**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Саратовский государственный технический университет**

**имени Гагарина Ю.А.»**

Институт прикладных информационных технологий и коммуникаций

Направление «Информационные системы и технологии»

Кафедра «Прикладные информационные технологии»

**КОНТРОЛЬНАЯ РАБОТА**

по дисциплине «Нереляционные базы данных»

Выполнил студент группы

б1-ИФСТ-21

очной формы обучения

Плотников Владислав Александрович

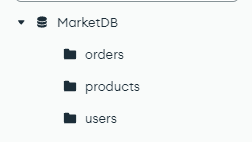
Номер зачетной книжки: 220814

Проверил: старший преподаватель кафедры ПИТ

Левченко Д.С.

Саратов, 2024

Задание 1



# Создание базы данных MarketDB

use MarketDB;

# Создание коллекции orders с полями buyerName, buyerEmail, selectedProducts

db.createCollection("orders");

# Создание коллекции users с полями email, name, phone

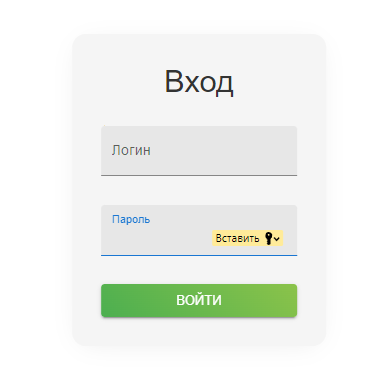
db.createCollection("users");

# Создание коллекции products с полями Name, Price, Description

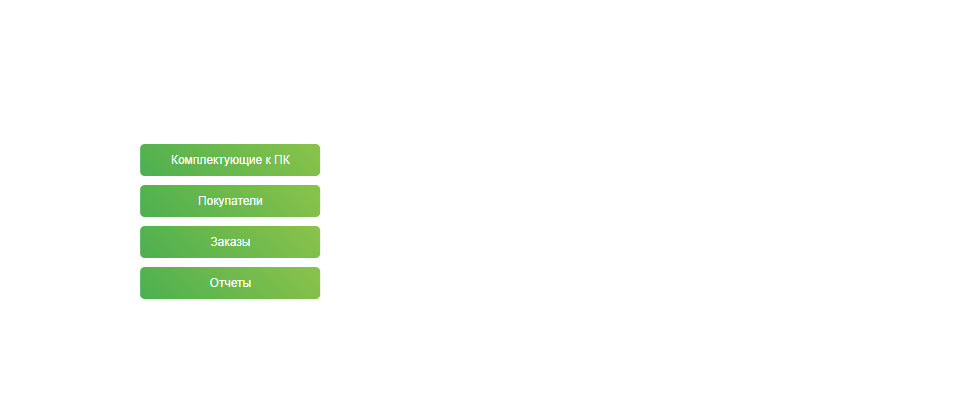
db.createCollection("products");

Задание 2

Форма для входа



Страница админа:



Страница эдитора:

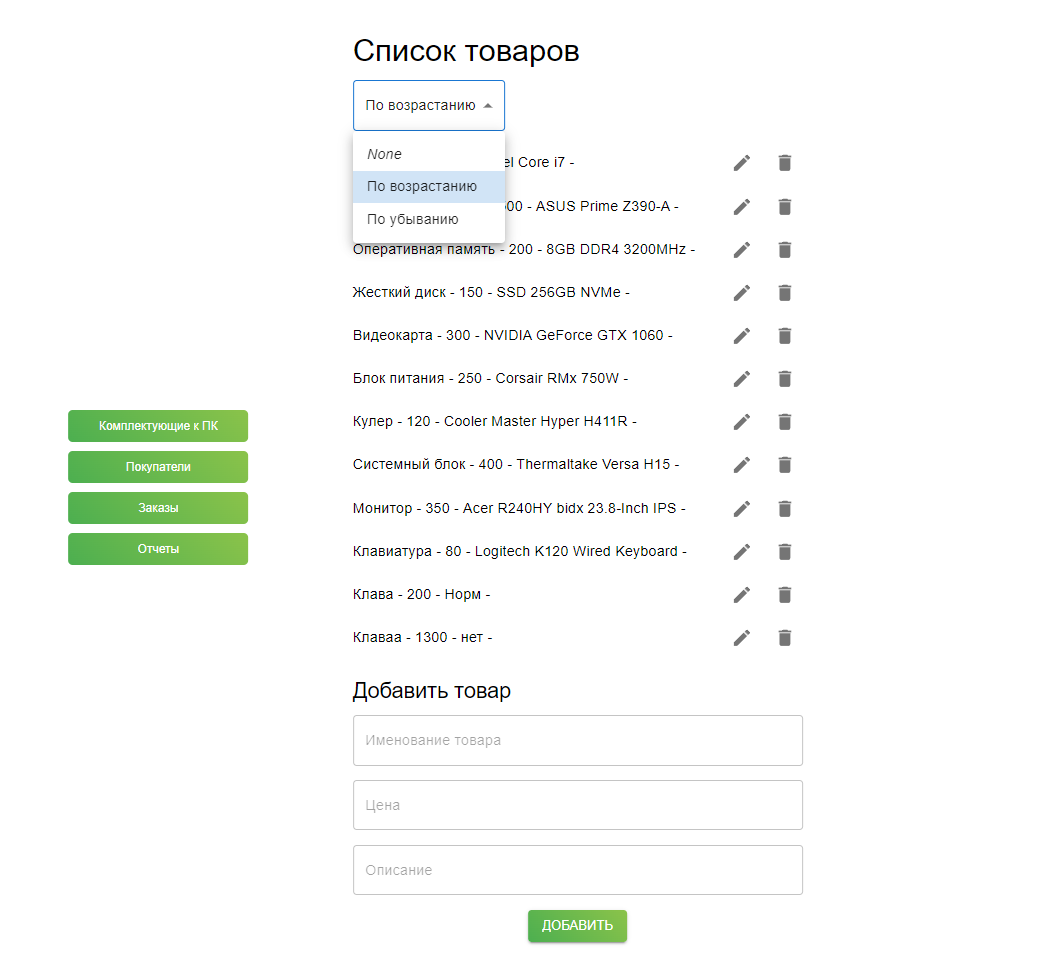


Страница пользователя:

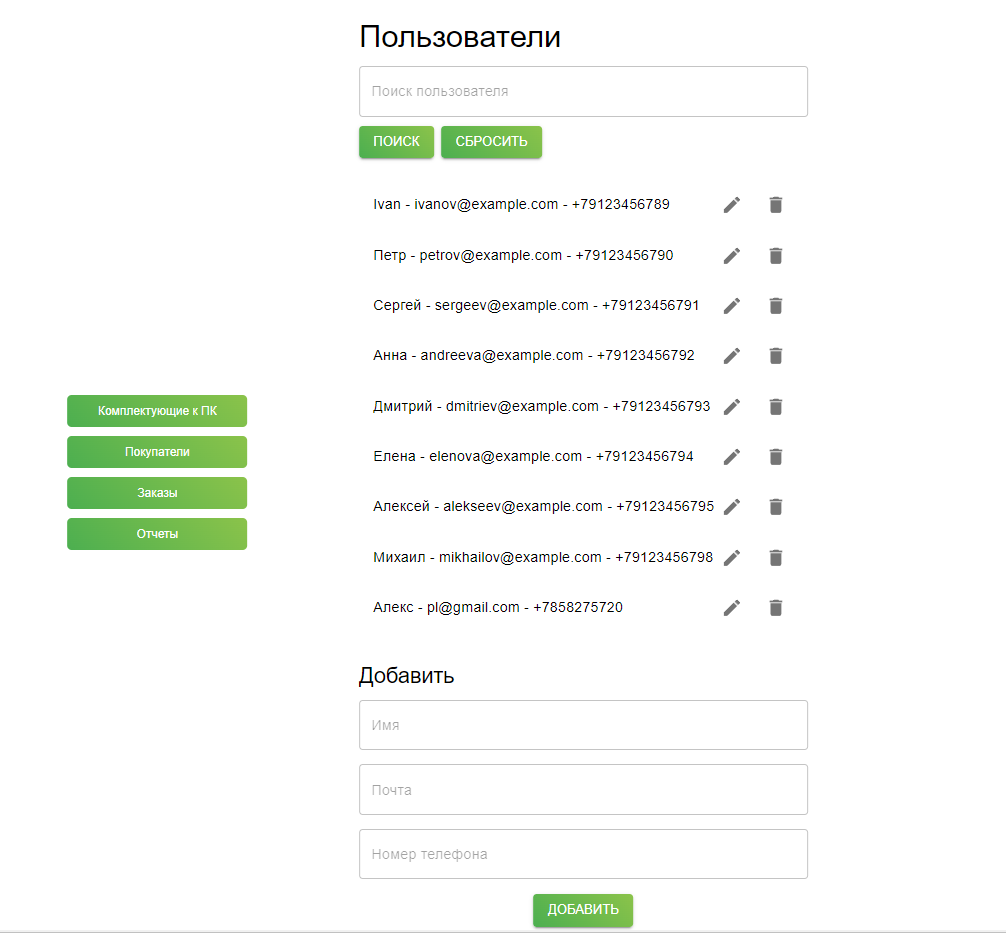


Контент:

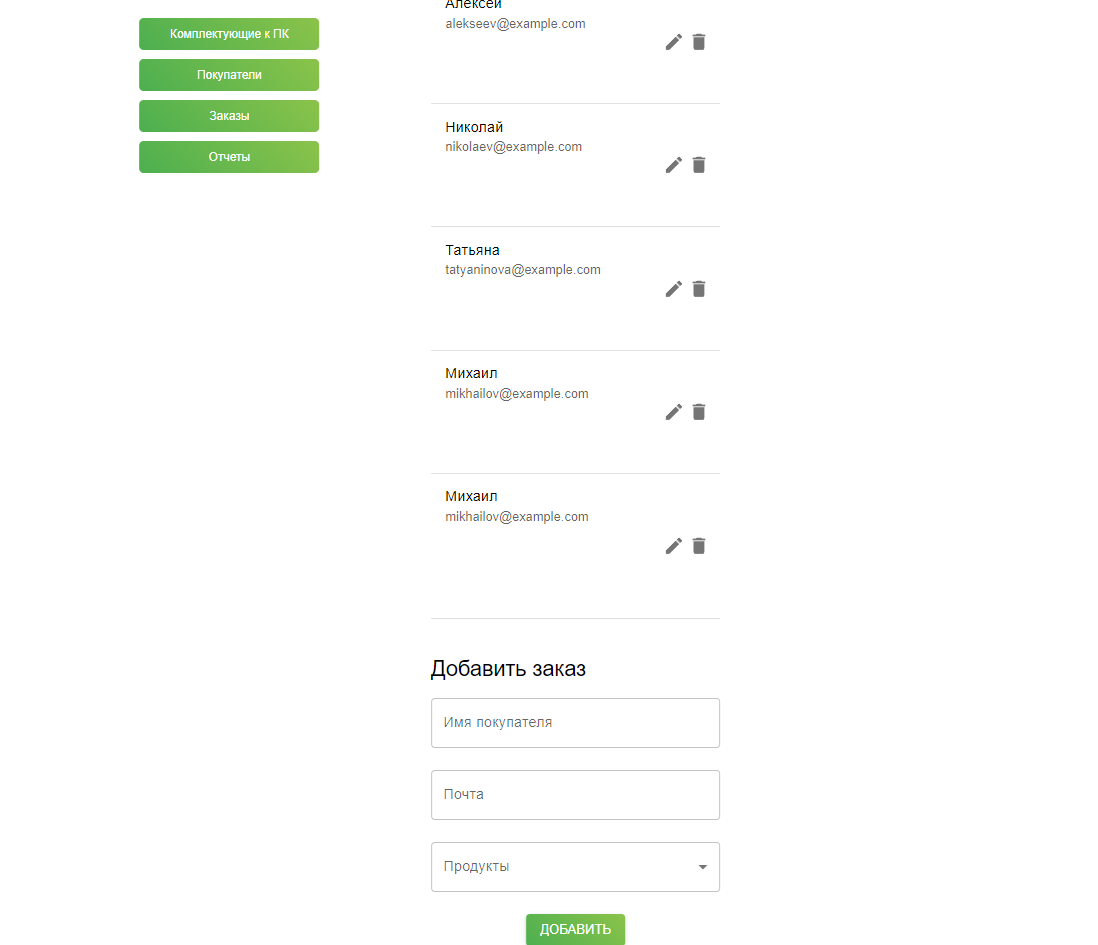
Весь список товаров. Возможность фильтровать по алфавиту и по стоимости по возрастанию или убыванию. Возможность добавлять удалять и редактировать товары.



