МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-14 «Интеллектуальные системы информационной безопасности»

Администрирование баз данных

Практическая работа № 1

ОТЧЁТ

Выполнил студент группы
БСБО-07-20
Любовский С.В.

Москва 2023

Выполнение задания.

1. Создадим базу **shop** с коллекциями: **product**, **productTypes**, **clients**, **carts**, **orders.** Определим валидаторы для данных схем. Все это реализуем в двух блоках **collectionUtils** и **schemaStore**.

Реализация блока collectionUtils:

```
const collectionUtils = {
    initialized: false,

    createAll: function () {
        for (const [key, value] of Object.entries(schemaStore)) {
            this._createCollection(key.replace("Schema", "") + "s", value);
        }
        this.initialized = true;
    },
    dropAll: function () {
        for (const [key, _] of Object.entries(schemaStore)) {
            this._dropCollection(key.replace("Schema", "") + "s");
        }
        this.initialized = false;
    },
    clearAll: function () {
        for (const [key, _] of Object.entries(schemaStore)) {
            this._clearCollection(key.replace("Schema", "") + "s");
        }
    },
    _createCollection: function (name, schema) {
        db.createCollection(name, schema);
    },
    _dropCollection: function (name) {
        db[name].drop();
    },
    _clearCollection: function (name) {
        db[name].deleteMany({});
    }
}
```

Реализация блока schemaStore:

```
const schemaStore = {
    productSchema: {
        validator: {
            $jsonSchema: {
                bsonType: 'object',
                title: 'product object schema',
                required: ['amount', 'price', 'name'],
                properties: {
                    amount: {
                        bsonType: 'int',
                        description: 'must be an integer and is required'
                },
                price: {
                    bsonType: 'double',
                    description: 'must be a double and is required'
                },
                name: {
                    bsonType: 'string',
                    description: 'must be a string and is required'
                },
                type: {
                    bsonType: "objectId",
                    description: "must be an ObjectId and is required"
                }
            }
        }
    }
},

productTypeSchema: {
    validator: {
        $jsonSchema: {
            bsonType: 'object',
            title: 'type object schema',
            required: ['name'],
            properties: {
                name: {
                    bsonType: 'string',
                    description: 'must be a string and is required'
                },
            }
        }
    }
},

clientSchema: {
    validator: {
        $jsonSchema: {
            bsonType: 'object',
            title: 'client object schema',
            required: ['name', 'email', 'phone'],
            properties: {
                name: {
                    bsonType: 'string',
                    description: 'must be a string and is required'
                },
                email: {
                    bsonType: 'string',
                    description: 'must be a string and is required'
                },
                phone: {
                    bsonType: 'string',
                        description: 'must be a string and is required',
                        pattern: '^[0-9]{3}-[0-9]{3}-[0-9]{4}$',
                    },
                }
            }
        }
    },

    orderSchema: {
        validator: {
            $jsonSchema: {
                bsonType: 'object',
                title: 'order object schema',
                required: ['client', 'products', 'total', 'date',
'status'],
                properties: {
                    client: {
                        bsonType: 'objectId',
                        description: 'must be an objectId and is required'
                    },
                    products: {
                        bsonType: 'array',
                        description: 'must be an array and is required',
                        items: {
                            bsonType: 'object',
                            properties: {
                                productId: {
                                    bsonType: 'objectId',
                                    description: 'must be an objectId and
is required'
                                },
                                amount: {
                                    bsonType: 'int',
                                    description: 'must be an integer and is
required'
                                }
                            }
                        }
                    },
                    total: {
                        bsonType: 'double',
                        description: 'must be a double and is required'
                    },
                    date: {
                        bsonType: 'date',
                        description: 'must be a date and is required'
                    },
                    status: {
                        bsonType: 'string',
                        description: 'must be a string and is required'
                    }
                }
            }
        }
    },

    cartSchema: {
        validator: {
            $jsonSchema: {
                bsonType: 'object',
                title: 'cart object schema',
                required: ['client', 'products', 'total'],
                properties: {
                    client: {
                        bsonType: 'objectId',
                        description: 'must be an objectId and is required'
                    },
                    products: {
                        bsonType: 'array',
                        description: 'must be an array and is required',
                        items: {
                            bsonType: 'object',
                            properties: {
                                productId: {
                                    bsonType: 'objectId',
                                    description: 'must be an objectId and
is required'
                                },
                                amount: {
                                    bsonType: 'int',
                                    description: 'must be an integer and is
required'
                                }
                            }
                        }
                    },
                    total: {
                        bsonType: 'double',
                        description: 'must be a double and is required'
                    },

                }
            }
        }
    }
}
```

