МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 42

КУРСОВАЯ РАБОТА (ПРОЕКТ)
ЗАЩИЩЕНА С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

ассистент		Шевяков Д. О.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ

по курсу: ИТ-модуль "Интернет вещей"

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 2146		Богодухов М. Ю.
номер группы	подпись, дата	инициалы, фамилия

Санкт-Петербург
2024

СОДЕРЖАНИЕ

1 Задание

Согласно выбранной или согласованной с преподавателем теме, представить проект и разработать систему интернета вещей. Система должна быть основана на веб-приложении с использованием фреймворка Flask или аналогичного ему. В системе должны применяться функции для обмена данными, реализованные в виде URL-запросов между различными частями системы.

Система должна быть разработана в парадигме объектно-ориентированного программирования и включать в себя классы, их поведение, атрибуты и иерархию. Классы должны быть реализованы в отдельном модуле. Если система включает в себя элементы, которые можно сгруппировать по функционалу, каждая из таких групп должна быть реализована в отдельном модуле.

Система должна быть разработана с использованием гибких подходов. Отдельные элементы системы должны иметь возможность работать независимо друг от друга. Система должна иметь возможность масштабирования и добавления новых сущностей без существенной модификации кода программы. В системе не должно присутствовать одинаковых фрагментов кода. Если какой-либо код используется в программе более одного раза, он должен быть оформлен в виде функции или метода. Система должна быть реализована таким образом, чтобы работоспособность каждой из функций можно было бы легко протестировать без модификации кода программы.

Система должна включать в себя, как минимум один графический интерфейс пользователя, с помощью которого можно получить доступ ко всем функциям системы. При оценке интерфейса внимание будет обращаться на функциональность, а не на привлекательность с точки зрения дизайна.

Система должна включать в себя модуль сбора и сохранения данных системы. Модуль сбора данных должен работать независимо от других модулей и частей системы, а также собирать и сохранять данные с заданным периодом. Данный модуль должен иметь функцию предварительной проверки и обработки данных перед

сохранением. Должны быть продуманы, как механизмы отказа сохранения данных, так и механизмы преобразования данных к требуемому виду или формату.

Система должна включать в себя модуль анализа сохраненных данных. Модуль анализа может включать в себя, как статистические характеристики системы, так и предиктивные модели, основанные на анализе имеющихся данных.

2 Проект системы

1. Система:

1.1. Абстрактный класс MedicalDevice:

Свойства:

device_id (идентификатор медицинского устройства)

name (наименование устройства)

status (статус устройства: включено/выключено)

1.2. Класс PatientBed (наследуется от MedicalDevice):

Дополнительные свойства:

patient_id (идентификатор пациента)

bed_number (номер кровати)

1.3. Класс VitalMonitor (наследуется от MedicalDevice):

Дополнительные свойства:

heart_rate (пульс)

blood_pressure (кровяное давление)

1.4. Класс MedicationDispenser (наследуется от MedicalDevice):

Дополнительные свойства:

medication_list (список предписанных лекарств)

1.5. Класс PatientInfoSystem:

Обработка и передача данных между медицинскими устройствами и интерфейсами.

Методы:

update_patient_data(patient_id, vital_signs, medication_status) (обновление данных о пациенте)

1.6. Класс MedicalDatabase:

Хранение данных о медицинских устройствах и состояниях пациентов.

Сущности:

medical_devices (список установленных медицинских устройств)

patient_records (данные о пациентах)

Связи между элементами:

Медицинские устройства (PatientBed, VitalMonitor, MedicationDispenser) наследуют от абстрактного класса MedicalDevice.

PatientInfoSystem взаимодействует с устройствами для обновления данных пациента.

MedicalDatabase хранит данные об устройствах и состояниях пациентов.

2. Интерфейсы:

2.1. Patient Interface:

Отображает основные данные о состоянии кровати (занята/свободна) и витальных показателях пациента.

Позволяет пациенту вызвать медсестру и контролировать некоторые параметры.

2.2. Nurse Interface:

Предоставляет медсестре общий обзор по всем кроватям и важным медицинским устройствам.

Позволяет устанавливать параметры устройств, следить за витальными показателями и управлять предписанными лекарствами.

Общая концепция:

Patient Interface скрывает сложные медицинские параметры, предоставляя пациенту только понятные данные.

Nurse Interface предоставляет полный доступ к медицинским данным для медсестры и врача.

MedicalDatabase хранит все данные, но через интерфейсы предоставляется разный уровень доступа.

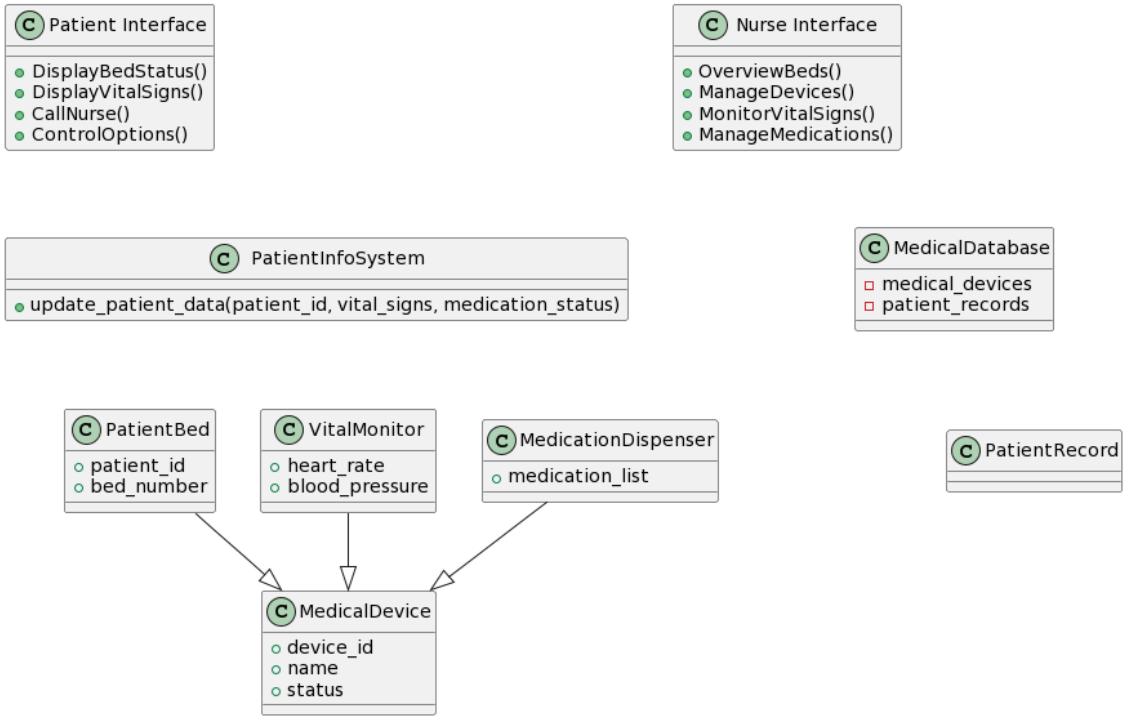

Рисунок 1 – UML диаграмма классов

3 Описание объектной модели

Представленная объектная модель описывает основные классы и их взаимосвязи в системе медицинского оборудования и управления данными о пациентах. Ниже представлено подробное описание каждого класса:

3.1 Классы

3.1.1 Класс MedicalDevice

Этот абстрактный класс представляет базовое медицинское устройство. Включает в себя свойства, общие для всех устройств, такие как уникальный идентификатор (device_id), наименование (name) и статус (status), который указывает, включено ли устройство или выключено.

3.1.2 Подклассы класса MedicalDevice

3.1.2.1 Класс PatientBed

Этот класс представляет кровать для пациента. Он наследует все свойства класса MedicalDevice и добавляет дополнительные атрибуты, такие как идентификатор пациента (patient_id) и номер кровати (bed_number).

3.1.2.2 Класс VitalMonitor

Класс VitalMonitor представляет мониторинг важных показателей пациента, таких как пульс и кровяное давление. Он также наследует свойства от класса MedicalDevice и содержит дополнительные атрибуты, такие как показатель пульса (heart_rate) и данные о кровяном давлении (blood_pressure).