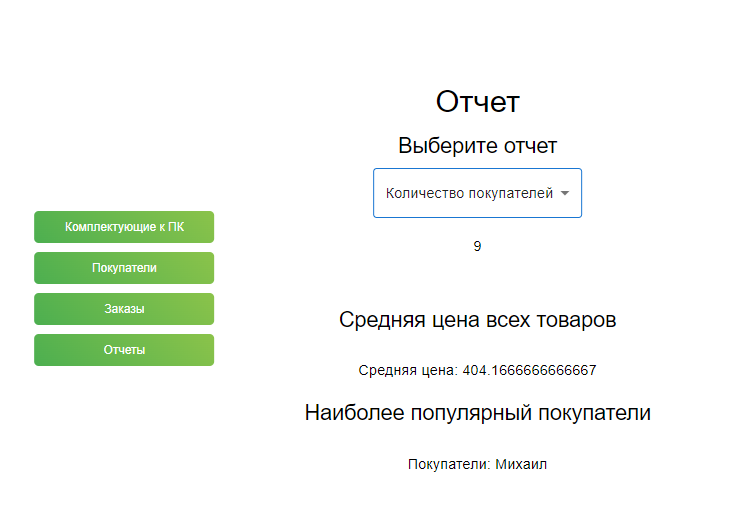
Весь список пользователей. Возможность поиска пользователя. Можете добавлять удалять и редактировать товары.



Список всех заказов. Возможность изменения удаления или добавления заказов.



Создание отчета для просмотра общего количества покупателей/заказов/товаров. Средняя цена всех товаров. Вывод частого покупателя.



Код:

Backend:

Models:  
Заказы:

@Setter  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@Document(collection = "orders")  
public class Order {  
 @Id  
 private String id;  
 @Indexed  
 private String buyerName;  
 private String buyerEmail;  
 @JsonProperty  
 @Indexed  
 private ArrayList<Product> selectedProducts;  
}

Продукты:

@Document(collection = "products")  
public class Product {  
 @Id  
 private String id;  
 private String Name;  
 @Indexed  
 private double Price;  
 private String Description;  
}

Пользователи:

@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Document(collection = "users")  
public class User {  
 @Id  
 private String id;  
 @Indexed  
 private String name;  
 private String email;  
 @Indexed  
 private String phone;  
}

Controller:

@RestController  
public class Controller {  
 public static MongoTemplate *mongoTemplate*;  
 @Autowired  
 MongoServiceImpl mongoService;  
 @PostMapping("/login")  
 public ResponseEntity<String> login(@RequestParam String username, @RequestParam String password) {  
 *mongoTemplate* = mongoService.mongoTemplate(username, password);  
 System.*out*.println(*mongoTemplate*.getCollectionNames());  
 return ResponseEntity.*ok*("Аунтефикация прошла успешно");  
 }  
 @PostMapping("/api/orders")  
 public Order createOrder(@RequestBody Order order) {  
 return *mongoTemplate*.save(order);  
 }  
  
 @GetMapping("/api/orders/{id}")  
 public Order getOrderById(@PathVariable String id) {  
 return *mongoTemplate*.findById(id, Order.class);  
 }  
  
 @GetMapping("/api/orders")  
 public ResponseEntity<List<Order>> getAllOrders() {  
 List<Order> list = *mongoTemplate*.findAll(Order.class);  
 return ResponseEntity.*ok*(list);  
 }  
  
 @PutMapping("/api/orders/{id}")  
 public Order updateOrder(@PathVariable String id, @RequestBody Order order) {  
 order.setId(id);  
 return *mongoTemplate*.save(order);  
 }  
  
 @DeleteMapping("/api/orders/{id}")  
 public void deleteOrder(@PathVariable String id) {  
 Order order = *mongoTemplate*.findById(id, Order.class);  
 if (order != null) {  
 *mongoTemplate*.remove(order);  
 }  
 }  
  
 @GetMapping("/api/orders/search")  
 public List<Order> searchOrders(@RequestParam String query) {  
 // Example of a simple search query using MongoTemplate  
 return *mongoTemplate*.find(  
 Query.*query*(Criteria.*where*("buyerName").regex(query, "i")), // Case-insensitive regex search on the 'name' field  
 Order.class  
 );  
 }  
  
