|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |
| Институт комплексной безопасности и специального приборостроения |
| Кафедра «Цифровые технологии обработки данных» |

**ОТЧЕТ**

**по практической работе**

**«ДОКУМЕНТООРИЕНТИРОВАННАЯ СУБД MONGODB»**

**по дисциплине «Нереляционные системы управления базами данных»**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил |  |  | Смирнов И.А.  *фамилия, имя, отчество* |
| шифр | 21Б0700 | группа | БСБО-11-21 |
|  |  |  |  |
| Проверил |  | к.т.н., доцент  *ученая степень, должность* | Ильин Д.Ю.  *фамилия, имя, отчество* |

**Москва 2023г.**

**Цель практической работы**

Цель настоящей практической работы – научиться использовать документоориентированную СУБД MongoDB.

**Задачи практической работы**

Для достижения поставленной цели необходимо решить следующие задачи:

1. Спроектировать программное обеспечение и отразить в результирующих схемах применение MongoDB.
2. Спроектировать схему базы данных MongoDB.
3. Установить MongoDB и осуществить к нему ручной доступ через любое доступное программное обеспечение.
4. Разработать программное обеспечение, использующее MongoDB для данных предметной области, определенной вариантом задания.
5. Протестировать программное обеспечение и продемонстрировать корректность его работы.
6. Подготовить ответы на контрольные вопросы.
7. Составить отчет о проведенной работе.

Предметная область: интернет-магазин. Реализация через REST API.

Задача для работы с СУБД: полнотекстовый поиск.

Схема ПО.

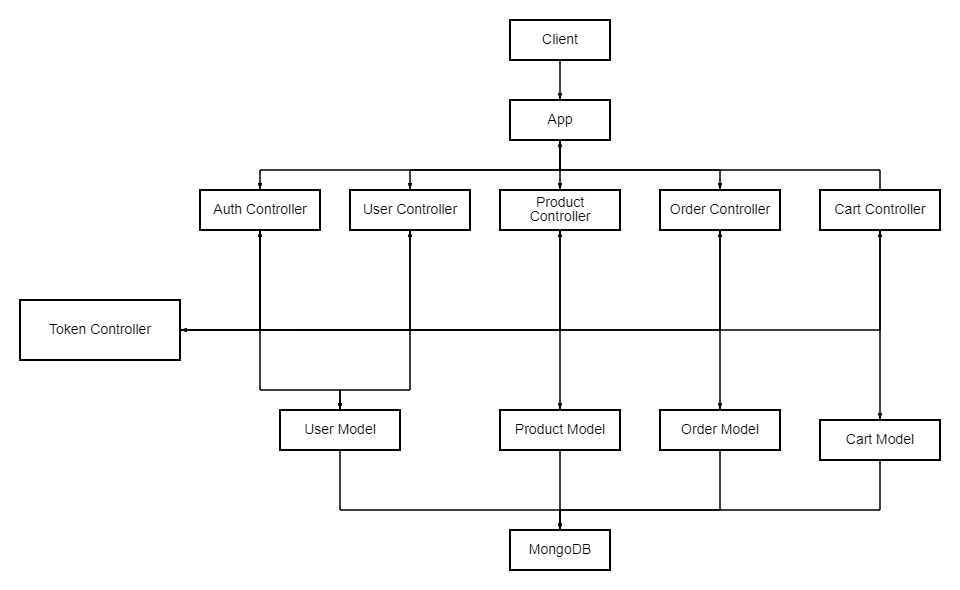


Рис. 1 – схема ПО.

Результат проектирование БД.

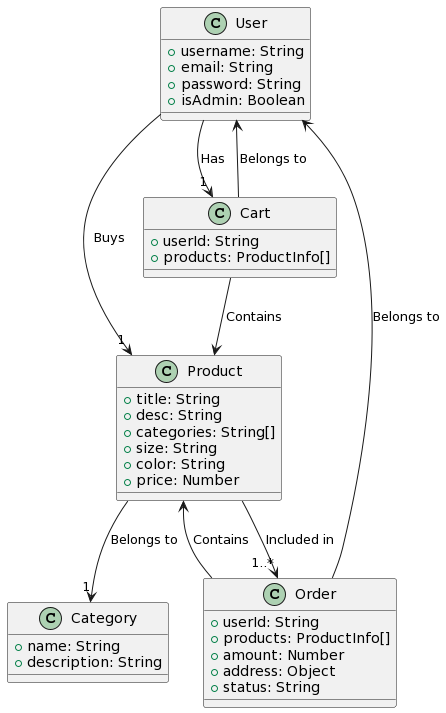


Рис. 2 – схема базы данных MongoDB.

