**# HIS (Hospital Information System) - Документация проекта**

**## Обзор проекта**

HIS - это многотенантная система управления больничной информацией, построенная на архитектуре микросервисов с использованием NestJS, MongoDB и RabbitMQ. Система поддерживает изоляцию данных между различными медицинскими организациями (тенантами) и включает в себя аудит, ограничения ресурсов и систему аутентификации.

**## Архитектура системы**

Проект состоит из трех основных компонентов:

**### 1. \*\*His\*\* (Основное приложение)**

- **\*\*Порт\*\***: 3000

- **\*\*Назначение\*\***: Основной API сервер с многотенантной архитектурой

- **\*\*База данных\*\***: MongoDB с отдельными базами для каждого тенанта

**### 2. \*\*Producer\*\* (Продюсер сообщений)**

- **\*\*Назначение\*\***: Микросервис для отправки сообщений в RabbitMQ

- **\*\*Функции\*\***: Управление заказами и отправка событий

**### 3. \*\*Consumer\*\* (Потребитель сообщений)**

- **\*\*Назначение\*\***: Микросервис для обработки сообщений из RabbitMQ

- **\*\*Функции\*\***: Обработка заказов и ведение статистики

**## Основные модули системы His**

**### 🏢 Tenants Module (Управление тенантами)**

**\*\*Файлы\*\***: `src/tenants/`

**\*\*Функциональность\*\***:

- Создание новых медицинских организаций (компаний)

- Генерация уникальных идентификаторов тенантов (nanoid)

- Автоматическое создание пользователя-администратора

- Создание секретных ключей для JWT токенов

**\*\*API Endpoints\*\***:

```

POST /tenants/create-company - Создание новой компании

```

**\*\*Схема данных\*\***:

```typescript

{

  companyName: string,  // Название компании

  tenantId: string      // Уникальный ID тенант��

}

```

**### 👤 Users Module (Управление пользователями)**

**\*\*Файлы\*\***: `src/users/`

**\*\*Функциональность\*\***:

- Создание пользователей с привязкой к тенанту

- Хеширование паролей (bcrypt)

- Поиск пользователей по email

**\*\*Схема данных\*\***:

```typescript

{

  name: string,

  email: string,

  password: string,    // Хешированный пароль

  tenantId: string     // Привязка к тенанту

}

```

**### 🔐 Auth Module (Аутентификация)**

**\*\*Файлы\*\***: `src/auth/`

**\*\*Функциональность\*\***:

- Аутентификация пользователей

- Генерация JWT токенов с тенант-специфичными секретными ключами

- Шифрование/дешифрование секретных ключей

- Создание секретных ключей для новых тенантов

**\*\*API Endpoints\*\***:

```

POST /auth/login - Вход в систему

```

**\*\*Процесс аутентификации\*\***:

1. Проверка существования пользователя

2. Сравнение паролей

3. Получение секретного ключа дл�� тенанта

4. Генерация JWT токена

5. Возврат токена и tenantId

**### 🏥 Patients Module (Управление пациентами)**

**\*\*Файлы\*\***: `src/patients/`

**\*\*Функциональность\*\***:

- Получение списка пациентов для конкретного тенанта

- Изоляция данных между тенантами

- Защита авторизацией

**\*\*API Endpoints\*\***:

```

GET /patients - Получение списка пациентов (требует аутентификации)

```

**\*\*Схема данных\*\***:

```typescript

{

  name: string,

  surname: string,

  age: string

}

```

**### 📊 Limits Module (Управление лимитами)**

**\*\*Файлы\*\***: `src/limits/`

**\*\*Функциональность\*\***:

- Установка лимитов для тенантов (документы, размер данных, запросы)

- Мониторинг использования ресурсов

- Блокировка при превышении лимитов

**\*\*API Endpoints\*\***:

```

GET /limits/:tenantId - Получение лимитов тенанта

PUT /limits/:tenantId - Установка лимитов тенанта

GET /limits/usage/:tenantId - Статистика использования

```

**\*\*Типы лимитов\*\***:

- `maxDocuments` - Максимальное количество документов

- `maxDataSizeKB` - Максимальный размер данных в KB

- `monthlyQueries` - Количество запросов в месяц

**### 📝 Audit Module (Аудит)**

**\*\*Файлы\*\***: `src/audit/`

**\*\*Функциональность\*\***:

- Логирование всех HTTP запросов

- Отправка событий аудита в RabbitMQ

- Санитизация чувствительных данных (пароли, токены)

**\*\*Данные аудита\*\***:

- Временная метка

- ID пользователя и тенанта

- HTTP метод и путь

- Статус код и время выполнения

- IP адрес и User-Agent

- Тело запроса (с санитизацией)

**### 🔄 Proxy Module (Прокси)**

**\*\*Файлы\*\***: `src/proxy/`

**\*\*Функциональность\*\***:

- Проверка состояния системы

- Базовый health check endpoint

**\*\*API Endpoints\*\***:

```

GET /proxy/health - Проверка состояния

```

**## Многотенантная архитектура**

**### Принцип работы**

1. **\*\*Изоляция на уровне базы данных\*\***: Каждый тенант получает отдельную базу данных MongoDB

