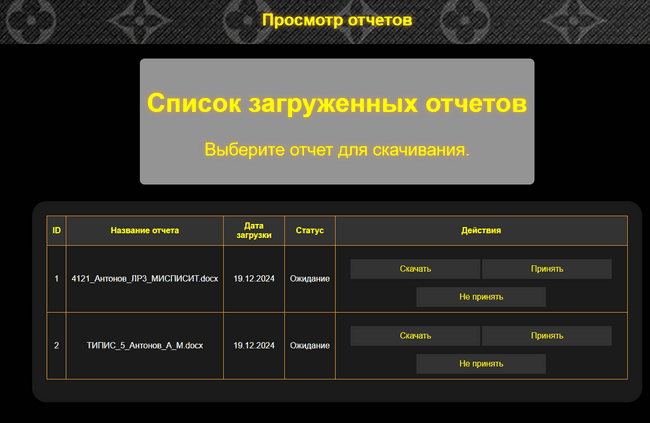


Рисунок 25 – Список загруженных отчётов

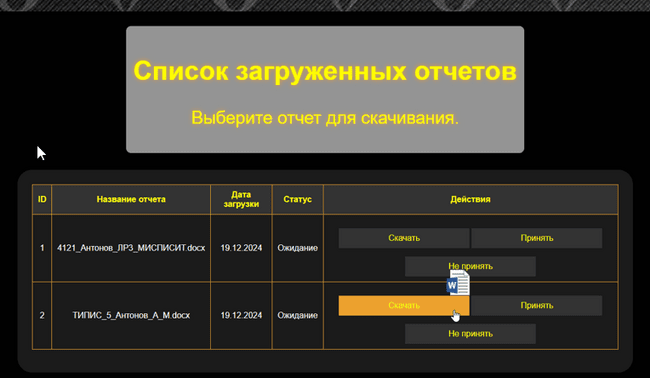


Рисунок 26 – Скачивание отчета

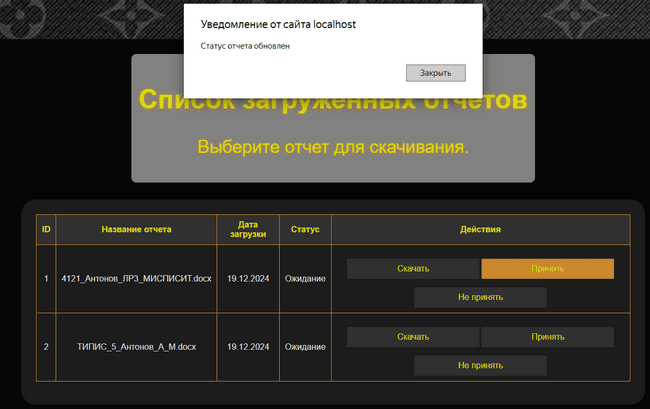


Рисунок 27 – Основление статуса отчёта

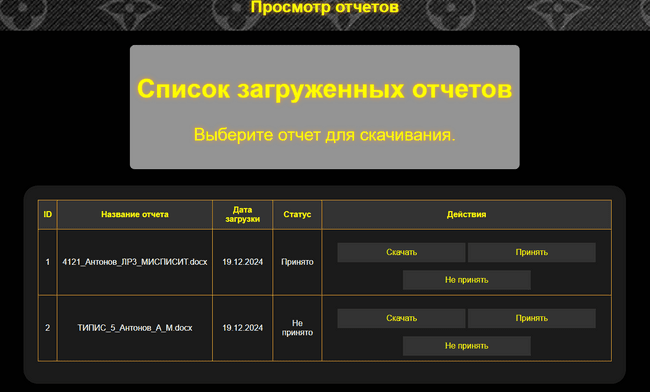


Рисунок 28 - Основление статуса отчёта

На рисунках 29-20 представлены страницы со статистикой продаж.