3.1.2.3 Класс MedicationDispenser

Этот класс представляет устройство для выдачи лекарств. Как и другие подклассы, он наследует свойства от MedicalDevice и содержит список предписанных лекарств (medication_list).

3.1.3 Дополнительные классы

3.1.3.1 Класс PatientInfoSystem

Этот класс отвечает за обработку и передачу данных между медицинскими устройствами и интерфейсами. Его методы включают функцию обновления данных о пациенте на основе витальных показателей и статуса приема лекарств.

3.1.3.2 Класс MedicalDatabase

Класс MedicalDatabase отвечает за хранение данных о медицинских устройствах и состояниях пациентов. Он содержит списки установленных медицинских устройств и данные о пациентах.

3.2 Взаимосвязь классов

Подклассы медицинских устройств (PatientBed, VitalMonitor, MedicationDispenser) наследуют свойства и методы от абстрактного класса MedicalDevice.

Класс PatientInfoSystem взаимодействует с медицинскими устройствами для обновления данных о пациентах.

Класс MedicalDatabase хранит и обеспечивает доступ к данным о медицинских устройствах и пациентах.

Эта объектная модель обеспечивает базовую структуру для разработки системы управления медицинскими устройствами и данными о пациентах.

4 Описание объектной модели

4.1 Собираемые данные

Модуль сбора данных собирает широкий спектр информации, необходимой для мониторинга состояния пациентов и функционирования медицинского оборудования.

Это включает:

- Витальные показатели пациента. Включая пульс, кровяное давление, температуру тела и другие важные показатели.
- Состояние медицинских устройств. Такие как текущий статус работы, параметры функционирования и данные о работе устройств.
- Данные о приёме лекарств. Список предписанных препаратов и информацию о приёме лекарств пациентами.
- Информация о пациентах. Включая их персональные данные, медицинскую историю и текущее состояние.

4.2 Формат и периодичность сохранения данных

Собранные данные сохраняются в базе данных системы, где каждый тип информации хранится в соответствующих таблицах. Регулярное резервное копирование и обновление базы данных происходит ежедневно для обеспечения целостности и доступности информации. Периодичность сохранения данных о витальных показателях и состоянии устройств зависит от их частоты обновления и может варьироваться от нескольких секунд до нескольких минут.

4.3 Процедура валидации данных

Перед сохранением в базу данных все данные проходят процесс валидации для обеспечения их корректности и достоверности. Это включает проверку наличия необходимых полей, соответствие формату данных и проведение дополнительных проверок на адекватность и логическую связь с уже существующими данными. В случае обнаружения ошибок или несоответствий система отправляет уведомления о проблеме для оперативного решения.

5 Описание функций анализа сохраненных данных

Расчёт среднего значения сердечного ритма и температуры:

Описание: Эта функция вычисляет среднее значение сердечного ритма и температуры для каждого пациента на основе сохраненных данных.

Применение: Средние значения могут использоваться для отслеживания общего состояния здоровья пациента на протяжении определенного периода времени. Они могут также помочь в диагностике и мониторинге изменений в пациентском состоянии.

Графическое представление данных:

Описание: Функция визуализирует анализированные данные в виде графиков, что облегчает восприятие и понимание результатов анализа.

Применение: Графическое представление данных позволяет врачам и медицинскому персоналу быстро обнаруживать паттерны и тренды в данных пациентов, делать выводы и принимать решения о дальнейших медицинских мероприятиях.

Реализация функций:

```
def receive_temperature_data(self, patient_id):
    data = request.json
    if patient_id:
        data['timestamp'] = datetime.now().isoformat()
        self.temperature_collection.insert_one({'patient_id': patient_id, 'data':
data})
        return jsonify({'status': 'success'})
    else:
        return jsonify({'status': 'error', 'message': 'Patient ID not provided'})

def receive_heart_rate_data(self, patient_id):
    data = request.json
    if patient_id:
        data['timestamp'] = datetime.now().isoformat()
        self.heart_rate_collection.insert_one({'patient_id': patient_id, 'data':
data})
        return jsonify({'status': 'success'})
    else:
        return jsonify({'status': 'error', 'message': 'Patient ID not provided'})

def patient_monitor(self, patient_id):
    temperature_data = list(self.temperature_collection.find({'patient_id':
patient_id}))
    heart_rate_data = list(self.heart_rate_collection.find({'patient_id':
patient_id}))
```

```
    if not temperature_data and not heart_rate_data:
        return 'No data found for this patient'

    temperature_values = []
    heart_rate_values = []
    timestamps_temperature = []
    timestamps_heart_rate = []

    for temp_data in temperature_data:
        temperature_values.append(temp_data['data']['temperature'])
        timestamps_temperature.append(temp_data['data']['timestamp'])

    for hr_data in heart_rate_data:
        heart_rate_values.append(hr_data['data']['heart_rate'])
        timestamps_heart_rate.append(hr_data['data']['timestamp'])

    temperature_fig = go.Figure(
        data=go.Scatter(x=timestamps_temperature, y=temperature_values, mode='lines',
name='Temperature'))
    heart_rate_fig = go.Figure(
        data=go.Scatter(x=timestamps_heart_rate, y=heart_rate_values, mode='lines',
name='Heart Rate'))

    temperature_graph = temperature_fig.to_html(full_html=False)
    heart_rate_graph = heart_rate_fig.to_html(full_html=False)

    avg_temperature = round(sum(temperature_values) / len(temperature_values), 1) if
temperature_values else None
    avg_heart_rate = round(sum(heart_rate_values) / len(heart_rate_values), 1) if
heart_rate_values else None

    return render_template('patient_monitor.html', patient_id=patient_id,
temperature_graph=temperature_graph,
                           heart_rate_graph=heart_rate_graph,
avg_temperature=avg_temperature,
                           avg_heart_rate=avg_heart_rate)
```

Скриншот:

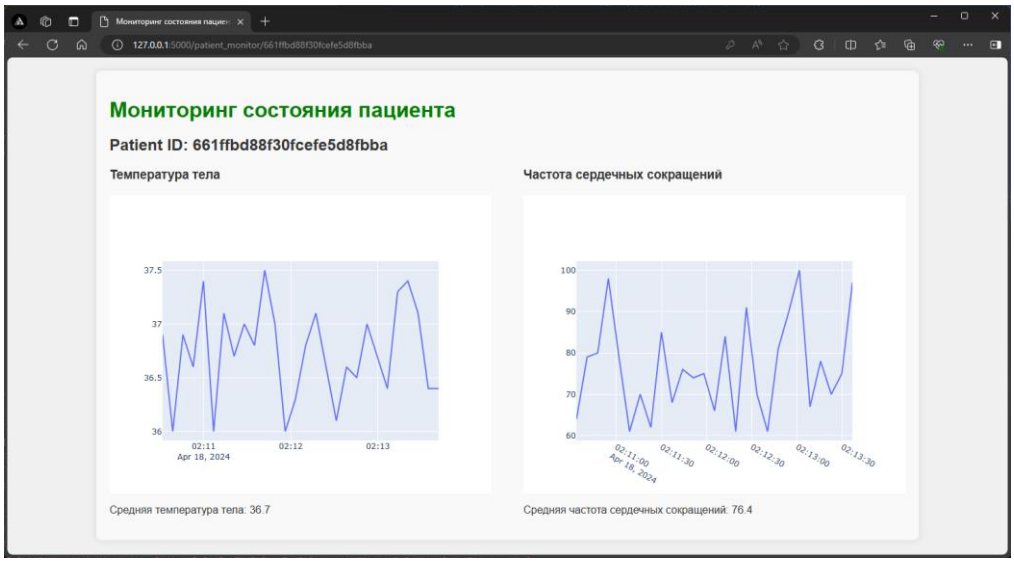

Рисунок 2 – Мониторинг состояния пациента

6 Интерфейсы

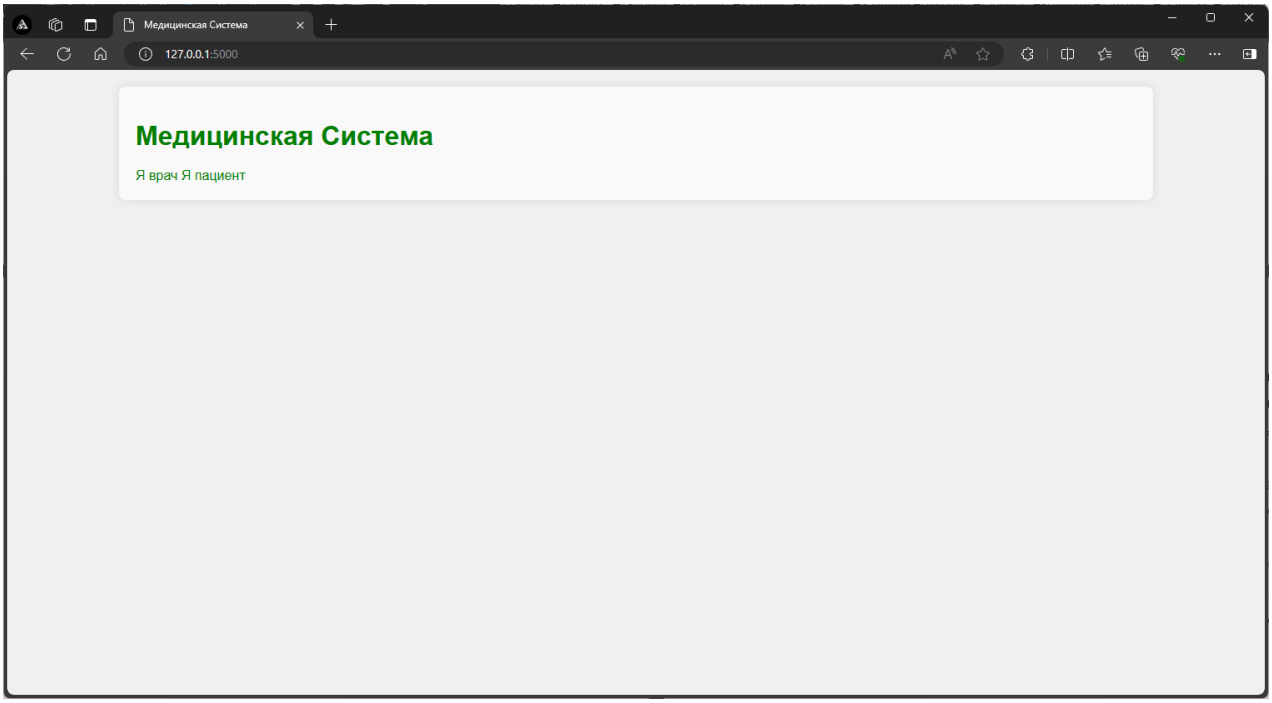

Рисунок 3 – Начальный страница

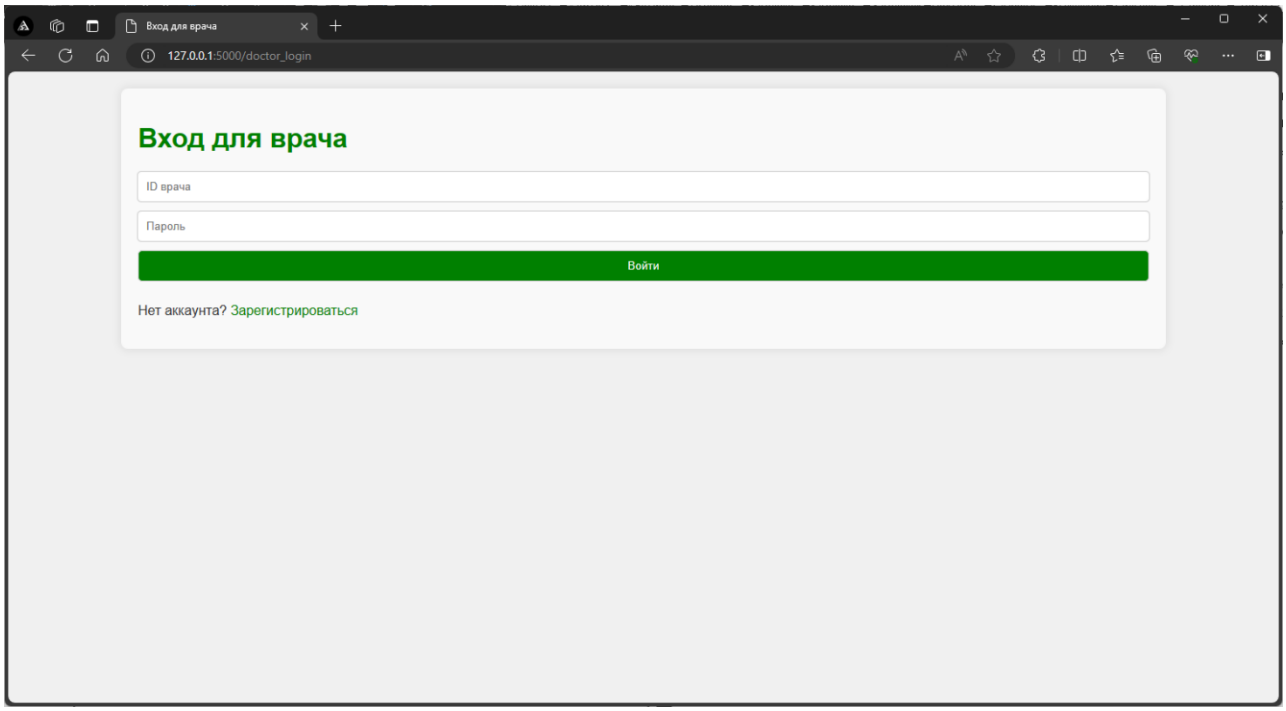

Рисунок 4 – Страница авторизации врача

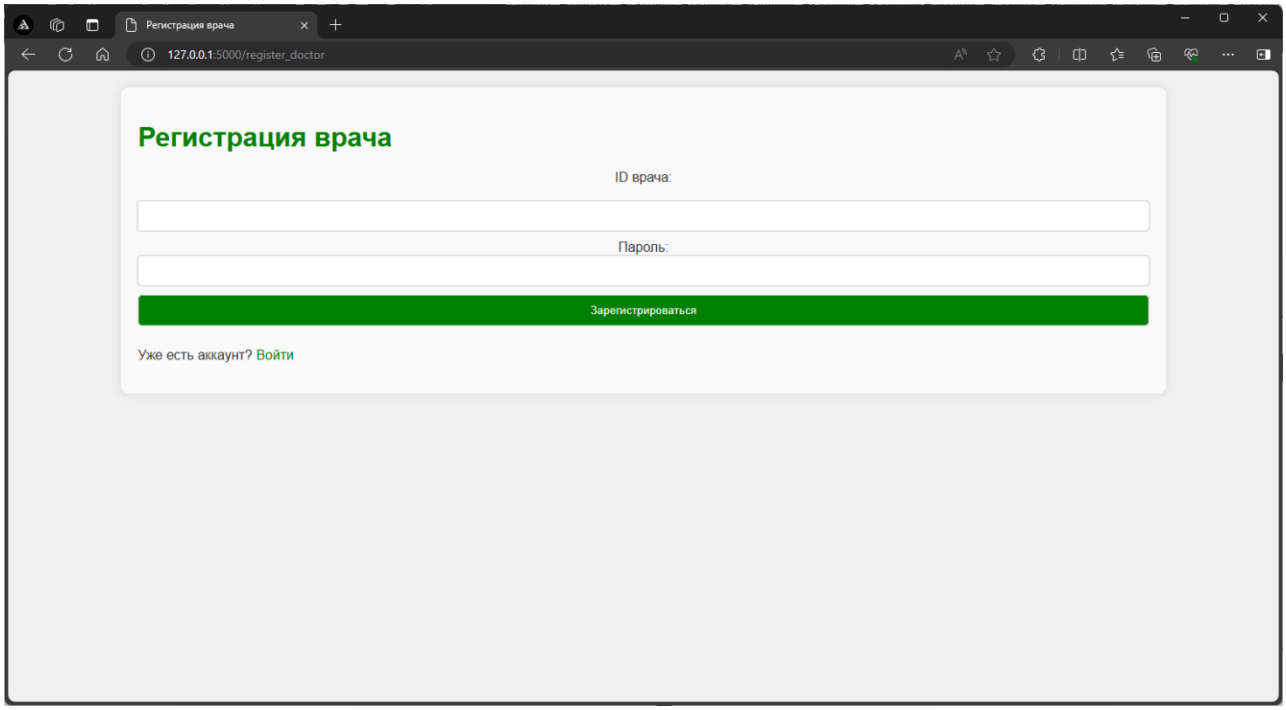

Рисунок 5 – Страница регистрации врача