 @PostMapping("/api/products")  
 public Product createProduct(@RequestBody Product product) {  
 return *mongoTemplate*.save(product);  
 }  
  
 @GetMapping("/api/products/{id}")  
 public Product getProductById(@PathVariable String id) {  
 return *mongoTemplate*.findById(id, Product.class);  
 }  
  
 @GetMapping("/api/products")  
 public List<Product> getAllProducts(@RequestParam(required = false) String sortBy,  
 @RequestParam(required = false) String order) {  
 Sort.Direction direction = Sort.Direction.*ASC*;  
 if (order != null && order.equalsIgnoreCase("desc")) {  
 direction = Sort.Direction.*DESC*;  
 }  
 Sort sort = Sort.*by*(direction, sortBy != null ? sortBy : "\_id"); // Default sort by "\_id" field if sortBy is not provided  
 return *mongoTemplate*.find(org.springframework.data.mongodb.core.query.Query.*query*(new org.springframework.data.mongodb.core.query.Criteria()).with(sort), Product.class);  
 }  
  
 @PutMapping("/api/products/{id}")  
 public Product updateProduct(@PathVariable String id, @RequestBody Product product) {  
 product.setId(id);  
 return *mongoTemplate*.save(product);  
 }  
  
 @DeleteMapping("/api/products/{id}")  
 public void deleteProduct(@PathVariable String id) {  
 Product product = *mongoTemplate*.findById(id, Product.class);  
 if (product != null) {  
 *mongoTemplate*.remove(product);  
 }  
 }  
 @PostMapping("/api/users")  
 public User createSeller(@RequestBody User seller) {  
 return *mongoTemplate*.save(seller);  
 }  
  
 @GetMapping("/api/users/{id}")  
 public User getSellerById(@PathVariable String id) {  
 return *mongoTemplate*.findById(id, User.class, "users");  
 }  
  
 @GetMapping("/api/users")  
 public List<User> getAllSellers(@RequestParam(required = false) String query) {  
 //System.out.println("Current Database URL: " + env.getProperty("spring.data.mongodb.uri"));  
 if (query != null) {  
 *mongoTemplate*.find(Query.*query*(Criteria.*where*("name").regex(query, "i")), User.class);  
 System.*out*.println(*mongoTemplate*.find(Query.*query*(Criteria.*where*("name").regex(query, "i")), User.class));  
 return *mongoTemplate*.find(Query.*query*(Criteria.*where*("name").regex(query, "i")), User.class); // Placeholder, returns all sellers for demonstration  
 }  
 return *mongoTemplate*.findAll(User.class);  
 }  
  
 @PutMapping("/api/users/{id}")  
 public User updateSeller(@PathVariable String id, @RequestBody User seller) {  
 seller.setId(id);  
 return *mongoTemplate*.save(seller);  
 }  
  
 @DeleteMapping("/api/users/{id}")  
 public void deleteSeller(@PathVariable String id) {  
 User seller = *mongoTemplate*.findById(id, User.class);  
 if (seller != null) {  
 *mongoTemplate*.remove(seller);  
 }  
 }  
  
 @GetMapping("api/reports/average-price")  
 public Double getAveragePrice() {  
 Aggregation aggregation = *newAggregation*(  
 Aggregation.*group*().avg("Price").as("averagePrice")  
 );  
  
 AggregationResults<Document> result = *mongoTemplate*.aggregate(aggregation, "products", Document.class);  
 List<Document> documents = result.getMappedResults(); // Получаем список документов  
  
 if (!documents.isEmpty()) {  
 Document firstDocument = documents.get(0); // Берем первый документ из списка  
 if (firstDocument.containsKey("averagePrice")) {  
 double averagePrice = firstDocument.getDouble("averagePrice");  
 System.*out*.println("Average Price of All Products: " + averagePrice);  
 return averagePrice;  
 } else {  
 System.*out*.println("Field 'averagePrice' not found in the result.");  
 return null;  
 }  
 } else {  
 System.*out*.println("No results found.");  
 return null;  
 }  
 }  
  
 @GetMapping("api/reports/most-popular-order-name")  
 public List<String> getMostPopularOrderName() {  
 Aggregation aggregation = *newAggregation*(  
 Aggregation.*group*("buyerName").count().as("orderCount")  
 );  
  
 AggregationResults<Document> result = *mongoTemplate*.aggregate(aggregation, "orders", Document.class);  
 List<Document> documents = result.getMappedResults();  
  
 System.*out*.println(result.getRawResults());  
  
 int max = 0;  
 for(Document doc : documents){  
 if(doc.getInteger("orderCount") > max){  
 max = doc.getInteger("orderCount");  
 }  
 }  
  
  
 List<String> mostPopularOrderName = new ArrayList<>();  
  
 if (!documents.isEmpty()) {  
 for(Document doc : documents){  
 if(doc.getInteger("orderCount") == max)  
 mostPopularOrderName.add(doc.getString("\_id"));  
 }  
 System.*out*.println("Most popular order name: " + mostPopularOrderName);  
 return mostPopularOrderName;  
 } else {  
 System.*out*.println("No orders found.");  
 mostPopularOrderName.add("No orders found.");  
 return mostPopularOrderName;  
 }  
 }  
  
  
 @GetMapping("/api/reports/get-len-users")  
 public AggregationCount getLenUsers() {  
 // Corrected aggregation query  
 Aggregation aggregation = Aggregation.*newAggregation*(Aggregation.*count*().as("total"));  
 AggregationResults<AggregationCount> result = *mongoTemplate*.aggregate(aggregation, "users", AggregationCount.class);  
  
 // Assuming the result contains a single document with the count  
 AggregationCount aggregationCount = result.getUniqueMappedResult();  
 if (aggregationCount!= null) {  
 System.*out*.println("Total Users Count: " + aggregationCount.getTotal());  
 }  
 return aggregationCount;  
 }  
  