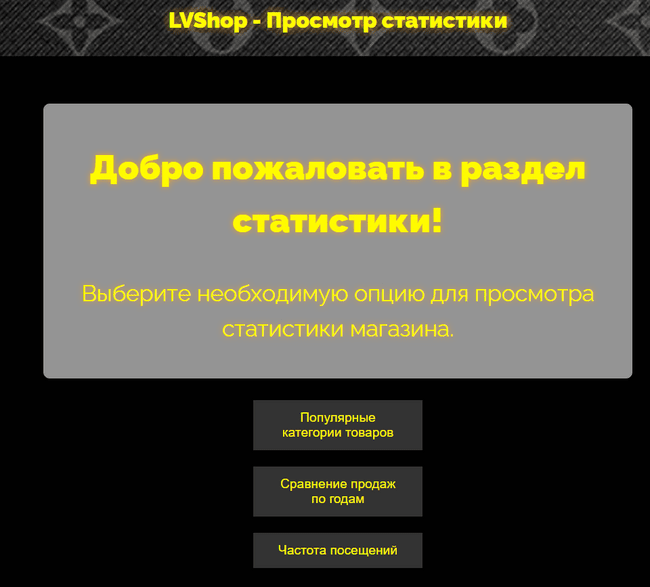


Рисунок 29 - Выбор опции

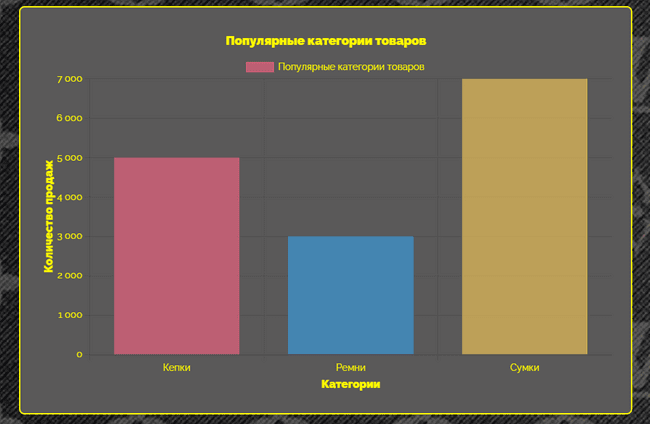


Рисунок 30 - Диаграмма «Популярные категории товаров»

После авторизации покупатель попадает на страницу gender.html, где может выбрать категорию товаров (рисунок 31):

* Мужской
* Женский



Рисунок 31 – Выбор пола

После выбора пола покупатель пользователь попадает на главную страницу сайта. На рисунке 32 представлена главная страница интернет-магазина с навигационной панелью, содержащей контактную информацию, каталог товаров и сведения о магазине.



Рисунок 32 - Главная страница

После выбора определённой категории товара пользователь попадает на страницу товаров определённой категории. На рисунке 33 представлена страница каталога одной из категорий товаров.

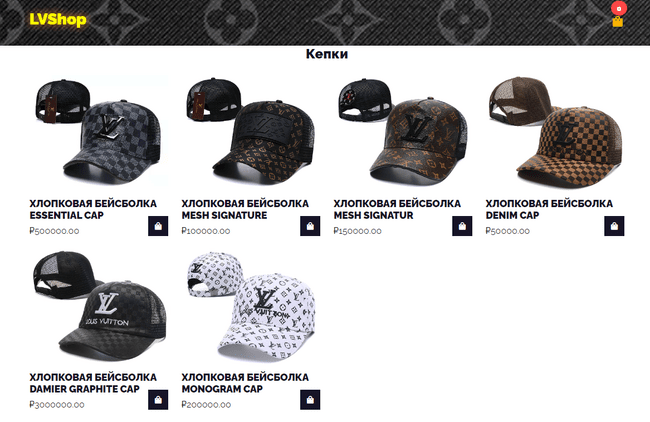


Рисунок 33 – Каталог

При клике на иконку корзины под товаром открывется карточка товара с описанием, фото, ценой, возможностью выбора размера товара и возможностью добавть его в корзину (рисунок 34).