**Реализация**

Для решения задачи использовал Express, Mongoose. Для начала надо подключиться к БД.

const PORT = 3000;  
const LINK = `http://localhost:${PORT}`;  
const URL = "mongodb://localhost:27017/eShop";  
  
mongoose  
 .connect(URL)  
 .then((res) => *console*.log('Подлючение к MongoDB завершено'))  
 .catch((err) => *console*.log(`Ошибка подлючения к бд: ${err}`));

Листинг 1 - подключение к БД.

Создал роутинг для приложения и запустил его.

app.use(express.*json*());  
  
app.use("/api/users", *userRoute*);  
app.use("/api/auth", *authRoute*);  
app.use("/api/products", *productRoute*);  
app.use("/api/carts", *cartRoute*);  
app.use("/api/orders", *orderRoute*);  
  
app.listen(PORT, (err) => {  
 err ? *console*.log(err) : *console*.log(`Приложение запущено на порту: ${PORT}. Перейдите на : ${LINK}`);  
});

Листинг 2 – роутинг приложения.

Для регистрации пользователей и последующего добавления их в БД, была создана пользовательская схема.

const mongoose = *require*('mongoose');  
  
const UserSchema = new mongoose.Schema({  
 username: {type: String, required: true, unique: true},  
 email: {type: String, required: true, unique: true},  
 password: {type: String, required: true},  
 isAdmin: {  
 type: Boolean,  
 default: false,  
 },  
 },  
 {timestamps: true}  
);  
  
const *User* = mongoose.model("User", UserSchema);  
  
*module*.exports = *User*;

Листинг 2.1 – модель пользователя.

Добавление новых пользователей происходит через регистрацию, вход по логину

*router*.post("/register", async (req, res) => {  
 const newUser = new *User*({  
 username: req.body.username,  
 email: req.body.email,  
 password: *cryptoJS*.AES.encrypt(req.body.password, 'laba').toString()  
 });  
 try {  
 const savedUser = await newUser.save();  
 res.status(200).json(savedUser);  
 } catch (e) {  
 res.status(500).json(e);  
 }  
});  
  
*router*.post("/login", async (req, res) => {  
 try {  
 const user = await *User*.findOne({username: req.body.username});  
 !user && res.status(401).json("Неверные данные");  
  
 const hashedPassword = *cryptoJS*.AES.decrypt(  
 user.password,  
 "laba"  
 )  
  
 const originalPassword = hashedPassword.toString(*cryptoJS*.enc.Utf8);  
  
 originalPassword !== req.body.password && res.status(401).json("Пароли не верны!");  
  
 const accessToken = *jwt*.sign(  
 {  
 id: user.\_id,  
 isAdmin: user.isAdmin,  
 },  
 "Laba",  
 {expiresIn: "3d"}  
 )  
  
 const {password, ...other} = user.\_doc;  
  
 res.status(200).json({...other, accessToken});  
 } catch (e) {  
 res.status(500).json(e);  
 }  
})

Листинг 2.3 – создание пользователя.

Реализация обновления, удаления и поиска пользователей. Доступ к операциям определяется по токену пользователя. Обычному пользователю запрещено удалять и изменять пользователей.

*router*.put("/:id", verifyTokenAndAuthorization, async (req, res) => {  
 if (req.body.password) {  
 req.body.password = CryptoJS.AES.encrypt(  
 req.body.password,  
 "Laba"  
 ).toString();  
 }  
  
 try {  
 const updatedUser = await *User*.findByIdAndUpdate(  
 req.params.id,  
 {  
 $set: req.body,  
 },  
 {new: true}  
 );  
 res.status(200).json(updatedUser);  
 } catch (err) {  
 res.status(500).json(err);  
 }  
});  
  
*router*.delete("/:id", verifyTokenAndAuthorization, async (req, res) => {  
 try {  
 await *User*.findByIdAndDelete(req.params.id);  
 res.status(200).json("Пользователь удалён");  
 } catch (err) {  
 res.status(500).json(err);  
 }  
});  
  
*router*.get("/find/:id", verifyTokenAndAdmin, async (req, res) => {  
 try {  
 const user = await *User*.findById(req.params.id);  
 const {password, ...others} = user.\_doc;  
 res.status(200).json(others);  
 } catch (err) {  
 res.status(500).json(err);  
 }  
});  
  
*router*.get("/", verifyTokenAndAdmin, async (req, res) => {  
 try {  
 const users = await *User*.find();  
 res.status(200).json(users);  
 } catch (err) {  
 res.status(500).json(err);  
 }  
});

Листинг 2.3 - код модели Пользователя.

const verifyToken = (req, res, next) => {  
 const authHeader = req.headers.token;  
 if (authHeader) {  
 const token = authHeader.split(" ")[1];  
 *jwt*.verify(token, "Laba", (err, user) => {  
 if (err) res.status(403).json("Невалидный токен");  
 req.user = user;  
 next();  
 });  
 } else {  
 return res.status(401).json("Авторизация не прошла");  
 }  
};  
  
const verifyTokenAndAuthorization = (req, res, next) => {  
 verifyToken(req, res, () => {  
 if (req.user.id === req.params.id || req.user.isAdmin) {  
 next();  
 } else {  
 res.status(403).json("Неверный токен");  
 }  
 });  
};  
  
const verifyTokenAndAdmin = (req, res, next) => {  
 verifyToken(req, res, () => {  
 if (req.user.isAdmin) {  
 next();  
 } else {  
 res.status(403).json("Неверный токен");  
 }  
 });  
};

Листинг 3 – проверка пользовательских токенов.