2. Для работы с тестовыми данными создадим блок **testDataUtils**

```
const testDataUtils = {
    idStore: {},
    initialized: false,

    _canInit: () => { return collectionUtils.initialized; },
    _initProductTypes: function () {
        const productTypesIds = {}

        const productTypes = ["Electronics", "Clothes", "Food"];

        productTypes.forEach((name) => {
            let id = crudUtils.createProductType(name);
            productTypesIds[name] = id
        });

        this.idStore.productTypes = productTypesIds;
    },
    _initProducts: function () {
        const productsIds = {}
        const productTypes = this.idStore.productTypes;

        const products = [
            { name: "iPhone", price: new Double(1000), amount: 10, type:
productTypes.Electronics },
            { name: "Samsung", price: new Double(900), amount: 10, type:
productTypes.Electronics },
            { name: "T-shirt", price: new Double(10), amount: 50, type:
productTypes.Clothes },
            { name: "Pants", price: new Double(20), amount: 15, type:
productTypes.Clothes },
            { name: "Bread", price: new Double(2), amount: 100, type:
productTypes.Food },
            { name: "Milk", price: new Double(3), amount: 70, type:
productTypes.Food }
        ];


        products.forEach((product) => {
            let id = crudUtils.createProduct(product.name, product.price,
product.amount, product.type);
            productsIds[product.name] = id
        });

        this.idStore.products = productsIds;
    },
    _initClients: function () {
        const clientsIds = {}

        const clients = [
            { name: "John Doe", email: "jhon.doe@gmail.com", phone: "123-
456-7890" },
            { name: "Jane Doe", email: "jane.doe@mail.ru", phone: "098-765-
4321" },
            { name: "Ivan Ivanov", email: "i.ivan@temp.ru", phone: "123-
459-9999" },
        ]

        clients.forEach((client) => {
            let id = crudUtils.createClient(client.name, client.email,
client.phone);
            clientsIds[client.name] = id

        });

        this.idStore.clients = clientsIds;
    },
    init: function () {
        if (!this._canInit()) {
            console.error("Can't init test data. Collections not
initialized.");
        }

        if (this.initialized) {
            console.error("Test data already initialized.");
        }

        this._initProductTypes();
        this._initProducts();
        this._initClients();
    }
}
```

3. Реализуем все необходимые запросы к базе в блоке **taskQuery**

```
const taskQuery = {
    getAllProductTypes: function () { // Получение списка всех категорий
        return db.productTypes.find();
    },
    getProductsByType: function (typeName) { // Получение списка товаров по
категории
        return db.products.find(
            {
                type: db.productTypes.find({ name: typeName
}).toArray()[0]._id
            }
        );
    },
    getProductByName: function (productName) { // Поиск продукта по
названию
        return db.products.find({ name: productName });
    },
        addProductToCart: function (clientId, productId, amount) { // 
    Добавление продукта в корзину клиента
            let product = db.products.find({ _id: productId }).toArray()[0];

            return db.carts.updateOne(
                { client: clientId },
                {
                    $push: { products: { productId: productId, amount: amount } 
    },
                    $inc: { total: Double(product.price * amount) }
                },
            );
        },
        clearCart: function (clientId) { // Очистка корзины
            return db.carts.updateOne(
                { client: clientId },
                {
                    $set: { products: [], total: new Double(0) }
                },
            );
        },
        createOrderFromCart: function (clientId) { // Создание заказа
            let cart = db.carts.find({ client: clientId }).toArray()[0];

            let res = crudUtils.createOrder(clientId, cart.products, 
    cart.total);
            this.clearCart(clientId);

            return res;
        },
        getOrdersByClient: function (clientId) { // Получение списка заказов по 
    клиенту
            return db.orders.find({ client: clientId });
        },
        setOrderStatus: function (orderId, status) { // Установка статуса 
    заказа
            return db.orders.updateOne(
                { _id: orderId },
                { $set: { status: status } }
            );
        },
        getTopProducts: function (limit) { // Получение списка топ-продаж за 
    последние месяцы с учетом цены и количества проданных товаров.
            return db.orders.aggregate([
                {
                    $match: {
                        date: {
                            $gte: new Date(new Date().getDate() - 30)
                        }
                    }
                },
            {
                $unwind: "$products"
            },
            {
                $group: {
                    _id: "$products.productId",
                    amount: { $sum: "$products.amount" },
                    price: { $avg: "$total" }
                }
            },
            {
                $sort: {
                    amount: -1,
                    price: -1
                }
            },
            {
                $limit: limit
            }
        ]);
    },
    getTopClients: function (orders_count) { // Получение списка клиентов, которые сделали более чем N покупок в последнее время.
        return db.orders.aggregate([
            {
                $match: {
                    date: {
                        $gte: new Date(new Date().getDate() - 30)
                    }
                }
            },
            {
                $group: {
                    _id: "$client",
                    orders_count: { $sum: 1 }
                }
            },
            {
                $match: {
                    count: {
                        $gt: orders_count
                    }
                }
            },
            {
                $lookup: {
                    from: "clients",
                    localField: "_id",
                    foreignField: "_id",
                    as: "client"
                }
            },
            {
                $unwind: "$client"
            },
            {
                $project: {
                    _id: 0,
                    orders_count: 1,
                    client: 1
                }
            }
        ]);
    },
    getTopProductTypes: function (days) { // Получите какие категории
товаров пользовались спросом в заданный срок.
        return db.orders.aggregate([
            {
                $match: {
                    date: {
                        $gte: new Date(new Date().getDate() - days)
                    }
                }
            },
            {
                $unwind: "$products"
            },
            {
                $lookup: {
                    from: "products",
                    localField: "products.productId",
                    foreignField: "_id",
                    as: "product"
                }
            },
            {
                $unwind: "$product"
            },
            {
                $lookup: {
                    from: "productTypes",
                    localField: "product.type",
                    foreignField: "_id",
                    as: "productType"
                }
            },
            {
                $unwind: "$productType"
            },
            {
                $group: {
                    _id: "$productType.name",
                    count: { $sum: 1 }
                }
            },
            {
                $sort: { count: -1 }
            }
        ]);
    },
    getNotSoldProductsInDate: function (date) { // Какие товары не были
проданы в какую-то дату.
        return db.products.aggregate([
            {
                $lookup: {
                    from: "orders",
                    localField: "_id",
                    foreignField: "products.productId",
                    as: "orders"
                }
            },
            {
                $match: {
                    orders: {
                        $not: {
                            $elemMatch: {
                                date: {
                                    $gte: new Date(date),
                                    $lt: new Date(date).setDate(new
Date(date).getDate() + 1)
                                }
                            }
                        }
                    }
                }
            },
            {
                $match: {
                    orders: { $size: 0 }
                }
            },
            {
                $project: {
                    _id: 1,
                    name: 1,
                    price: 1,
                    type: 1
                }
            }
        ]);
    }
}
```