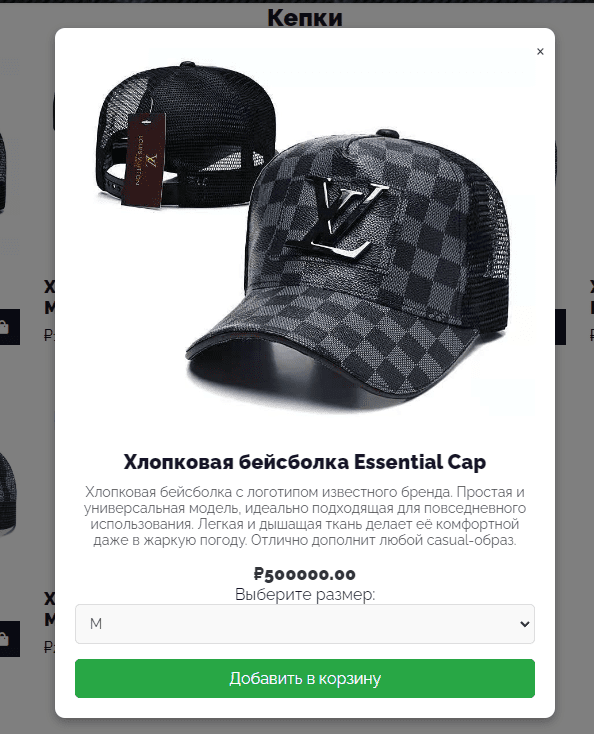


Рисунок 34 - Карточка товара

На рисунке 35 представлена корзина с выбранными товарами. В корзине реализованы функции удаления товаров, изменения их количества, а также автоматический подсчёт общей стоимости всего заказа.

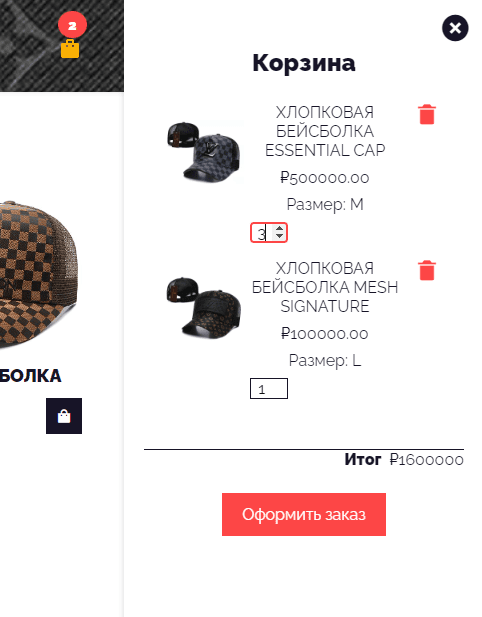


Рисунок 35 – Добавление товаров в корзину

На рисунке 36 представлена функция оформления заказа в магазине.

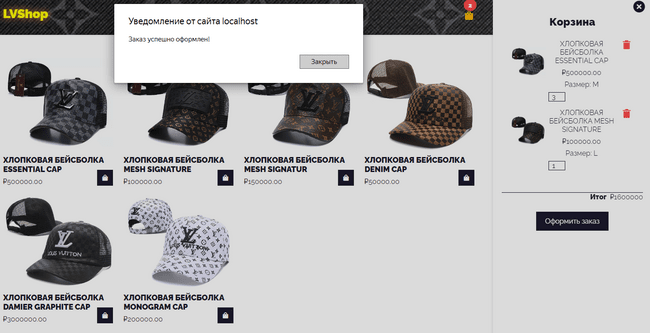


Рисунок 37 – Функция оформления заказа

**3.3 Тексты программ с описаниями**

Server.js (здесь показан код только серверной части приложения):

const express = require('express')

const mysql = require('mysql2')

const bodyParser = require('body-parser')

const session = require('express-session')

const multer = require('multer')

const path = require('path')

const fs = require('fs')

const iconv = require('iconv-lite')

const app = express()

const port = 3000

app.use(bodyParser.urlencoded({ extended: false }))

app.use(bodyParser.json())

// Конфигурация сессий

app.use(

    session({

        secret: 'your\_secret\_key',

        resave: false,

        saveUninitialized: true,

    })

)

// Создание подключения к базе данных

const db = mysql.createConnection({

    host: 'localhost',

    user: 'root',

    password: '1111',

    database: 'интернет\_магазин',

    port: 3307,

    charset: 'utf8mb4',

})

// Подключение к базе данных

db.connect(err => {

    if (err) throw err

    console.log('Подключено к базе данных')

})

app.use(express.static(\_\_dirname + '/public'))

app.get('/', (req, res) => {

    res.sendFile(\_\_dirname + '/public/index.html') // Путь к файлу главной страницы

})

// Страница регистрации

app.get('/register', (req, res) => {

    const message = req.query.message || ''

    res.sendFile(\_\_dirname + '/public/register.html', { message }) // Путь к файлу регистрации

})

app.post('/register', (req, res) => {

    const { name, surname, email, password } = req.body

    db.query(

        'SELECT \* FROM пользователи WHERE email = ?',

        [email],

        (err, results) => {

            if (results.length > 0) {

                return res.redirect(

                    '/register?message=Пользователь с таким email уже существует.&error=true'