CRUD операции для товара, здесь же был реализован полнотекстовый поиск товаров.

*router*.post("/", verifyTokenAndAdmin, async (req, res) => {  
 const newProduct = new *Product*(req.body);  
 const categories = req.body.categories;  
  
 try {  
 const savedProduct = await newProduct.save();  
  
 for (const categoryName of categories) {  
 let category = await *Category*.findOne({name: categoryName});  
  
 if (!category) {  
 category = await *Category*.create({  
 name: categoryName  
 });  
 }  
 }  
  
 res.status(200).json(savedProduct);  
 } catch (err) {  
 res.status(500).json(err);  
 }  
});  
  
*router*.put("/:id", verifyTokenAndAdmin, async (req, res) => {  
 try {  
 const updatedProduct = await *Product*.findByIdAndUpdate(  
 req.params.id,  
 {  
 $set: req.body,  
 },  
 {new: true}  
 );  
 res.status(200).json(updatedProduct);  
 } catch (err) {  
 res.status(500).json(err);  
 }  
});  
  
*router*.delete("/:id", verifyTokenAndAdmin, async (req, res) => {  
 try {  
 await *Product*.findByIdAndDelete(req.params.id);  
 res.status(200).json("Товар удалён");  
 } catch (err) {  
 res.status(500).json(err);  
 }  
});  
  
*router*.get("/find/:id", async (req, res) => {  
 try {  
 const product = await *Product*.findById(req.params.id);  
 res.status(200).json(product);  
 } catch (err) {  
 res.status(500).json(err);  
 }  
});  
  
*router*.get("/", async (req, res) => {  
 let qCategory = req.query.category || "All";  
 const qSearch = req.query.search || "";  
 let qSort = req.query.sort || "rating";  
  
 const categoriesObject = await *Category*.find();  
  
 const categories = categoriesObject.map(category => category.name);  
  
 try {  
  
 qCategory === "All"  
 ? (qCategory = [...categories])  
 : (qCategory = req.query.category.split(","));  
  
 req.query.sort ? (qSort = req.query.sort.split(",")) : (qSort = [qSort]);  
  
 let sortBy = {};  
 if (qSort[1]) {  
 sortBy[qSort[0]] = qSort[1];  
 } else {  
 sortBy[qSort[0]] = "asc";  
 }  
  
 const products = await *Product*.find({$text: {$search: qSearch}})  
 .where("categories")  
 .in([...qCategory])  
 .sort(sortBy)  
  
 res.status(200).json(products);  
 } catch (err) {  
 res.status(500).json(err);  
 }  
});

Листинг 4 – CRUD операции для товара.

Реализация полнотекстового поиска

const products = await *Product*.find({$text: {$search: qSearch}})

const *Product* = mongoose.model("Product", ProductSchema);

В приложении реализовано создание, чтение, изменение и удаление пользователей, товаров, корзин и заказов. Пример создания, изменения, чтения и удаления товара. Для того, чтобы создать новый товар, нужно обладать правами пользователя. Зайдём под админом.



Рис. 1 – данные админа.