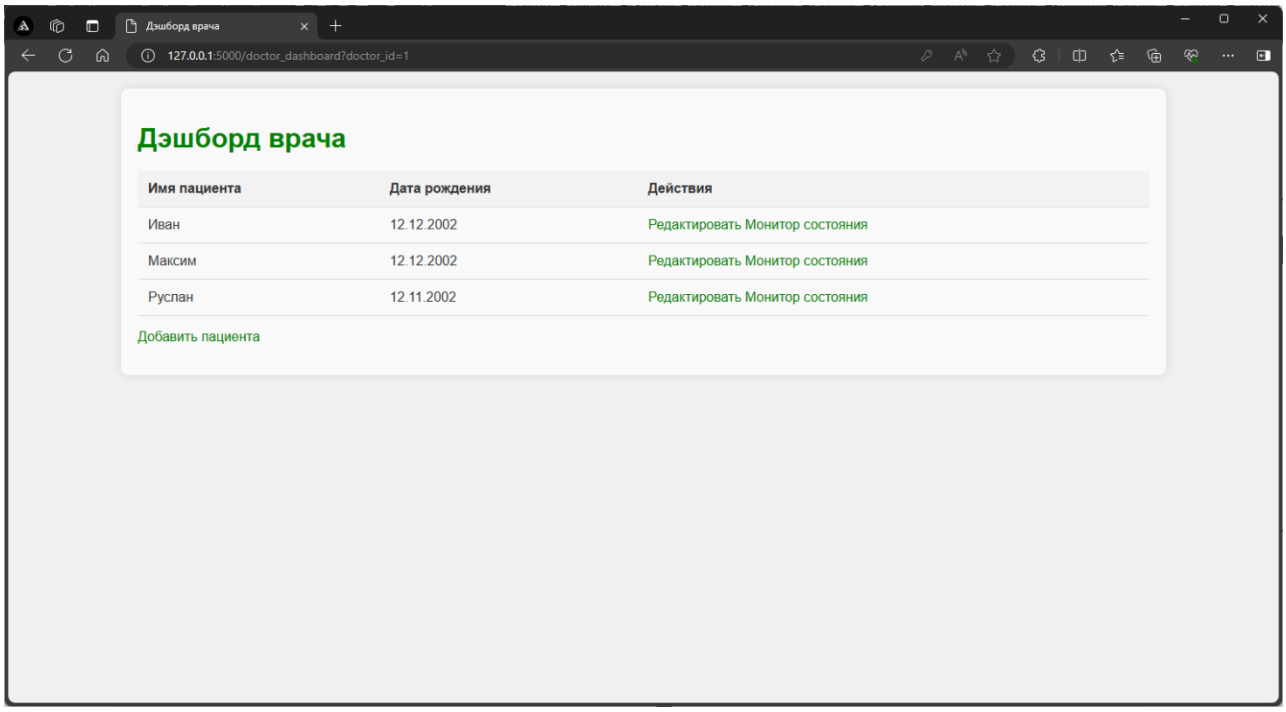

Рисунок 6 – Дэшборд врача после успешной авторизации

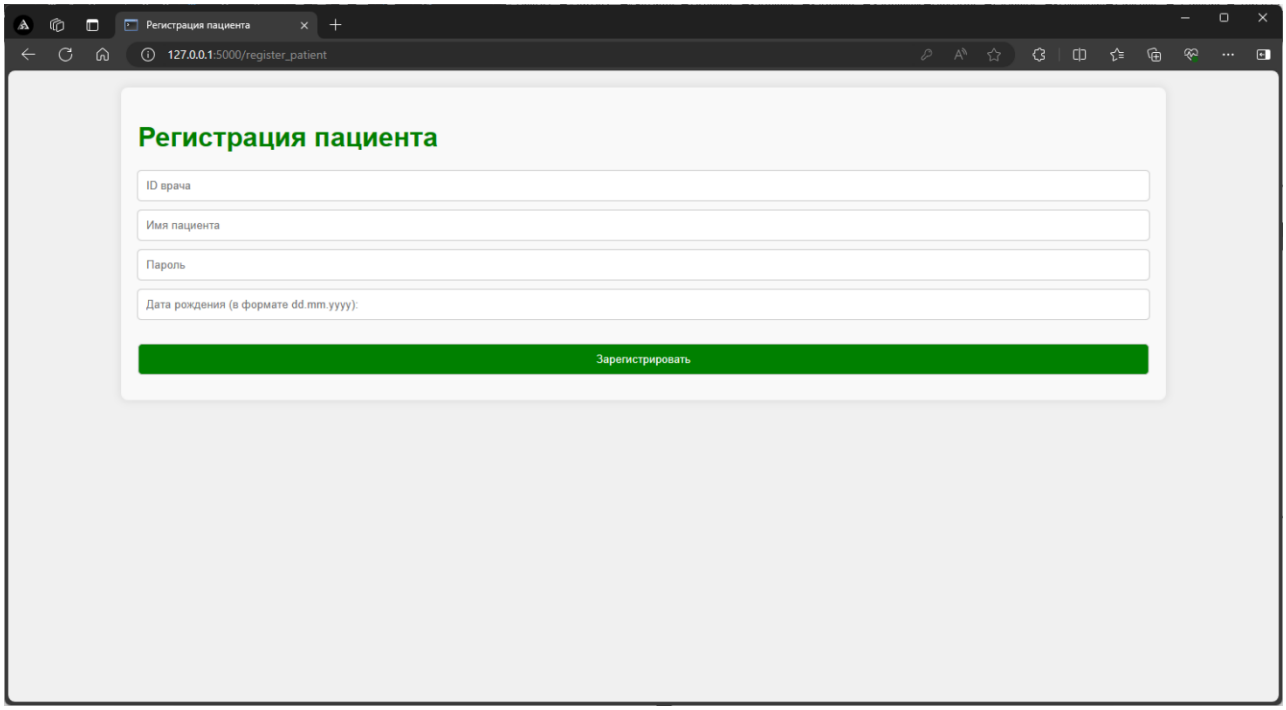

Рисунок 7 – Страница регистрации пациента

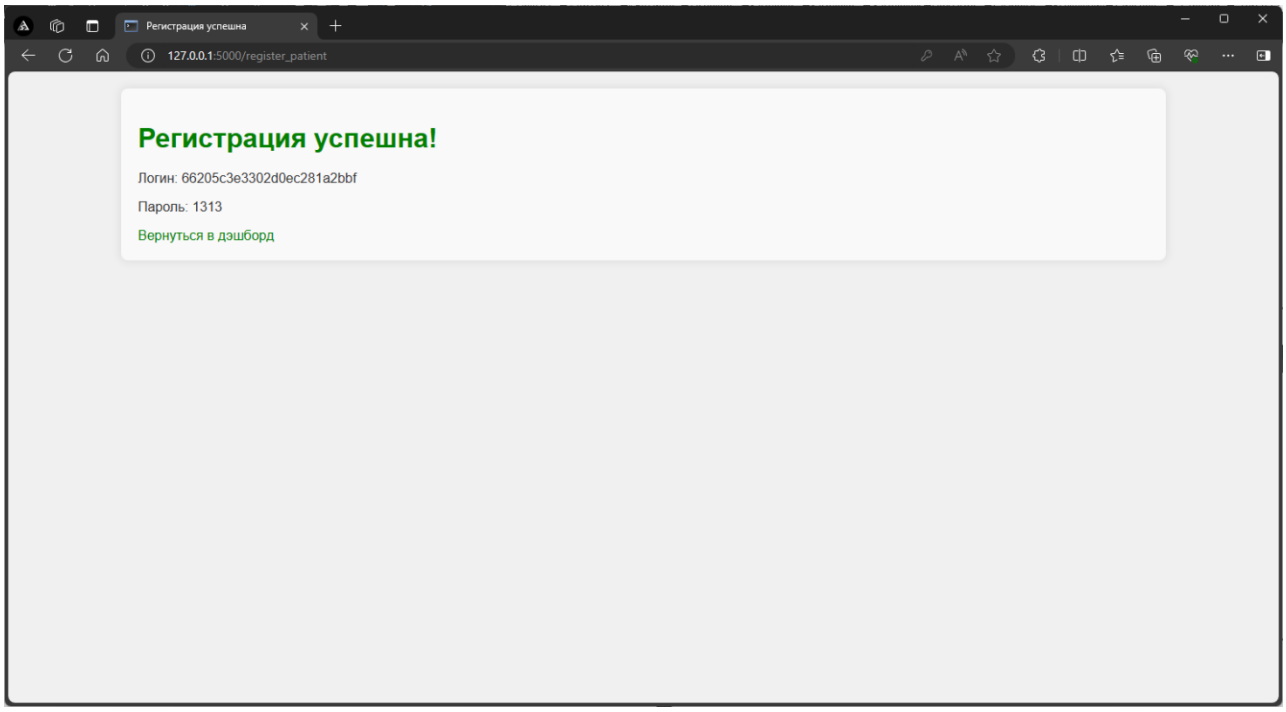

Рисунок 8 – Страница-подтверждение успешной регистрации пациента

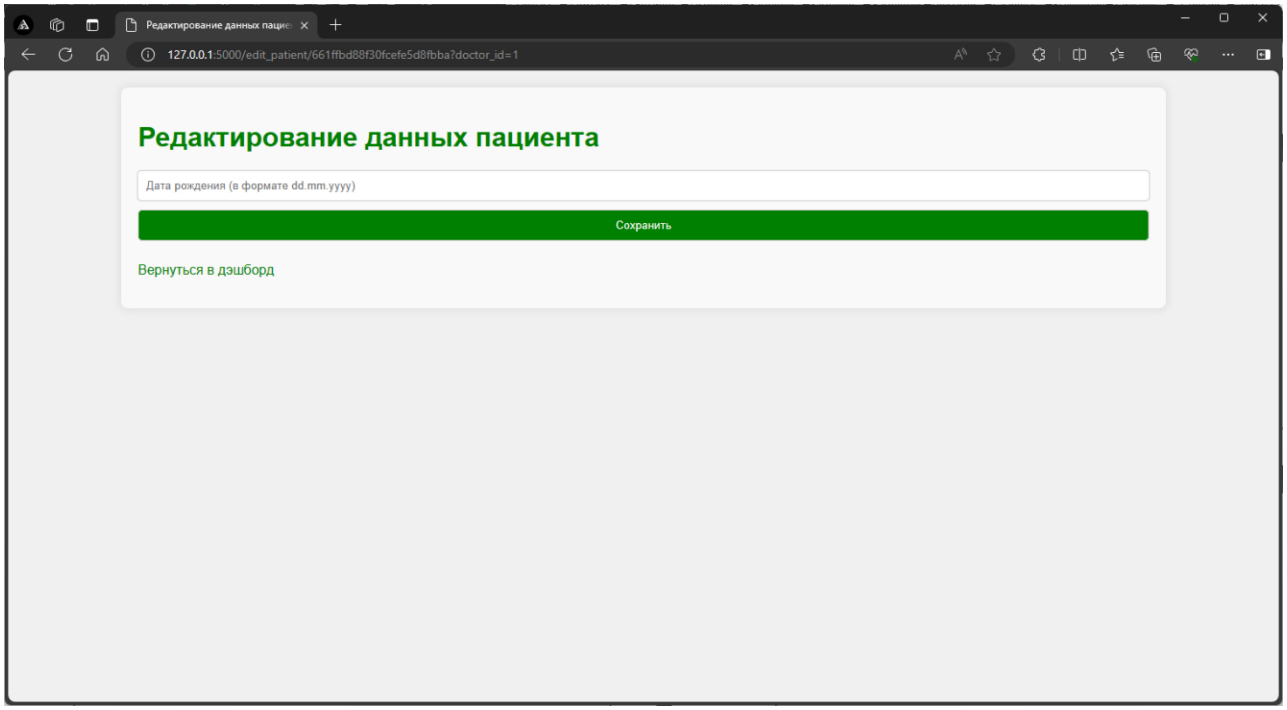

Рисунок 9 – Страница редактирования данных пациента