  
 @GetMapping("/api/reports/get-len-products")  
 public AggregationCount getLenProducts() {  
 Aggregation aggregation = Aggregation.*newAggregation*(Aggregation.*count*().as("total"));  
 AggregationResults<AggregationCount> result = *mongoTemplate*.aggregate(aggregation, "products", AggregationCount.class);  
  
 AggregationCount aggregationCount = result.getUniqueMappedResult();  
 if (aggregationCount!= null) {  
 System.*out*.println("Total Products Count: " + aggregationCount.getTotal());  
 }  
 return aggregationCount;  
 }  
  
 @GetMapping("/api/reports/get-len-orders")  
 public AggregationCount getLenOrders() {  
 Aggregation aggregation = Aggregation.*newAggregation*(Aggregation.*count*().as("total"));  
 AggregationResults<AggregationCount> result = *mongoTemplate*.aggregate(aggregation, "orders", AggregationCount.class);  
  
 AggregationCount aggregationCount = result.getUniqueMappedResult();  
 if (aggregationCount!= null) {  
 System.*out*.println("Total Orders Count: " + aggregationCount.getTotal());  
 }  
 return aggregationCount;  
 }  
  
  
}

Service:

public class MongoServiceImpl implements MongoService{  
  
 public MongoTemplate mongoTemplate(String name, String password) {  
 String connectionString = "mongodb://" + name + ":" + password + "@localhost:27017/MarketDB";  
 MongoClientSettings mongoClientSettings = MongoClientSettings.*builder*()  
 .applyConnectionString(new ConnectionString(connectionString))  
 .build();  
 MongoClient mongoClient = MongoClients.*create*(mongoClientSettings);  
 System.*out*.println(connectionString);  
 return new MongoTemplate(mongoClient, "MarketDB");  
 }  
}

Frontend:

Страница для логина:

import React, { useState } from 'react';

import { useNavigate } from 'react-router-dom';

import { TextField, Button, Container, Typography, Box, Paper, Alert } from '@mui/material';

import { styled } from '@mui/system';

// Обновление стилизованного компонента для контейнера

const StyledPaper = styled(Paper)(({ theme }) => ({

  display: 'flex',

  flexDirection: 'column',

  alignItems: 'center',

  padding: theme.spacing(4),

  borderRadius: '15px',

  backgroundColor: '#f5f5f5',

  boxShadow: '0 4px 30px -12px rgba(0, 0, 0, 0.1)',

  transition: 'all 0.3s ease-in-out',

}));

const LoginForm = ({ onLogin }) => {

    const [username, setUsername] = useState('');

    const [password, setPassword] = useState('');

    const [loginError, setLoginError] = useState(false);

const handleLogin = async () => {

    try {

        const success = await onLogin(username, password);

        if (!success) {

            setLoginError(true);

        } else {

            setLoginError(false);

        }

    } catch (error) {

        console.error(error);

        setLoginError(true);

    }

};

    return (

        <Container maxWidth="xs" sx={{ height: '85vh', display: 'flex', justifyContent: 'center', alignItems: 'center' }}>

            <StyledPaper elevation={6} sx={{ width: '90%', maxWidth: 400 }}>

                <Typography variant="h4" align="center" gutterBottom sx={{ color: '#333' }}>

                    Вход

                </Typography>

                {loginError && ( // Отображаем сообщение об ошибке, если оно есть

                    <Alert severity="error" onClose={() => setLoginError(false)}>

                        Неверный логин или пароль.

                    </Alert>

                )}

                <TextField

                    variant="filled"

                    margin="normal"

                    fullWidth

                    label="Логин"

                    value={username}

                    onChange={(e) => setUsername(e.target.value)}

                    InputProps={{

                        style: { color: '#333' },

                    }}

                    sx={{ marginBottom: 2 }}

                />

                <TextField

                    variant="filled"

                    margin="normal"

                    fullWidth

                    label="Пароль"

                    type="password"

                    value={password}

                    onChange={(e) => setPassword(e.target.value)}

                    InputProps={{

                        style: { color: '#333' },

                    }}

                    sx={{ marginBottom: 2 }}

                />

                <Button

                    variant="contained"

                    color="primary"

                    fullWidth

                    onClick={handleLogin}

                    sx={{ marginTop: 2 }}

                    style={{ background: 'linear-gradient(45deg, #4CAF50, #8BC34A)'}}

                >

                    Войти

                </Button>

            </StyledPaper>

        </Container>

    );

};

export default LoginForm;

Основная страница:

import React, { useState } from 'react';

import { BrowserRouter as Router, Routes, Route, Link, Navigate, useNavigate } from 'react-router-dom';

import Sellers from './Sellers';

import Products from './Products';

import Orders from './Orders';

import LoginForm from './LoginForm';

import Reports from './Reports';

import { AppBar, Toolbar, Button, Typography, Container, Box, Alert } from '@mui/material';

function App() {

    const [isLoggedIn, setIsLoggedIn] = useState(false);

    const [authError, setAuthError] = useState(null);

    const [username, setUsername] = useState('');

    const [isSidebarOpen, setIsSidebarOpen] = useState(false); // Новое состояние для боковой панели

    const [content, setContent] = useState(null); // Состояние для хранения текущего контента

    const handleLogin = async (username, password) => {

        try {

            const formData = new URLSearchParams();

            formData.append('username', username);

            formData.append('password', password);

            const response = await fetch('http://localhost:8080/login', {

                method: 'POST',

                headers: {

                    'Content-Type': 'application/x-www-form-urlencoded', // Устанавливаем Content-Type в URL-кодированный формат

                },

                body: formData.toString(), // Преобразуем параметры в строку URL-кодированного формата