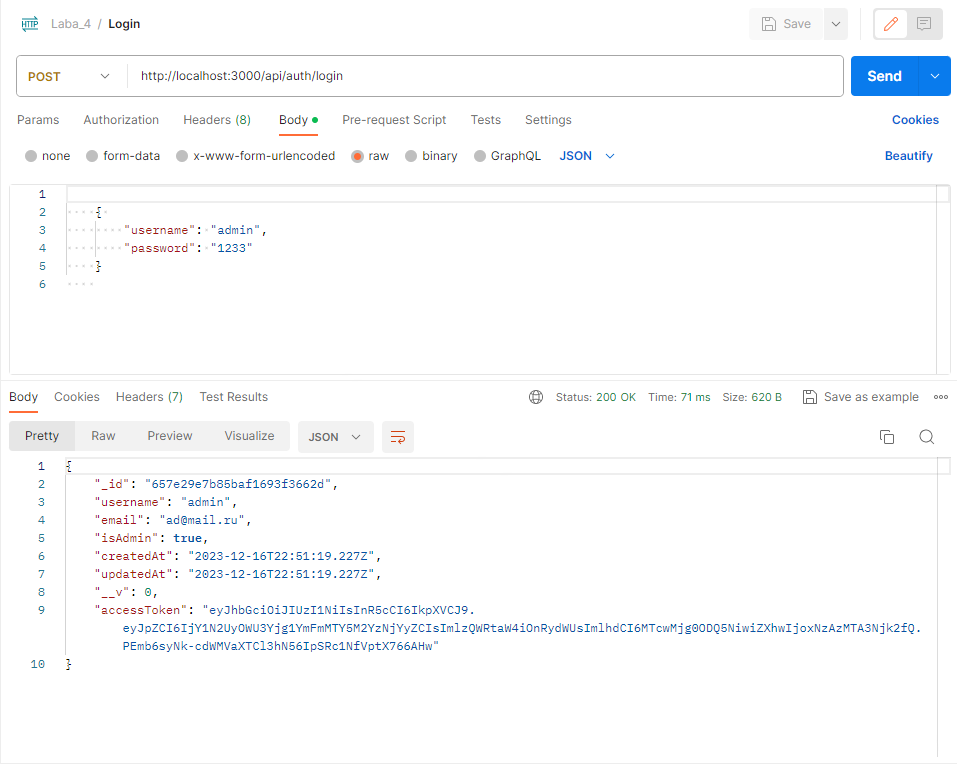


Рис. 2 – вход под админом.

Передача специального токена в Header запроса.

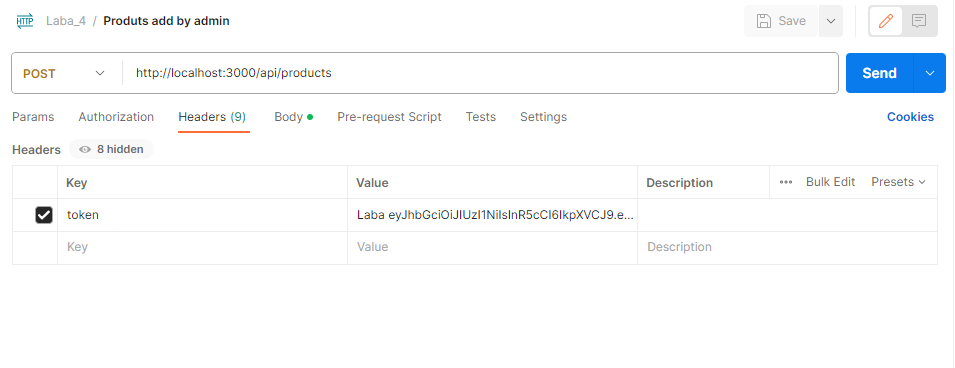


Рис. 3 – токен админа.

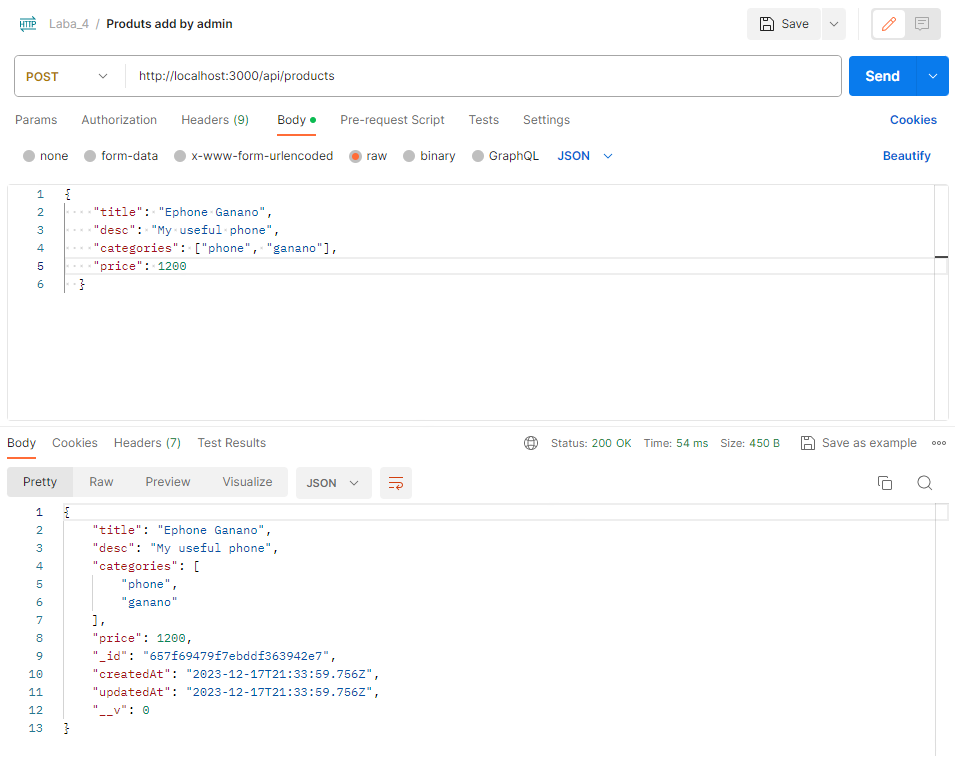


Рис. 4 – создание товара.