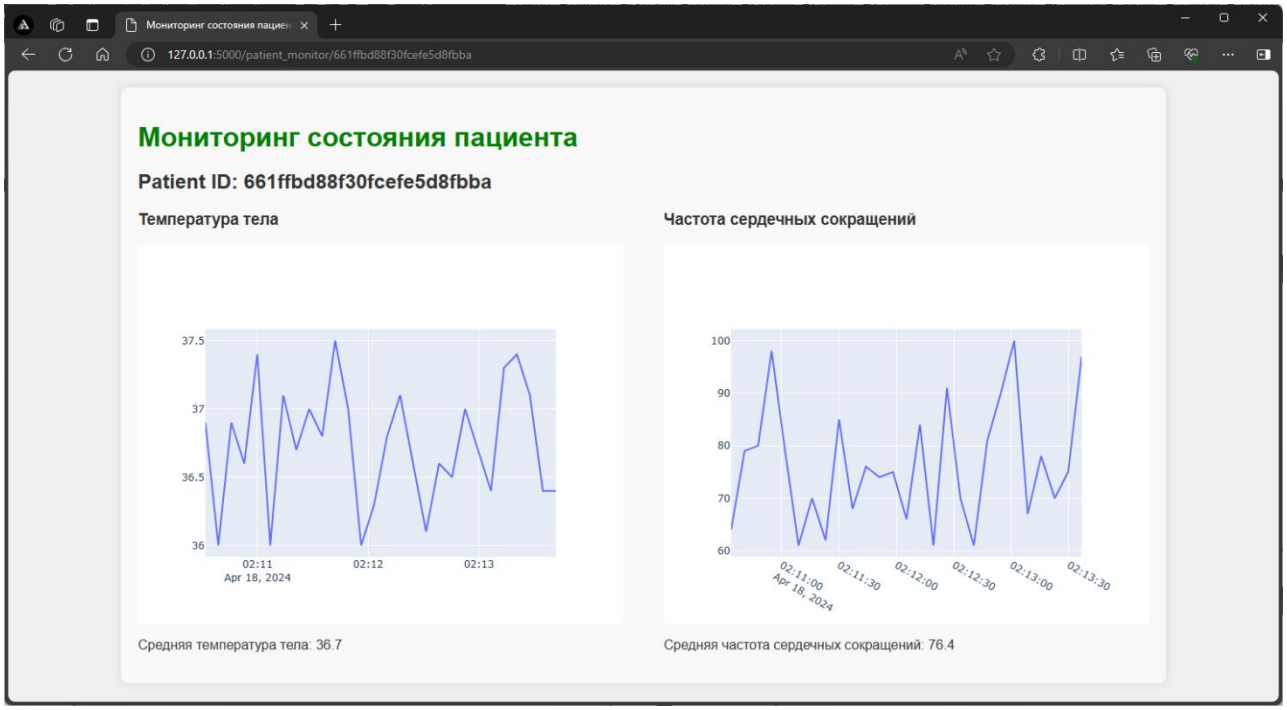

Рисунок 10 – Страница мониторинга состояния пациента

7 Вывод

В ходе выполнения проекта была создана система интернета вещей (IoT), основанная на веб-приложении с использованием фреймворка Flask. Целью работы было применение полученных знаний, умений и навыков для разработки полноценной системы IoT.