            });

            if (response.ok) {

                setIsLoggedIn(true);

                setUsername(username);

                setAuthError(null);

            } else {

                console.error('Ошибка аутентификации');

            }

        } catch (error) {

            console.error('Ошибка:', error);

        }

    };

    const handleLogout = () => {

        setIsLoggedIn(false);

        window.location.href = "http://localhost:5173";

    };

    return (

        <Router>

          <div style={{ display: 'flex', flexDirection: 'row', alignItems: 'center', height: '100vh', translate: '25%' }}>

            <div style={{ display: 'flex', flexDirection: 'column', alignItems: 'flex-start', width: '200px', backgroundColor: 'white', textAlign: 'center', marginRight: "100px"}}>

            {isLoggedIn && username === "admin"? (

                <>

              <Link to="/products" style={{width: '100%'}}>

                <button style={{ width: '100%', background: 'linear-gradient(45deg, #4CAF50, #8BC34A)', color: 'white', border: 'none', borderRadius: '5px', padding: '10px 20px', margin: '5px 0', cursor: 'pointer' }}>

                  Комплектующие к ПК</button>

              </Link>

              <Link to="/sellers" style={{width: '100%'}}>

                <button style={{ width: '100%', background: 'linear-gradient(45deg, #4CAF50, #8BC34A)', color: 'white', border: 'none', borderRadius: '5px', padding: '10px 20px', margin: '5px 0', cursor: 'pointer' }}>

                  Покупатели</button>

              </Link>

              <Link to="/orders" style={{width: '100%'}}>

                <button style={{ width: '100%', background: 'linear-gradient(45deg, #4CAF50, #8BC34A)', color: 'white', border: 'none', borderRadius: '5px', padding: '10px 20px', margin: '5px 0', cursor: 'pointer' }}>

                  Заказы</button>

              </Link>

              <Link to="/reports" style={{width: '100%'}}>

                <button style={{ width: '100%', background: 'linear-gradient(45deg, #4CAF50, #8BC34A)', color: 'white', border: 'none', borderRadius: '5px', padding: '10px 20px', margin: '5px 0', cursor: 'pointer' }}>

                  Отчеты</button>

              </Link>  </>

            ) : null}

            {isLoggedIn && username === "editor"? (

                <>

              <Link to="/products" style={{width: '100%'}}>

                <button style={{ width: '100%', background: 'linear-gradient(45deg, #4CAF50, #8BC34A)', color: 'white', border: 'none', borderRadius: '5px', padding: '10px 20px', margin: '5px 0', cursor: 'pointer' }}>

                  Комплектующие к ПК</button>

              </Link>

              <Link to="/orders" style={{width: '100%'}}>

                <button style={{ width: '100%', background: 'linear-gradient(45deg, #4CAF50, #8BC34A)', color: 'white', border: 'none', borderRadius: '5px', padding: '10px 20px', margin: '5px 0', cursor: 'pointer' }}>

                  Заказы</button>

              </Link>

                </>

            ) : null}

            {isLoggedIn && username === "reader"? (

                <>

              <Link to="/products" style={{width: '100%'}}>

                <button style={{ width: '100%', background: 'linear-gradient(45deg, #4CAF50, #8BC34A)', color: 'white', border: 'none', borderRadius: '5px', padding: '10px 20px', margin: '5px 0', cursor: 'pointer' }}>

                  Комплектующие к ПК</button>

              </Link>

              </>

            ) : null}

            </div>

            <div className='content'>

                    <Routes>

                        {!isLoggedIn && <Route path="/" element={<LoginForm onLogin={handleLogin} />} />}

                        {isLoggedIn && (

                            <>

                                <Route path="/sellers" element={<Sellers />} />

                                <Route path="/products" element={<Products />} />

                                <Route path="/orders" element={<Orders />} />

                                <Route path="/reports" element={<Reports />} />

                            </>

                        )}

                    </Routes>

                </div>

          </div>

        </Router>

      );

}

export default App;

Задание 3:

db.createUser(

{

user: "admin",

pwd: "123456",

roles: [{ role: "readWriteAnyDatabase", db: "MarketDB" }]

}

);

db.createUser(

{

user: "reader",

pwd: "1234",

roles: [

{

"role": "read",

"db": "MarketDB"

}

]

}

);

db.createUser(

{

user: "editor",

pwd: "12345",

roles: [

{

"role": "readWrite",

"db": "MarketDB"

}

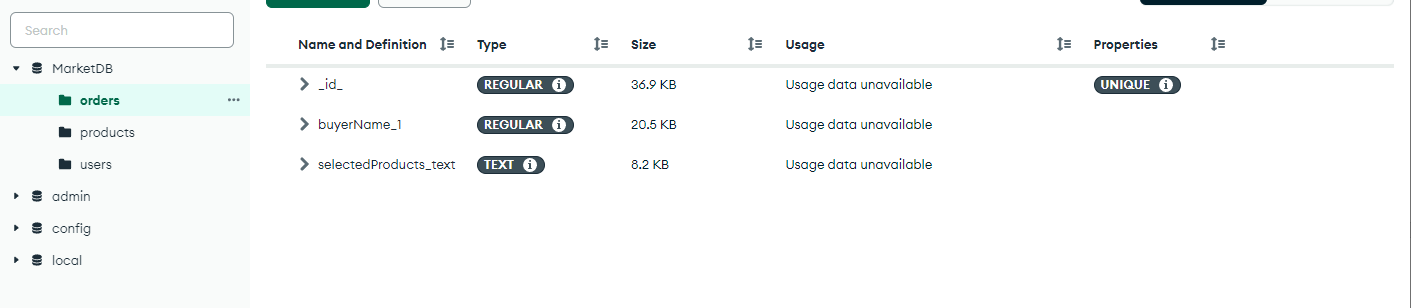
]