2. **\*\*Именование баз\*\***: `tenant\_{tenantId}`

3. **\*\*Middleware проверки\*\***: Проверяет наличие и валидность `X-TENANT-ID` заголовка

4. **\*\*Динамическое подключение\*\***: Автоматическое переключение между базами тенантов

**### Компоненты многотенантности**

**#### TenantMiddleware**

- Извлекает `tenantId` из заголовка `X-TENANT-ID`

- Проверяет существование тенанта

- Добавляет `tenantId` в объект запроса

**#### TenantConnectionProvider**

- Создает подключения к базам данных тенантов

- Использует request-scoped провайдеры

- Автоматически переключается между базами

**#### TenantAuthenticationGuard**

- Проверяет JWT токены с тенант-специфичными ключами

- Обеспечивает безопасность на уровне тенанта

**## Система безопасности**

**### Шифрование**

- **\*\*Алгоритм\*\***: Cryptr для симметричного шифрования

- **\*\*Применение\*\***: Секретные ключи JWT токено��

- **\*\*Ключ шифрования\*\***: Хранится в переменных окружения

**### JWT токены**

- **\*\*Уникальные ключи\*\***: Каждый тенант имеет свой секретный ключ

- **\*\*Время жизни\*\***: 10 часов

- **\*\*Содержимое\*\***: userId в payload

**### Хеширование паролей**

- **\*\*Алгоритм\*\***: bcrypt

- **\*\*Раунды\*\***: 10

**## Микросервисная архитектура**

**### Producer Service**

**\*\*Функции\*\***:

- Отправка заказов в RabbitMQ

- Получение списка заказов

- Эмиссия событий `order-placed`

**\*\*API\*\***:

```

POST /orders - Создание заказа

GET /orders - Получение заказов

```

**### Consumer Service**

**\*\*Функции\*\***:

- Обработка событий `order-placed`

- Хранение заказов в памяти

- Предоставление API для получения заказов

**\*\*Обработчики\*\***:

- `handleOrderPlaced` - Обработка новых заказов

- `getOrders` - Возврат списка заказов

**## Конфигурация и развертывание**

**### Переменные окружения (.env)**

```

PORT=3000

DATABASE\_CONNECTION\_STRING=mongodb://localhost:27017/his

SECURITY\_ENCRYPTION\_SECRET\_KEY=your-encryption-key

```

**### Зависимости**

**\*\*Основные\*\***:

- NestJS (фреймворк)

- Mongoose (MongoDB ODM)

- JWT (аутентификация)

- bcrypt (хеширование)

- nanoid (генерация ID)

- cryptr (шифрование)

- RabbitMQ (сообщения)

**### Запуск системы**

```bash

# Основное приложение

cd His

npm run start:dev

# Producer

cd producer

npm run start:dev

# Consumer

cd consumer

npm run start:dev

```

**## API Документация**

**### Создание компании**

```http

POST /tenants/create-company

Content-Type: application/json

{

  "companyName": "Городская больница №1",

  "user": {

    "name": "Иван Иванов",

    "email": "admin@hospital1.ru",

    "password": "securepassword"

  }

}

```

**### Аутентификация**

```http

POST /auth/login

Content-Type: application/json

{

  "email": "admin@hospital1.ru",

  "password": "securepassword"

}

```

**### Получение пациентов**

```http

GET /patients

X-TENANT-ID: {tenantId}

Authorization: Bearer {jwt-token}

```

**### Управление лимитами**

```http

PUT /limits/{tenantId}

Authorization: Bearer {jwt-token}

Content-Type: application/json

{

  "maxDocuments": 5000,

  "maxDataSizeKB": 102400,

  "monthlyQueries": 10000

}

```

**## Особенности реализации**

**### Преимущества**

1. **\*\*Полная изоляция данных\*\*** между тенантами

2. **\*\*Масштабируемость\*\*** - каждый тенант может иметь свою базу

3. **\*\*Безопасность\*\*** - уникальные ключи шифрования для каждого тенанта

4. **\*\*Аудит\*\*** - полное логирование всех операций

5. **\*\*Контроль ресурсов\*\*** - система лимитов предотвращает злоупотребления

**### Потенциальные улучшения**

1. **\*\*Валидация данных\*\*** - добавить DTO валидацию

2. **\*\*Обработка ошибок\*\*** - централизованная система обработки ошибок

3. **\*\*Кеширование\*\*** - Redis для улучшения производительности

4. **\*\*Мониторинг\*\*** - интеграция с системами мониторинга

5. **\*\*Тестирование\*\*** - покрытие unit и integration тестами

**## Заключение**

HIS представляет собой современную многотенантную систему управления больничной информацией с акцентом на безопасность, изоляцию данных и масштабируемость. Архитектура позволяет легко добавлять новые медицинские организации без влияния на существующие данные и обеспечивает высокий уровень безопасности через индивидуальные ключи шифрования и изолированные базы данных.