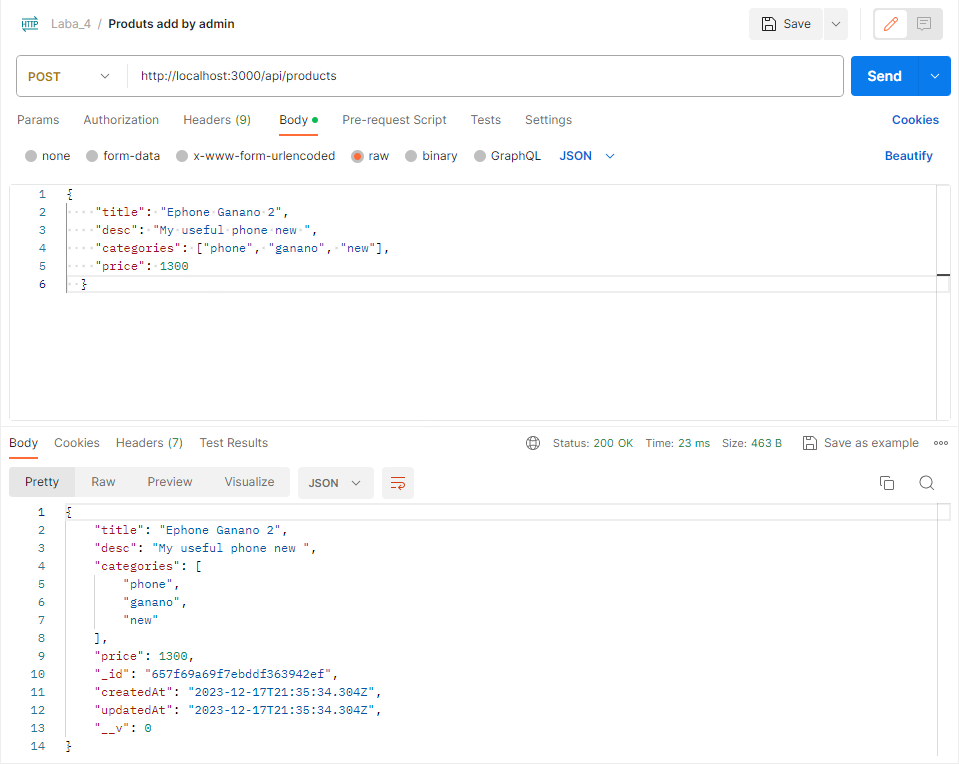


Рис. 5 – обновление товара.

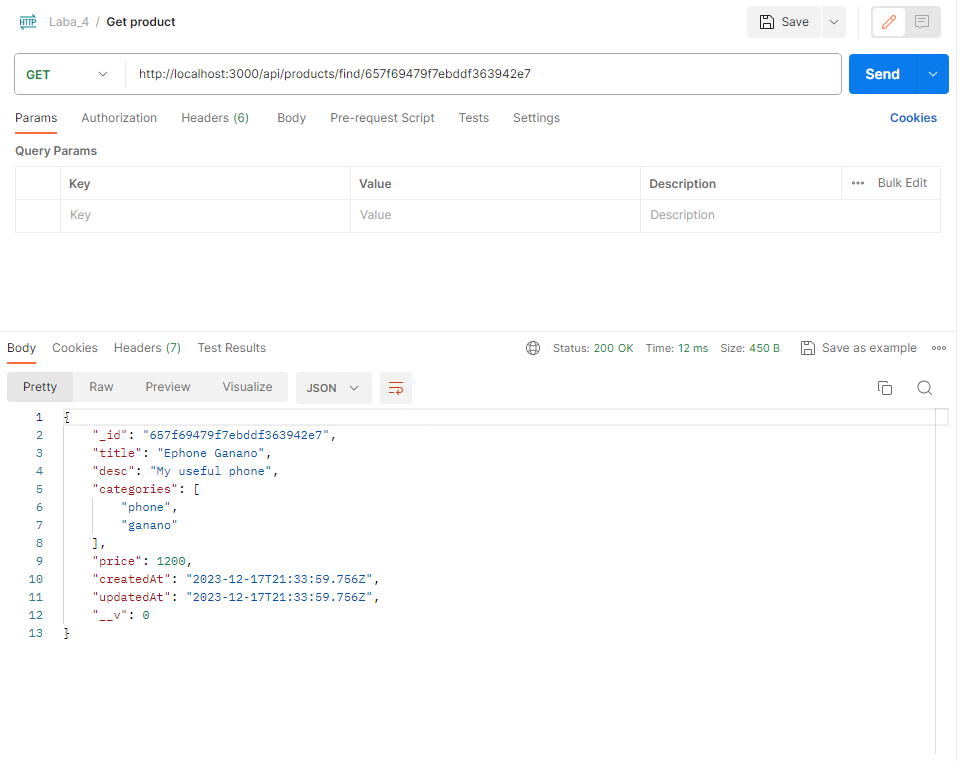


Рис.6 – чтение товара.