}

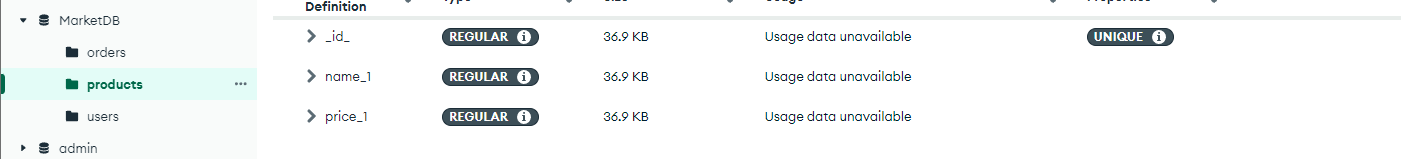
);

Задание 4:

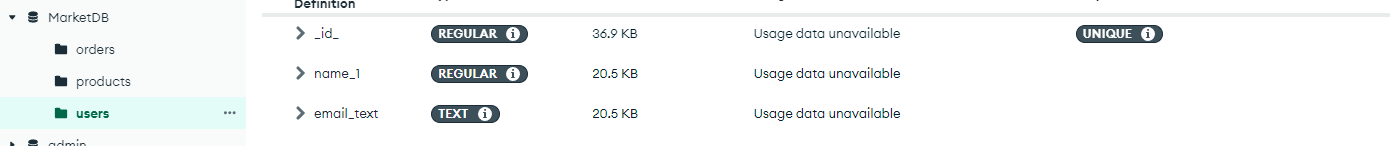
Для заказов:



Для продуктов:



Для пользователей:



Задание 5:

@GetMapping("api/reports/average-price")  
public Double getAveragePrice() {  
 Aggregation aggregation = *newAggregation*(  
 Aggregation.*group*().avg("Price").as("averagePrice")  
 );  
  
 AggregationResults<Document> result = *mongoTemplate*.aggregate(aggregation, "products", Document.class);  
 List<Document> documents = result.getMappedResults(); // Получаем список документов  
  
 if (!documents.isEmpty()) {  
 Document firstDocument = documents.get(0); // Берем первый документ из списка  
 if (firstDocument.containsKey("averagePrice")) {  
 double averagePrice = firstDocument.getDouble("averagePrice");  
 System.*out*.println("Average Price of All Products: " + averagePrice);  
 return averagePrice;  
 } else {  
 System.*out*.println("Field 'averagePrice' not found in the result.");  
 return null;  
 }  
 } else {  
 System.*out*.println("No results found.");  
 return null;  
 }  
}  
  
@GetMapping("api/reports/most-popular-order-name")  
public List<String> getMostPopularOrderName() {  
 Aggregation aggregation = *newAggregation*(  
 Aggregation.*group*("buyerName").count().as("orderCount")  
 );  
  
 AggregationResults<Document> result = *mongoTemplate*.aggregate(aggregation, "orders", Document.class);  
 List<Document> documents = result.getMappedResults();  
  
 System.*out*.println(result.getRawResults());  
  
 int max = 0;  
 for(Document doc : documents){  
 if(doc.getInteger("orderCount") > max){  
 max = doc.getInteger("orderCount");  
 }  
 }  
  
  
 List<String> mostPopularOrderName = new ArrayList<>();  
  
 if (!documents.isEmpty()) {  
 for(Document doc : documents){  
 if(doc.getInteger("orderCount") == max)  
 mostPopularOrderName.add(doc.getString("\_id"));  
 }  
 System.*out*.println("Most popular order name: " + mostPopularOrderName);  
 return mostPopularOrderName;  
 } else {  
 System.*out*.println("No orders found.");  
 mostPopularOrderName.add("No orders found.");  
 return mostPopularOrderName;  
 }  
}  
  
  
@GetMapping("/api/reports/get-len-users")  
public AggregationCount getLenUsers() {  
 // Corrected aggregation query  
 Aggregation aggregation = Aggregation.*newAggregation*(Aggregation.*count*().as("total"));  
 AggregationResults<AggregationCount> result = *mongoTemplate*.aggregate(aggregation, "users", AggregationCount.class);  
  
 // Assuming the result contains a single document with the count  
 AggregationCount aggregationCount = result.getUniqueMappedResult();  
 if (aggregationCount!= null) {  
 System.*out*.println("Total Users Count: " + aggregationCount.getTotal());  
 }  
 return aggregationCount;  
}  
  
  
@GetMapping("/api/reports/get-len-products")  
public AggregationCount getLenProducts() {  
 Aggregation aggregation = Aggregation.*newAggregation*(Aggregation.*count*().as("total"));  
 AggregationResults<AggregationCount> result = *mongoTemplate*.aggregate(aggregation, "products", AggregationCount.class);  
  
 AggregationCount aggregationCount = result.getUniqueMappedResult();  
 if (aggregationCount!= null) {  
 System.*out*.println("Total Products Count: " + aggregationCount.getTotal());  
 }  
 return aggregationCount;  
}  
  
@GetMapping("/api/reports/get-len-orders")  
public AggregationCount getLenOrders() {  
 Aggregation aggregation = Aggregation.*newAggregation*(Aggregation.*count*().as("total"));  
 AggregationResults<AggregationCount> result = *mongoTemplate*.aggregate(aggregation, "orders", AggregationCount.class);  
  
 AggregationCount aggregationCount = result.getUniqueMappedResult();  
 if (aggregationCount!= null) {  
 System.*out*.println("Total Orders Count: " + aggregationCount.getTotal());  
 }  
 return aggregationCount;  
}

Весь скрипт для корректного запуска в монго:

# Создание базы данных MarketDB

use MarketDB;