4. Реализуем функцию **initUsers** для создания ролей и пользователей

```
function initUsers() {
    db.createRole( // Создание роли для просмотра продуктов
        {
            role: "products_viewer",
            privileges: [
                {
                    actions: ["find"],
                    resource: { db: "shop", collection: "products" }
                },
                {
                    actions: ["find"],
                    resource: { db: "shop", collection: "productTypes" }
                }
            ]
        }
    )

    db.createRole( // Создание роли администратора
        {
            role: "admin",
            privileges: [
                {
                    actions: ["insert", "update", "remove"],
                    resource: { db: "shop", collection: "" }
                }
            ],
            roles: [
                "products_viewer",
            ]
        }
    )

    db.createRole( // Создание роли менеджера
        {
            role: "manager",
            privileges: [
                {
                    actions: ["insert", "update", "remove"],
                    resource: { db: "shop", collection: "products" }
                },
                {
                    actions: ["insert", "update", "remove"],
                    resource: { db: "shop", collection: "productTypes" }
                }
            ],
            roles: [
                "products_viewer",
            ]
    }
)

db.createRole( // Создание роли клиента
    {
        role: "client",
        privileges: [
            {
                actions: ["find", "insert", "update"],
                resource: { db: "shop", collection: "carts" }
            },
            {
                actions: ["find", "insert"],
                resource: { db: "shop", collection: "orders" }
            }
        ],
        roles: [
            "products_viewer",
        ]
    }
)

db.createUser(
    {
        user: "some_admin_1",
        pwd: "changeme",
        roles: [
            {
                role: "admin",
                db: "shop"
            }
        ]
    }
)

db.createUser(
    {
        user: "manager",
        pwd: "changeme",
        roles: [
            {
                role: "manager",
                db: "shop"
            }
        ]

    }
)

db.createUser(
    {
            user: "client",
            pwd: "changeme",
            roles: [
                {
                    role: "client",
                    db: "shop"
                }
            ]
        }
    )

    db.createUser(
        {
            user: "viewer",
            pwd: "changeme",
            roles: [
                {
                    role: "products_viewer",
                    db: "shop"
                }
            ]
        }
    )

}
```

5. Создадим функцию **testSchemas** для проведения тестирования валидаторов на коллекциях.

```
function testSchemas() {
    // 1
    try {
        db.products.insertOne({
            name: "test",
            price: Double(100),
            type: "test", // Тип продукта должен быть ObjectId. Ожидаем
ошибку
            amount: 100
        })
    } catch (error) {
        if (error.errInfo == null) {
            return Error("Тест 1 не пройден")
        }
    }

    // 2
    try {
        db.productTypes.insertOne({
            name: 123 // Название должно быть строкой. Ожидаем ошибку
        })
    } catch (error) {
        if (error.errInfo == null) {
            return Error("Тест 2 не пройден")
        }
    }


    // 3
    try {
        db.clients.insertOne({
            name: "test",
            email: "test",
            phone: "777-999-999", // Номер телефона должен соответствовать
паттерну '^[0-9]{3}-[0-9]{3}-[0-9]{4}$'. Ожидаем ошибку
        })
    } catch (error) {
        if (result.errInfo == null) {
            return Error("Тест 3 не пройден")
        }
    }

    // 4
    try {
        result = db.orders.insertOne({
            date: "01-01-2020",
            products: [
                {
                    productId: "test", // ID продукта должен быть ObjectId.
Ожидаем ошибку
                    amount: 100
                }
            ],
            client: ObjectId(1),
            status: "test",
            total: Double(100)
        })
    } catch (error) {

        if (result.errInfo == null) {
            return Error("Тест 4 не пройден")
        }
    }

    return "Тесты пройдены"

}
```