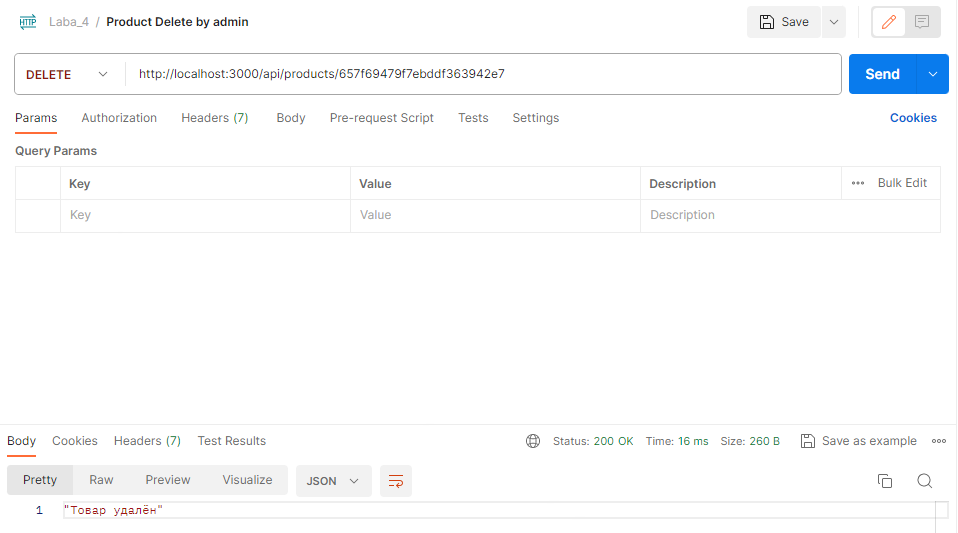


Рис. 7 – удаление товара.

Полнотекстовый поиск был реализован для поиска товаров в магазине.