# Создание коллекции orders с полями buyerName, buyerEmail, selectedProducts

db.createCollection("orders");

# Создание коллекции users с полями email, name, phone

db.createCollection("users");

# Создание коллекции products с полями Name, Price, Description

db.createCollection("products");

db.products.insertMany( [

{

"Name": "Процессор",

"Price": 1000,

"Description": "Intel Core i7"

},

{

"Name": "Материнская плата",

"Price": 500,

"Description": "ASUS Prime Z390-A"

},

{

"Name": "Оперативная память",

"Price": 200,

"Description": "8GB DDR4 3200MHz"

},

{

"\_id": ObjectId("6653a11b6de206ba95748f0e"),

"Name": "Жесткий диск",

"Price": 150,

"Description": "SSD 256GB NVMe"

},

{

"Name": "Видеокарта",

"Price": 300,

"Description": "NVIDIA GeForce GTX 1060"

},

{

"Name": "Блок питания",

"Price": 250,

"Description": "Corsair RMx 750W"

},

{

"Name": "Кулер",

"Price": 120,

"Description": "Cooler Master Hyper H411R"

},

{

"Name": "Системный блок",

"Price": 400,

"Description": "Thermaltake Versa H15"

},

{

"Name": "Монитор",

"Price": 350,

"Description": "Acer R240HY bidx 23.8-Inch IPS"

},

{

"Name": "Клавиатура",

"Price": 80,

"Description": "Logitech K120 Wired Keyboard"

},

{

"Name": "Клава",

"Price": 200,

"Description": "Норм",

},

{

"Name": "Клаваа",

"Price": 1300,

"Description": "нет",

}

]);

db.orders.insertMany( [

{

"buyerEmail": "ivanov@example.com",

"buyerName": "Иван",

"selectedProducts": [

{

"component": "Процессор",

"quantity": 1

},

{

"component": "Оперативная память",

"quantity": 2

}

]

},

{

"buyerEmail": "petrov@example.com",

"buyerName": "Петр",

"selectedProducts": [

{

"component": "Видеокарта",

"quantity": 1

},

{

"component": "Блок питания",

"quantity": 1

}

]

},

{

"buyerEmail": "andreeva@example.com",

"buyerName": "Анна",

"selectedProducts": [

{

"component": "Материнская плата",

"quantity": 1

},

{

"component": "Системный блок",

"quantity": 1

}

]

},

{

"buyerEmail": "dmitriev@example.com",

"buyerName": "Дмитрий",

"selectedProducts": [

{

"component": "Монитор",

"quantity": 1

},

{

"component": "Клавиатура",

"quantity": 1

}

]

},

{

"buyerEmail": "elenova@example.com",

"buyerName": "Елена",

"selectedProducts": [

{

"component": "Процессор",

"quantity": 2

},

{

"component": "Оперативная память",

"quantity": 4

}

]

},

{

"buyerEmail": "alekseev@example.com",

"buyerName": "Алексей",

"selectedProducts": [

{

"component": "Видеокарта",

"quantity": 2

},

{

"component": "Блок питания",

"quantity": 2

}

]

},

{

"buyerEmail": "nikolaev@example.com",

"buyerName": "Николай",

"selectedProducts": [

{

"component": "Жесткий диск",

"quantity": 2

},

{

"component": "Кулер",

"quantity": 2

}

]

},

{

"buyerEmail": "tatyaninova@example.com",

"buyerName": "Татьяна",

"selectedProducts": [

{

"component": "Материнская плата",

"quantity": 2

},

{

"component": "Системный блок",

"quantity": 2

}

]

},

{

"buyerEmail": "mikhailov@example.com",

"buyerName": "Михаил",

"selectedProducts": [

{

"component": "Монитор",

"quantity": 2

},

{

"component": "Клавиатура",

"quantity": 2

}

]

}

], {

ordered: false

});

db.users.insertMany( [

{

"email": "ivanov@example.com",

"name": "Ivan",

"phone": "+79123456789"

},

{

"email": "petrov@example.com",

"name": "Петр",

"phone": "+79123456790"

},

{

"email": "sergeev@example.com",

"name": "Сергей",

"phone": "+79123456791"

},

{

"email": "andreeva@example.com",

"name": "Анна",

"phone": "+79123456792"

},

{

"email": "dmitriev@example.com",

"name": "Дмитрий",

"phone": "+79123456793"

},

{

"email": "elenova@example.com",

"name": "Елена",

"phone": "+79123456794"

},

{

"email": "alekseev@example.com",

"name": "Алексей",

"phone": "+79123456795"

},

{

"email": "mikhailov@example.com",

"name": "Михаил",

"phone": "+79123456798"

},

{

"name": "Алекс",

"email": "pl@gmail.com",

"phone": "+7858275720"

}

]);

db.products.aggregate( [

{

$group: {

\_id: "$name",

count: { $sum: 1 }

}

}

])

db.users.countDocuments()

db.orders.countDocuments()

db.products.aggregate( [

{

$group: {

\_id: null,

avgPrice: { $avg: "$price" }

}

}

])

db.orders.aggregate( [

{ $unwind: "$selectedProducts" },

{

$group: {

\_id: "$selectedProducts.component",

count: { $sum: 1 }

}

},

{ $sort: { count: -1 } },

])

db.createUser(

{

user: "admin",

pwd: "123456",

roles: [{ role: "readWriteAnyDatabase", db: "MarketDB" }]

}

);

db.createUser(

{

user: "reader",

pwd: "1234",

roles: [

{

"role": "read",

"db": "MarketDB"

}

]

}

);

db.createUser(

{

user: "editor",

pwd: "12345",

roles: [

{

"role": "readWrite",

"db": "MarketDB"

}

]

}

);

Ссылка на git: https://github.com/prolgrammer/mongoApp.git