Система включает в себя следующие основные компоненты:

1. Классы и объектно-ориентированное программирование: Была реализована иерархия классов для различных сущностей системы, таких как пациенты, врачи и данные с датчиков. Каждый класс обладает атрибутами и методами для обеспечения нужной функциональности.
2. Веб-приложение с использованием Flask: Было разработано веб-приложение, которое позволяет пользователям взаимодействовать с системой через веб-интерфейс. Пользователи могут регистрироваться, входить в систему, просматривать данные и анализировать информацию.
3. Модуль сбора и сохранения данных: Система включает модуль для сбора и сохранения данных с датчиков. Данные собираются с заданным периодом и сохраняются в базе данных для последующего анализа.
4. Модуль анализа сохраненных данных: Был реализован модуль для анализа сохраненных данных. Этот модуль позволяет вычислять статистические характеристики данных, выявлять аномалии и тренды, а также предоставлять информацию для принятия медицинских решений.
5. Гибкий подход и масштабируемость: Вся система разработана с использованием гибких подходов, позволяющих добавлять новые функции и компоненты без существенной модификации кода. Каждый элемент системы может работать независимо друг от друга, что обеспечивает гибкость и масштабируемость системы.

6. Тестирование и отладка: Каждая функция системы может быть легко протестирована без модификации кода программы. Это обеспечивает надежность и стабильность работы системы.

В целом, выполнение проекта позволило успешно применить знания и навыки, полученные в ходе изучения курса, для разработки полноценной системы IoT. Система предоставляет возможность мониторинга здоровья пациентов, анализа данных и принятия медицинских решений на основе этих данных.

8 Приложения

app.py - главный модуль приложения

```
from flask import Flask
from web_app import WebApp

app = Flask(__name__)
web_app = WebApp(app)

if __name__ == '__main__':
    app.run(debug=True)
```

web_app.py - модуль для веб-приложения

```
from flask import Flask, render_template, request, redirect, url_for, jsonify
from pymongo import MongoClient
from datetime import datetime
import plotly.graph_objs as go
from bson import ObjectId


class WebApp:
    def __init__(self, app):
        self.app = app
        self.client = MongoClient('mongodb://localhost:27017/')
        self.db = self.client['medical_system']
        self.patients_collection = self.db['patients']
        self.doctors_collection = self.db['doctors']
        self.heart_rate_collection = self.db['heart_rate_sensor_data']
        self.temperature_collection = self.db['temperature_sensor_data']  # Замена
коллекции на температуру тела

        self.register_routes()

    def register_routes(self):
        self.app.route('/')(self.index)
        self.app.route('/register_doctor', methods=['GET',
'POST'])(self.register_doctor)
        self.app.route('/register_patient', methods=['GET',
'POST'])(self.register_patient)
        self.app.route('/login', methods=['POST'])(self.login)
        self.app.route('/doctor_login')(self.doctor_login)
        self.app.route('/doctor_dashboard')(self.doctor_dashboard)
        self.app.route('/receive_temperature_data/<patient_id>',
methods=['POST'])(self.receive_temperature_data)
        self.app.route('/receive_heart_rate_data/<patient_id>',
methods=['POST'])(self.receive_heart_rate_data)
        self.app.route('/edit_patient/<patient_id>', methods=['GET',
'POST'])(self.edit_patient)
        self.app.route('/patient_monitor/<patient_id>', methods=['GET',
'POST'])(self.patient_monitor)

    def index(self):
        return render_template('index.html')

    def register_doctor(self):
        if request.method == 'POST':
            doctor_id = request.form['doctor_id']
            doctor_password = request.form['doctor_password']
```

```
            self.doctors_collection.insert_one({'_id': doctor_id, 'password':
doctor_password})
            return redirect(url_for('index'))
        return render_template('register_doctor.html')

    def register_patient(self):
        if request.method == 'POST':
            doctor_id = request.form['doctor_id']
            patient_name = request.form['patient_name']
            patient_password = request.form['patient_password']
            date_of_birth = request.form['date_of_birth']
            try:
                datetime.strptime(date_of_birth, '%d.%m.%Y')
            except ValueError:
                return 'Неправильный формат даты рождения. Используйте формат
dd.mm.yyyy'
            patient = {
                'name': patient_name,
                'password': patient_password,
                'date_of_birth': date_of_birth,
                'vital_signs': [],
                'doctor_id': doctor_id
            }
            self.patients_collection.insert_one(patient)
            return render_template('registration_success.html', patient=patient)
        return render_template('register_patient.html')

    def login(self):
        user_type = request.form.get('type')
        user_id = request.form.get('user_id')
        user_password = request.form.get('user_password')

        if user_type == 'doctor':
            doctor = self.doctors_collection.find_one({'_id': user_id, 'password':
user_password})
            if doctor:
                return redirect(url_for('doctor_dashboard', doctor_id=user_id))
        elif user_type == 'patient':
            patient = self.patients_collection.find_one({'_id': user_id, 'password':
user_password})
            if patient:
                return redirect(url_for('patient_dashboard', patient_id=user_id))
        return redirect(url_for('index'))

    def doctor_login(self):
        return render_template('doctor_login.html')

    def doctor_dashboard(self):
        current_doctor_id = request.args.get('doctor_id')
        patients = list(self.patients_collection.find({'doctor_id':
current_doctor_id}))
        return render_template('doctor_dashboard.html', patients=patients,
doctor_id=current_doctor_id)

    def receive_temperature_data(self, patient_id):
        data = request.json
        if patient_id:
            data['timestamp'] = datetime.now().isoformat()
            self.temperature_collection.insert_one({'patient_id': patient_id, 'data':
```

```
data})
            return jsonify({'status': 'success'})
        else:
            return jsonify({'status': 'error', 'message': 'Patient ID not provided'})

    def receive_heart_rate_data(self, patient_id):
        data = request.json
        if patient_id:
            data['timestamp'] = datetime.now().isoformat()
            self.heart_rate_collection.insert_one({'patient_id': patient_id, 'data':
data})
            return jsonify({'status': 'success'})
        else:
            return jsonify({'status': 'error', 'message': 'Patient ID not provided'})

    def edit_patient(self, patient_id):
        current_doctor_id = request.args.get('doctor_id')
        if request.method == 'POST':
            return redirect(url_for('doctor_dashboard', doctor_id=current_doctor_id))
        else:
            patient = self.patients_collection.find_one({'_id': patient_id})
            return render_template('edit_patient.html', patient=patient,
doctor_id=current_doctor_id)

    def patient_monitor(self, patient_id):
        temperature_data = list(self.temperature_collection.find({'patient_id':
patient_id}))
        heart_rate_data = list(self.heart_rate_collection.find({'patient_id':
patient_id}))

        if not temperature_data and not heart_rate_data:
            return 'No data found for this patient'

        temperature_values = []
        heart_rate_values = []
        timestamps_temperature = []
        timestamps_heart_rate = []

        for temp_data in temperature_data:
            temperature_values.append(temp_data['data']['temperature'])
            timestamps_temperature.append(temp_data['data']['timestamp'])

        for hr_data in heart_rate_data:
            heart_rate_values.append(hr_data['data']['heart_rate'])
            timestamps_heart_rate.append(hr_data['data']['timestamp'])

        temperature_fig = go.Figure(
            data=go.Scatter(x=timestamps_temperature, y=temperature_values,
mode='lines', name='Temperature'))
        heart_rate_fig = go.Figure(
            data=go.Scatter(x=timestamps_heart_rate, y=heart_rate_values,
mode='lines', name='Heart Rate'))

        temperature_graph = temperature_fig.to_html(full_html=False)
        heart_rate_graph = heart_rate_fig.to_html(full_html=False)

        avg_temperature = round(sum(temperature_values) / len(temperature_values), 1)
if temperature_values else None
        avg_heart_rate = round(sum(heart_rate_values) / len(heart_rate_values), 1) if
```

```
heart_rate_values else None

        return render_template('patient_monitor.html', patient_id=patient_id,
temperature_graph=temperature_graph,
                               heart_rate_graph=heart_rate_graph,
avg_temperature=avg_temperature,
                               avg_heart_rate=avg_heart_rate)


if __name__ == '__main__':
    app = Flask(__name__)
    web_app = WebApp(app)
    app.run(debug=True)
```