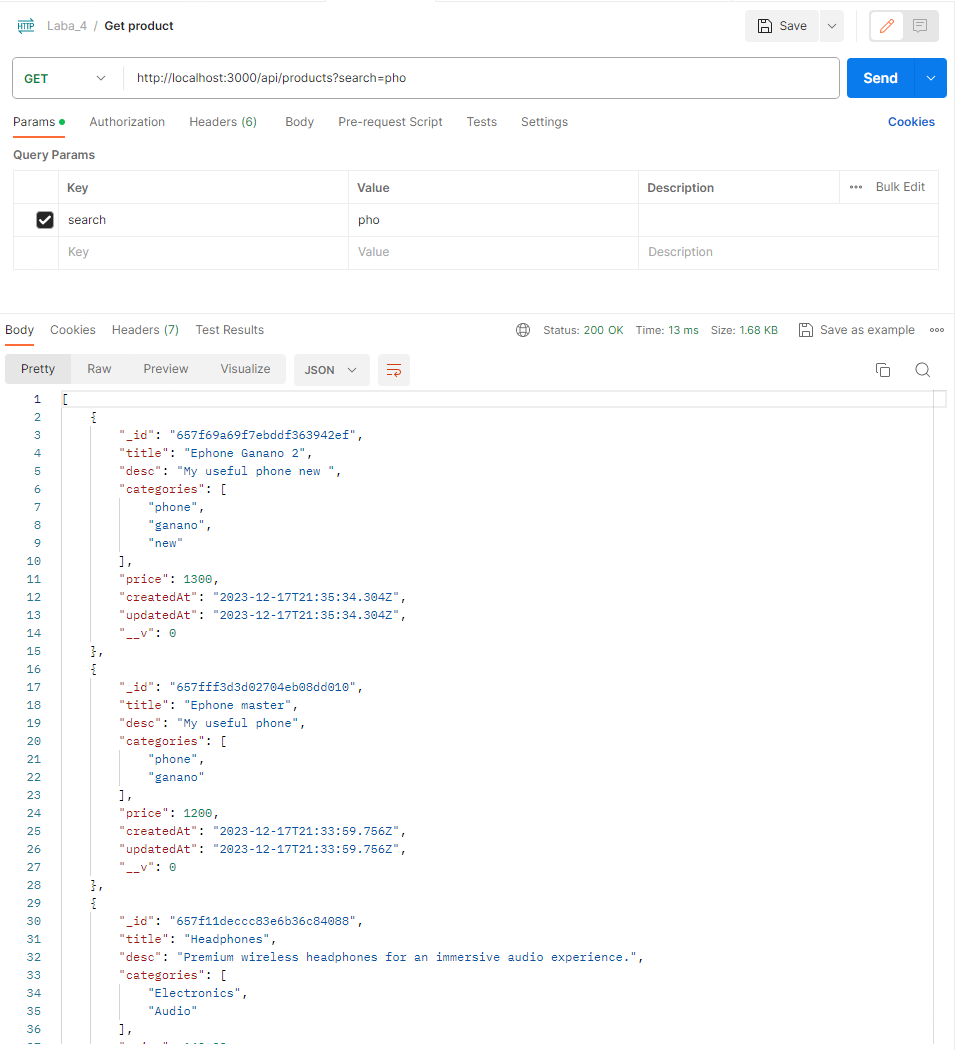
****

Рис. 8 – результат поиска по запросу “pho”.

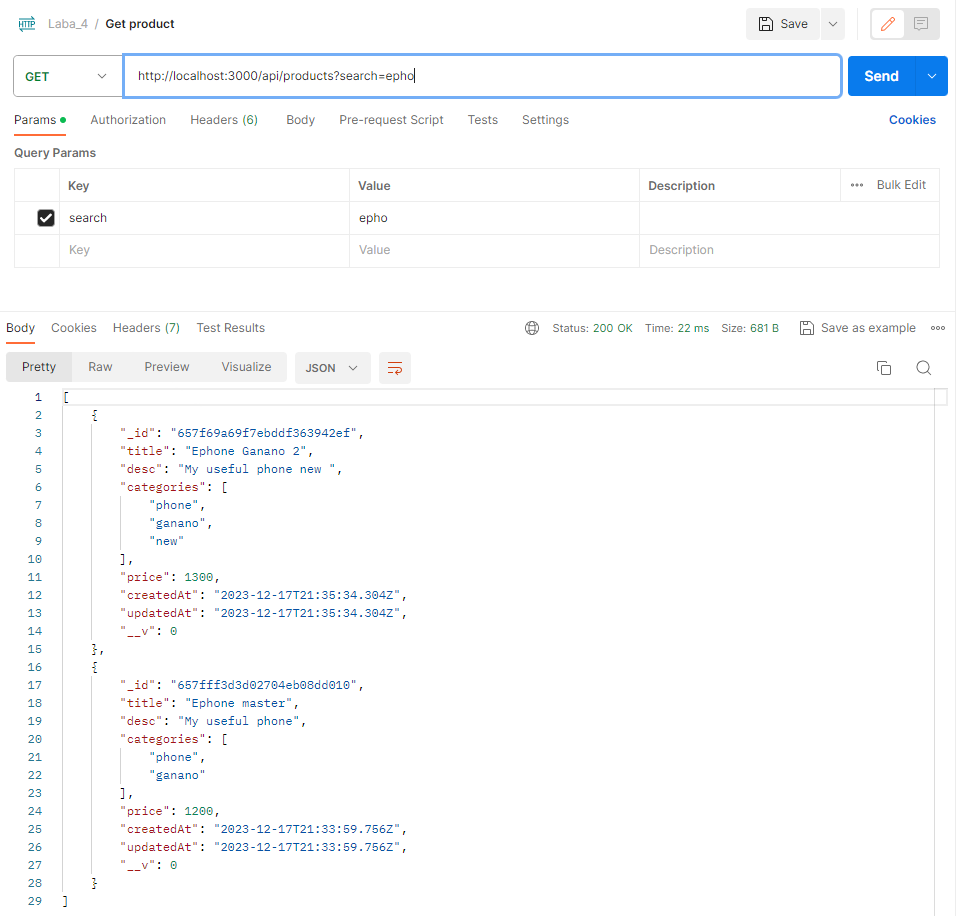


Рис. 9 - результат поиска по запросу “epho”.