                )

            }

            db.query(

                'INSERT INTO пользователи (имя, фамилия, email, пароль) VALUES (?, ?, ?, ?)',

                [name, surname, email, password],

                err => {

                    if (err) {

                        console.error(err)

                        return res.redirect(

                            '/register?message=Ошибка при регистрации.&error=true'

                        )

                    }

                    res.redirect('/register?message=Успешная регистрация!&success=true')

                }

            )

        }

    )

})

// Страница логина

app.get('/login', (req, res) => {

    const message = req.query.message || ''

    res.sendFile(\_\_dirname + '/public/login.html', { message }) // Путь к файлу логина

})

app.post('/login', (req, res) => {

    const { email, password } = req.body

    db.query(

        'SELECT \* FROM пользователи WHERE email = ?',

        [email],

        (err, userResults) => {

            if (err || userResults.length === 0) {

                return res.redirect('/login?message=Пользователь не найден&error=true')

            }

            const user = userResults[0]

            if (user.пароль !== password) {

                return res.redirect('/login?message=Неверный пароль&error=true')

            }

            req.session.user = user

            switch (user.роль) {

                case 'покупатель':

                    res.redirect('/gender')

                    break

                case 'руководитель':

                    res.redirect('/pyk')

                    break

                case 'администратор':

                    res.redirect('/admin')

                    break

                default:

                    res.redirect('/')

                    break

            }

        }

    )

})

// Страница выбора пола (для покупателя)

app.get('/gender', (req, res) => {

    if (!req.session.user || req.session.user.роль !== 'покупатель') {

        return res.redirect('/login') // Если пользователь не авторизован

    }

    res.sendFile(\_\_dirname + '/public/gender.html') // Путь к файлу выбора пола

})

// Панель управления руководителя

app.get('/pyk', (req, res) => {

    if (!req.session.user || req.session.user.роль !== 'руководитель') {

        return res.redirect('/login') // Если не авторизован

    }

    res.sendFile(\_\_dirname + '/public/pyk.html') // Панель управления руководителя

})

// Панель управления администратора

app.get('/admin', (req, res) => {

    if (!req.session.user || req.session.user.роль !== 'администратор') {

        return res.redirect('/login') // Если не авторизован

    }

    res.sendFile(\_\_dirname + '/public/admin.html') // Панель управления администратора

})

// Получение списка клиентов

app.get('/customers', (req, res) => {

    db.query('SELECT id, имя, фамилия, email, роль FROM пользователи', (err, results) => {

        if (err) {

            console.error(err);

            return res.status(500).json({ success: false, message: 'Ошибка получения данных.' });

        }

        res.json(results);

    });

});

// Удаление клиента

app.delete('/customers/:id', (req, res) => {

    const customerId = req.params.id;

    db.query('DELETE FROM пользователи WHERE id = ?', [customerId], (err, result) => {

        if (err) {

            console.error(err);

            return res.status(500).json({ success: false, message: 'Ошибка удаления клиента.' });

        }

        if (result.affectedRows > 0) {

            res.json({ success: true, message: 'Клиент успешно удален.' });

        } else {

            res.json({ success: false, message: 'Клиент не найден.' });

        }

    });

});

// Настройка хранения файлов для загрузки отчетов

const storage = multer.diskStorage({

    destination: (req, file, cb) => {

        cb(null, 'uploads/')

    },

    filename: (req, file, cb) => {

        const originalName = iconv.encode(file.originalname, 'utf-8') // Преобразование в нужную кодировку

        cb(null, Date.now() + path.extname(file.originalname))

    },

})

const upload = multer({ storage: storage });

// Страница загрузки отчета

app.get('/upload-report', (req, res) => {

  if (!req.session.user || req.session.user.роль !== 'администратор') {

    return res.redirect('/login');

  }

  res.sendFile(\_\_dirname + '/public/upload-report.html');

});

// Обработка загрузки отчета

app.post('/upload-report', upload.single('report'), (req, res) => {

    if (!req.file) {

        return res.redirect(

            '/upload-report?message=Ошибка загрузки файла&error=true'

        )