database_manager.py - модуль для работы с базой данных

```
from pymongo import MongoClient
from datetime import datetime

class DatabaseManager:
    def __init__(self):
        self.client = MongoClient('mongodb://localhost:27017/')
        self.db = self.client['medical_system']
        self.patients_collection = self.db['patients']
        self.doctors_collection = self.db['doctors']
        self.heart_rate_collection = self.db['heart_rate_sensor_data']
        self.temperature_collection = self.db['temperature_sensor_data']
    def register_doctor(self, doctor_id, doctor_password):
        self.doctors_collection.insert_one({'_id': doctor_id, 'password':
doctor_password})

    def register_patient(self, doctor_id, patient_name, patient_password,
date_of_birth):
        try:
            datetime.strptime(date_of_birth, '%d.%m.%Y')
        except ValueError:
            return 'Неправильный формат даты рождения. Используйте формат dd.mm.yyyy'
        patient = {
            'name': patient_name,
            'password': patient_password,
            'date_of_birth': date_of_birth,
            'vital_signs': [],
            'doctor_id': doctor_id
        }
        self.patients_collection.insert_one(patient)
        return patient

    def find_doctor(self, doctor_id, doctor_password):
        return self.doctors_collection.find_one({'_id': doctor_id, 'password':
doctor_password})

    def find_patient(self, patient_id, patient_password):
        return self.patients_collection.find_one({'_id': patient_id, 'password':
patient_password})

    def get_patients_for_doctor(self, doctor_id):
        return list(self.patients_collection.find({'doctor_id': doctor_id}))

    def receive_temperature_data(self, patient_id, data):
        data['timestamp'] = datetime.now().isoformat()
```

```
        self.temperature_collection.insert_one({'patient_id': patient_id, 'data':
data})
        return {'status': 'success'}
    def receive_heart_rate_data(self, patient_id, data):
        data['timestamp'] = datetime.now().isoformat()
        self.heart_rate_collection.insert_one({'patient_id': patient_id, 'data':
data})
        return {'status': 'success'}
    def find_patient_by_id(self, patient_id):
        return self.patients_collection.find_one({'_id': patient_id})

    def get_temperature_data(self, patient_id):
        return list(self.temperature_collection.find({'patient_id': patient_id}))

    def get_heart_rate_data(self, patient_id):
        return list(self.heart_rate_collection.find({'patient_id': patient_id}))
```

heart_rate_emulator.py – эмулятор умного пульсометра

```
import requests
import random
from datetime import datetime
import time

# URL основного Flask-приложения
MAIN_APP_URL = 'http://localhost:5000'
# ID пациента для отправки данных
PATIENT_ID = '661ffbd88f30fcefe5d8fbba'

def send_data():
    while True:
        # Генерация случайных данных для сердечного ритма
        heart_rate = random.randint(60, 100)
        timestamp = datetime.now().isoformat()

        # Формирование JSON-запроса
        data = {
            'patient_id': PATIENT_ID,
            'heart_rate': heart_rate,
            'timestamp': timestamp
        }

        # Отправка POST-запроса на основное приложение
        response =
requests.post(f"{MAIN_APP_URL}/receive_heart_rate_data/{PATIENT_ID}", json=data)
        print(response.text)

        time.sleep(5)

if __name__ == '__main__':
    send_data()
```

heart_rate_emulator.py – эмулятор умного градусника

```
import requests
import random
from datetime import datetime
import time

# URL основного Flask-приложения
MAIN_APP_URL = 'http://localhost:5000'
```

```
# ID пациента для отправки данных о температуре
PATIENT_ID = '661ffbd88f30fcefe5d8fbba'

def send_data():
    while True:
        # Генерация случайной температуры
        temperature = round(random.uniform(36.0, 37.5), 1)
        timestamp = datetime.now().isoformat()

        # Формирование JSON-запроса
        data = {
            'patient_id': PATIENT_ID,
            'temperature': temperature,
            'timestamp': timestamp
        }

        # Отправка POST-запроса на основное приложение
        response =
requests.post(f"{MAIN_APP_URL}/receive_temperature_data/{PATIENT_ID}", json=data)
        print(response.text)

        time.sleep(5)


if __name__ == '__main__':
    send_data()
```

doctor_dashboard.html – дэшборд врача

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Дэшборд врача</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Дэшборд врача</h1>
        <table>
            <thead>
                <tr>
                    <th>Имя пациента</th>
                    <th>Дата рождения</th>
                    <th>Действия</th>
                </tr>
            </thead>
            <tbody>
                {% for patient in patients %}
                <tr>
                    <td>{{ patient['name'] }}</td>
                    <td>{{ patient['date_of_birth'] }}</td>
                    <td>
                        <a href="/edit_patient/{{ patient['_id'] }}?doctor_id={{
doctor_id }}">Редактировать</a>
                        <a href="{{ url_for('patient_monitor',
patient_id=patient._id) }}">Монитор состояния</a>
                    </td>
                </tr>
```

```
                {% endfor %}
            </tbody>
        </table>
        <div class="add-patient-link">
            <p><a href="/register_patient">Добавить пациента</a></p>
        </div>
    </div>
</body>
</html>
```

doctor_login.html – Страница авторизации врача

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Вход для врача</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Вход для врача</h1>
        <form action="/login" method="post">
            <input type="hidden" name="type" value="doctor">
            <input type="text" name="user_id" placeholder="ID врача" required>
            <input type="password" name="user_password" placeholder="Пароль"
required>
            <input type="submit" value="Войти" class="button">
        </form>
        <p>Нет аккаунта? <a href="/register_doctor"
class="link">Зарегистрироваться</a></p>
    </div>
</body>
</html>
```

edit_patient.html – Страница редактирования данных пациента

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Редактирование данных пациента</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Редактирование данных пациента</h1>
        <form action="/edit_patient/{{ patient_id }}" method="post">
            <input type="hidden" name="type" value="patient">
            <input type="text" id="date_of_birth" name="date_of_birth"
placeholder="Дата рождения (в формате dd.mm.yyyy)" value="{% if patient %}{{
patient.get('date_of_birth', '') }}{% endif %}" required>
            <input type="submit" value="Сохранить" class="button">
        </form>
        <p><a href="{{ url_for('doctor_dashboard', doctor_id=doctor_id) }}"
class="button">Вернуться в дэшборд</a></p>
    </div>
</body>
</html>
```

index.html – Начальная страница

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Медицинская Система</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Медицинская Система</h1>
        <div class="buttons">
            <a href="/doctor_login" class="button">Я врач</a>
            <a href="/patient_login" class="button">Я пациент</a>
        </div>
    </div>
</body>
</html>
```

patient_monitor.html – Монитор состояния пациента

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Мониторинг состояния пациента</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Мониторинг состояния пациента</h1>
        <h2>Patient ID: {{ patient_id }}</h2>

        <div class="vitals">
            <div class="temperature">
                <h3>Температура тела</h3>
                {{ temperature_graph | safe }}
                {% if avg_temperature is not none %}
                    <p>Средняя температура тела: {{ avg_temperature }}</p>
                {% endif %}
            </div>

            <div class="heart-rate">
                <h3>Частота сердечных сокращений</h3>
                {{ heart_rate_graph | safe }}
                {% if avg_heart_rate is not none %}
                    <p>Средняя частота сердечных сокращений: {{ avg_heart_rate }}</p>
                {% endif %}
            </div>
        </div>
    </div>
</body>
</html>
```

register_doctor.html – Страница регистрации врача

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Регистрация врача</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Регистрация врача</h1>
        <form action="/register_doctor" method="post">
            <label for="doctor_id">ID врача:</label><br>
            <input type="text" id="doctor_id" name="doctor_id" required>
            <label for="doctor_password">Пароль:</label>
            <input type="password" id="doctor_password" name="doctor_password"
required>
            <input type="submit" value="Зарегистрироваться">
        </form>
        <p>Уже есть аккаунт? <a href="/doctor_login">Войти</a></p>
    </div>
</body>
</html>
```

register_patient.html – Страница регистрации пациента

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Регистрация пациента</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Регистрация пациента</h1>
        <form action="/register_patient" method="post">
            <input type="text" id="doctor_id" name="doctor_id" placeholder="ID врача"
required>
            <input type="text" id="patient_name" name="patient_name" placeholder="Имя
пациента" required>
            <input type="password" id="patient_password" name="patient_password"
placeholder="Пароль" required>
            <input type="text" id="date_of_birth" name="date_of_birth"
placeholder="Дата рождения (в формате dd.mm.yyyy):" required><br>
            <input type="submit" value="Зарегистрировать">
        </form>
    </div>
</body>
</html>
```

registration_success.html – Страница-подтверждение успешной регистрации пациента