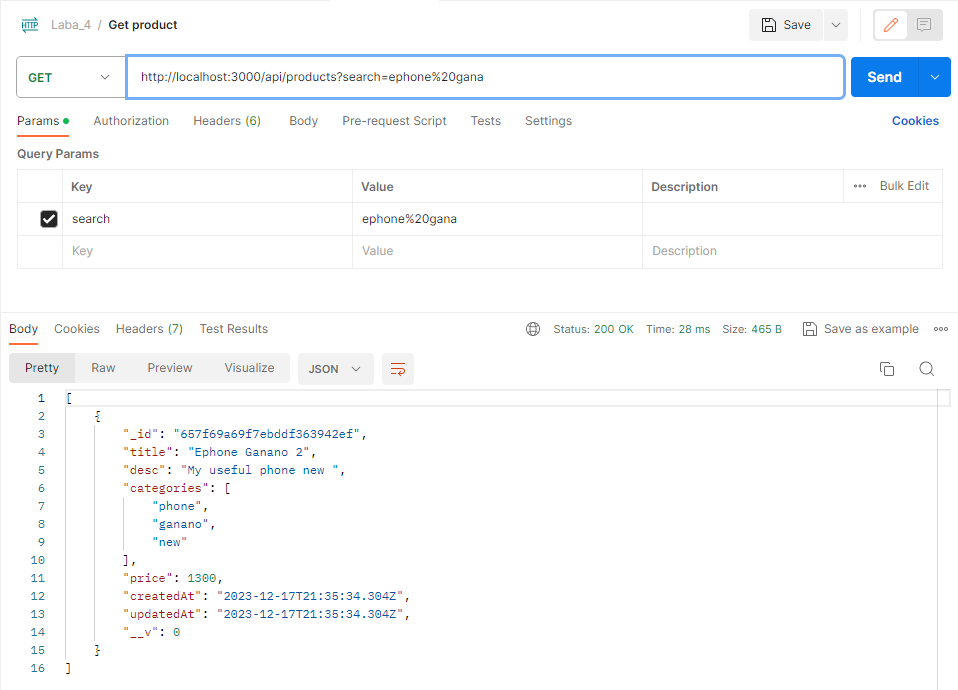


Рис. 10 - результат поиска по запросу “ephone%20gana”.

**Вывод**

В рамках данной практической работы, основной целью является изучение и использование документоориентированной СУБД MongoDB. Для достижения этой цели были выполнены следующие задачи:

1. Спроектировано программное обеспечение, в котором MongoDB используется для хранения данных в соответствии с предметной областью, определенной вариантом задания.
2. Разработана схема базы данных MongoDB, которая позволяет эффективно хранит информацию о пользователях, товарах, корзинах и заказах.
3. Установлена и сконфигурирована MongoDB, а также осуществлен ручной доступ к базе данных с использованием доступных программных средств.
4. Написано программное обеспечение, которое взаимодействует с MongoDB с использованием REST API для выполнения операций, связанных с предметной областью.
5. Протестировано программное обеспечение, чтобы убедиться в корректности его работы, а также чтобы выявить и устранить возможные проблемы.
6. Подготовлены ответы на контрольные вопросы, связанные с использованием MongoDB и документоориентированными СУБД.
7. Составлен отчет о выполненной работе, включая информацию о схеме базы данных, описание программного обеспечения и результаты тестирования.

В контексте задачи для работы с СУБД, был реализован полнотекстовый поиск

**Ответы на контрольные вопросы**

1. MongoDB применяется в информационных системах для хранения и управления большими объемами неструктурированных данных. Основные области применения включают в себя управление данными веб-сайтов, журналами событий, аналитическими системами, системами учета и хранения данных IoT (Интернета вещей), каталогами продуктов, и многое другое.
2. Ограничения MongoDB в отношении операций агрегации включают ограничение по размеру результата агрегации (по умолчанию 16 МБ), ограничение по времени выполнения (по умолчанию 10 минут), и ограничения на ресурсы, доступные серверу. Например, при выполнении агрегации, сервер MongoDB может потреблять больше памяти, чем доступно, что может привести к прерыванию операции.
3. Для программного взаимодействия с MongoDB могут использоваться различные библиотеки и драйверы. Некоторые из них включают официальный драйвер MongoDB для разных языков программирования, такие как Python, Java, Node.js, и другие. Также существуют сторонние библиотеки и фреймворки, которые облегчают взаимодействие с MongoDB.
4. Настройки коллекций документов в MongoDB включают в себя параметры, такие как размер документа, индексы, правила доступа, хранение данных и другие. Коллекции также могут иметь параметры для управления хранением данных, кэшированием, шардингом и репликацией.
5. Документоориентированные СУБД, как MongoDB, характеризуются следующими особенностями:
   1. Хранение данных в формате BSON (Binary JSON), что удобно для хранения и передачи неструктурированных данных.
   2. Гибкая схема данных, позволяющая хранить разные типы данных в одной коллекции без строгой схемы.
   3. Поддержка масштабирования горизонтального (шардинга) для обработки больших объемов данных.
   4. Поддержка репликации для обеспечения отказоустойчивости и доступности данных.
   5. Мощный язык запросов и агрегации для эффективного извлечения данных.
   6. Возможность работы с неструктурированными данными, такими как текст, изображения и географические данные.
   7. Высокая производительность и масштабируемость для обработки большого количества операций чтения и записи.