    }

    const { originalname, path: filePath } = req.file

    const userId = req.session.user.id

    // Кодировка имени файла в базу данных

    const encodedFileName = iconv.decode(originalname, 'utf-8')

    db.query(

        'INSERT INTO отчёты (название, файл, создан\_пользователем) VALUES (?, ?, ?)',

        [encodedFileName, filePath, userId],

        err => {

            if (err) {

                console.error(err)

                return res.redirect(

                    '/upload-report?message=Ошибка при загрузке отчета&error=true'

                )

            }

            res.redirect('/upload-report?message=Отчет успешно загружен&success=true')

        }

    )

})

// Страница просмотра отчетов для руководителя

app.get('/view-reports', (req, res) => {

  if (!req.session.user || req.session.user.роль !== 'руководитель') {

    return res.redirect('/login');

  }

  db.query('SELECT \* FROM отчёты', (err, results) => {

    if (err) {

      console.error(err);

      return res.status(500).send('Ошибка при получении отчетов');

    }

    res.sendFile(\_\_dirname + '/public/view-reports.html');

  });

});

// Получить список отчетов для руководителя и администратора

app.get('/api/reports', (req, res) => {

  if (!req.session.user || !['руководитель', 'администратор'].includes(req.session.user.роль)) {

    return res.redirect('/login');

  }

  db.query('SELECT \* FROM отчёты', (err, results) => {

    if (err) {

      console.error(err);

      return res.status(500).json({ success: false, message: 'Ошибка получения отчетов' });

    }

    res.json(results);

  });

});

// Скачать отчет

app.get('/download-report/:id', (req, res) => {

    const reportId = req.params.id

    db.query('SELECT \* FROM отчёты WHERE id = ?', [reportId], (err, results) => {

        if (err) {

            console.error(err)

            return res.status(500).send('Ошибка при скачивании отчета')

        }

        const report = results[0]

        if (!report) {

            return res.status(404).send('Отчет не найден')

        }

        // Убедитесь, что имя файла с русскими символами в правильной кодировке

        const decodedFileName = iconv.decode(Buffer.from(report.файл), 'utf-8')

        res.download(decodedFileName)

    })

})

// Обновление статуса отчета

app.post('/update-report-status', (req, res) => {

  const { reportId, status } = req.body;

  db.query(

    'UPDATE отчёты SET статус = ? WHERE id = ?',

    [status, reportId],

    (err) => {

      if (err) {

        console.error(err);

        return res.status(500).send('Ошибка при обновлении статуса');

      }

      res.json({ success: true, message: 'Статус отчета обновлен' });

    }

  );

});

app.delete('/api/reports/:id', (req, res) => {

    const reportId = req.params.id

    // Проверяем статус отчета

    db.query(

        'SELECT статус FROM отчёты WHERE id = ?',

        [reportId],

        (err, results) => {

            if (err) {

                console.error(err)

                return res

                    .status(500)

                    .json({ success: false, message: 'Ошибка проверки статуса отчета.' })

            }

            if (results.length === 0) {

                return res

                    .status(404)

                    .json({ success: false, message: 'Отчет не найден.' })

            }

            const reportStatus = results[0].статус

            if (reportStatus === 'принято') {

                return res

                    .status(403)

                    .json({ success: false, message: 'Нельзя удалить принятый отчет.' })

            }

            // Удаление отчета

            db.query('DELETE FROM отчёты WHERE id = ?', [reportId], (err, result) => {

                if (err) {

                    console.error(err)

                    return res

                        .status(500)

                        .json({ success: false, message: 'Ошибка удаления отчета.' })

                }

                if (result.affectedRows > 0) {

                    res.json({ success: true, message: 'Отчет успешно удален.' })

                } else {

                    res.json({ success: false, message: 'Отчет не найден.' })

                }

            })

        }

    )

})

// Получение списка товаров

app.get('/api/products', (req, res) => {

    db.query('SELECT \* FROM товары', (err, results) => {

        if (err) {

            console.error(err);

            return res.status(500).json({ success: false, message: 'Ошибка получения данных товаров.' });

        }

        res.json(results);

    });

});

// Получение всех категорий

app.get('/categories', (req, res) => {

    db.query('SELECT \* FROM категории', (err, results) => {

        if (err) {

            console.error(err);

            return res.status(500).send('Ошибка при получении категорий');

        }

        res.json(results);

    });

});

// Получение товаров по фильтрам категории и пола

app.post('/products', (req, res) => {

    const { category, gender } = req.body;

    let query = 'SELECT \* FROM товары WHERE id\_категории = ? AND пол = ?';

    db.query(query, [category, gender], (err, results) => {

        if (err) {

            console.error(err);

            return res.status(500).send('Ошибка при получении товаров');

        }

        res.json(results);

    });

});

// Редактирование товара

app.post('/edit-product', (req, res) => {

    const {

        id,

        name,

        description,

        price,

        quantity,

        size,

        image,

        category,

        gender,

    } = req.body

    const query = `

        UPDATE товары

        SET название = ?, описание = ?, цена = ?, количество = ?, размер = ?, изображение = ?, id\_категории = ?, пол = ?

        WHERE id = ?

    `

    db.query(

        query,

        [name, description, price, quantity, size, image, category, gender, id],

        (err, results) => {

            if (err) {

                console.error(err)

                return res.status(500).send('Ошибка при обновлении товара')

            }

            res.json({ message: 'Товар успешно обновлен' })

        }

    )

})

// Добавление нового товара

app.post('/add-product', (req, res) => {

    const { name, description, price, quantity, size, image, category, gender } = req.body;

    const query = 'INSERT INTO товары (название, описание, цена, количество, размер, изображение, id\_категории, пол) VALUES (?, ?, ?, ?, ?, ?, ?, ?)';

    db.query(query, [name, description, price, quantity, size, image, category, gender], (err, results) => {

        if (err) {

            console.error(err);

            return res.status(500).send('Ошибка при добавлении товара');

        }

        res.json({ message: 'Товар успешно добавлен' });

    });

});

// Удаление товара

app.post('/delete-product', (req, res) => {

    const { id } = req.body;

    const query = 'DELETE FROM товары WHERE id = ?';

    db.query(query, [id], (err, results) => {

        if (err) {

            console.error(err);

            return res.status(500).send('Ошибка при удалении товара');

        }

        res.json({ message: 'Товар успешно удален' });

    });

});

app.get('/products/:id', (req, res) => {

    const productId = req.params.id

    const query = 'SELECT \* FROM товары WHERE id = ?'

    db.query(query, [productId], (err, results) => {

        if (err) {

            console.error(err)

            return res.status(500).send('Ошибка при получении данных товара')

        }

        if (results.length === 0) {

            return res.status(404).send('Товар не найден')

        }

        res.json(results[0])

    })

})

// Получение списка товаров

app.get('/api/products', (req, res) => {

    db.query('SELECT \* FROM товары', (err, results) => {

        if (err) {

            console.error(err);

            return res.status(500).json({ success: false, message: 'Ошибка получения данных товаров.' });

        }

        res.json(results.map(product => ({

            id: product.id,

            название: product.название,

            описание: product.описание,

            цена: product.цена,

            количество: product.количество,

            размер: product.размер,

            изображение: product.изображение,

            id\_категории: product.id\_категории,

            пол: product.пол,

        })));

    });

});

app.post('/api/order', (req, res) => {

    const { cartItems, address, deliveryMethod } = req.body

    if (!req.session.user) {

        return res

            .status(401)

            .json({ success: false, message: 'Пользователь не авторизован.' })

    }

    const userId = req.session.user.id

    if (!cartItems || cartItems.length === 0) {

        return res.status(400).json({ success: false, message: 'Корзина пуста.' })

    }

    let totalAmount = 0

    cartItems.forEach(item => {

        totalAmount += item.price \* item.quantity

    })

    db.query(

        'INSERT INTO заказы (id\_пользователя, сумма) VALUES (?, ?)',

        [userId, totalAmount],

        (err, orderResult) => {

            if (err) {

                console.error(err)

                return res

                    .status(500)

                    .json({ success: false, message: 'Ошибка создания заказа.' })

            }

            const orderId = orderResult.insertId

            const orderDetails = cartItems.map(item => [

                orderId,

                item.id, // Используем id товара

                item.quantity,

                item.price,

            ])

            db.query(

                'INSERT INTO детали\_заказа (id\_заказа, id\_товара, количество, цена) VALUES ?',

                [orderDetails],

                err => {

                    if (err) {

                        console.error(err)

                        return res.status(500).json({

                            success: false,

                            message: 'Ошибка добавления товаров заказа.',

                        })

                    }

                    res.json({ success: true, message: 'Заказ успешно создан.' })

                }

            )

        }

    )

})

// Запуск сервера

app.listen(port, () => {

    console.log(`Сервер запущен на http://localhost:${port}`)

})