```
<!DOCTYPE html>
<html lang="ru">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Регистрация успешна</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Регистрация успешна!</h1>
        <p>Логин: {{ patient['_id'] }}</p>
        <p>Пароль: {{ patient['password'] }}</p>
        <a href="{{ url_for('doctor_dashboard', doctor_id=patient.doctor_id) }}"
class="button">Вернуться в дэшборд</a>
    </div>
</body>
</html>
```

styles.css – Стили

```
/* styles.css */

body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    color: #333;
    margin: 0;
}

.container {
    width: 90%;
    max-width: 1200px;
    margin: 20px auto;
    background-color: #f9f9f9;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1 {
    color: #008000;
    margin-bottom: 20px;
}

.vitals {
    display: flex;
    justify-content: space-between;
}

.temperature, .heart-rate {
    width: 48%;
}

.temperature h3, .heart-rate h3 {
    margin-top: 0;
}

form {
    margin-top: 20px;
    display: flex;
    flex-direction: column;
    align-items: center;
```

```
}

input[type="text"],
input[type="password"],
input[type="submit"] {
    width: calc(100% - 20px);
    padding: 10px;
    border-radius: 5px;
    border: 1px solid #ccc;
    margin-bottom: 10px;
}

input[type="submit"] {
    width: 100%;
    background-color: #008000;
    color: white;
    cursor: pointer;
    transition: background-color 0.3s, box-shadow 0.3s;
}

input[type="submit"]:hover {
    background-color: #006400;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

a {
    color: #008000;
    text-decoration: none;
}

a:hover {
    text-decoration: underline;
}


table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
}

th, td {
    padding: 12px;
    text-align: left;
    border-bottom: 1px solid #ddd;
}

th {
    background-color: #f2f2f2;
    font-weight: bold;
    color: #333;
}

tr:hover {
    background-color: #f5f5f5;
}
```

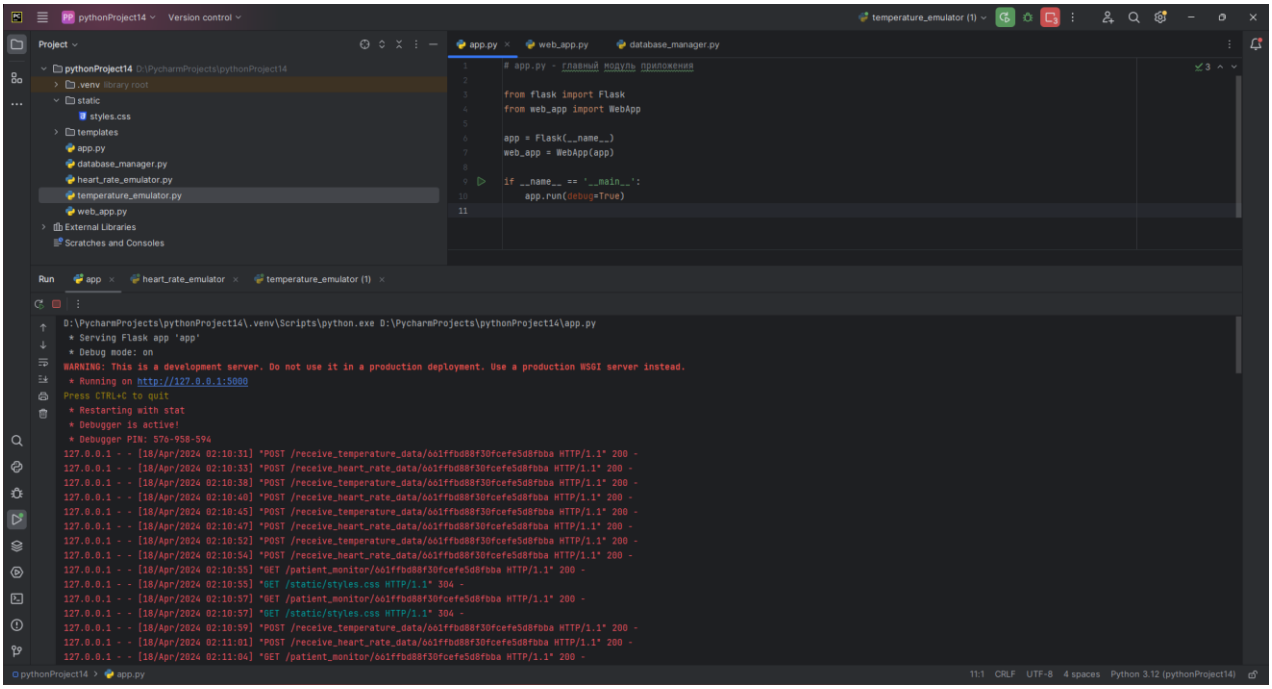

Рисунок 11 – Лог основной программы

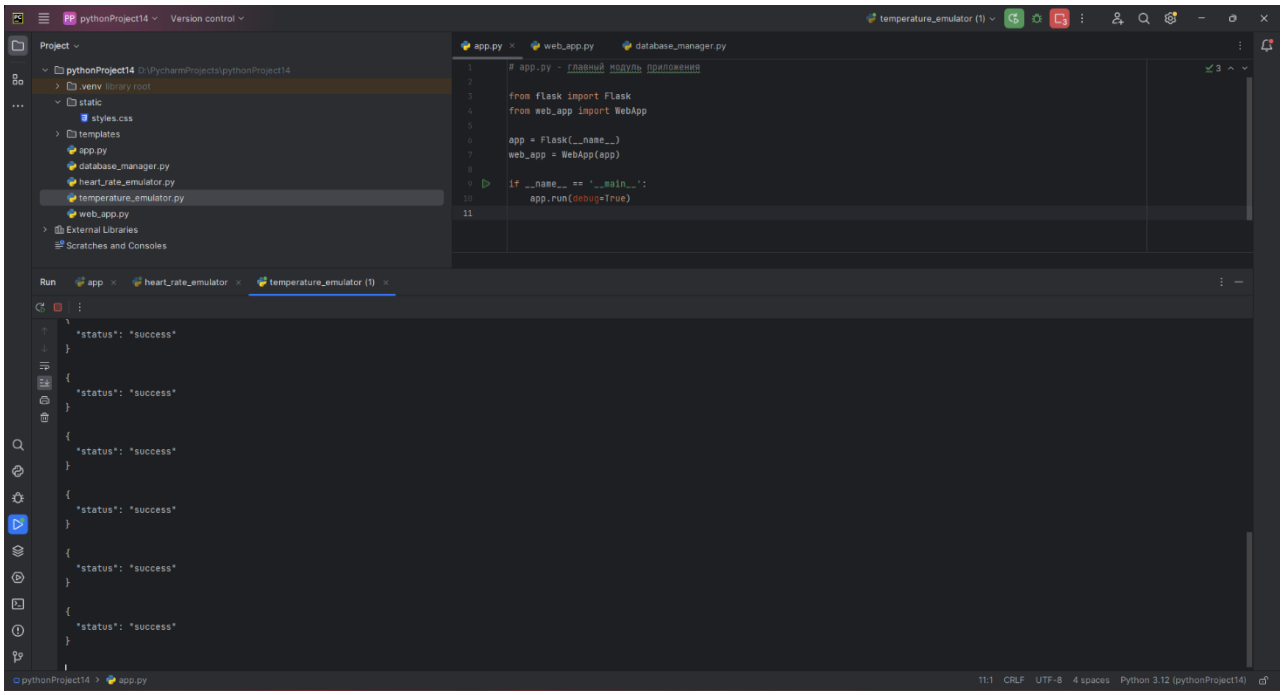

Рисунок 11 – Лог эмулятора умного градусника

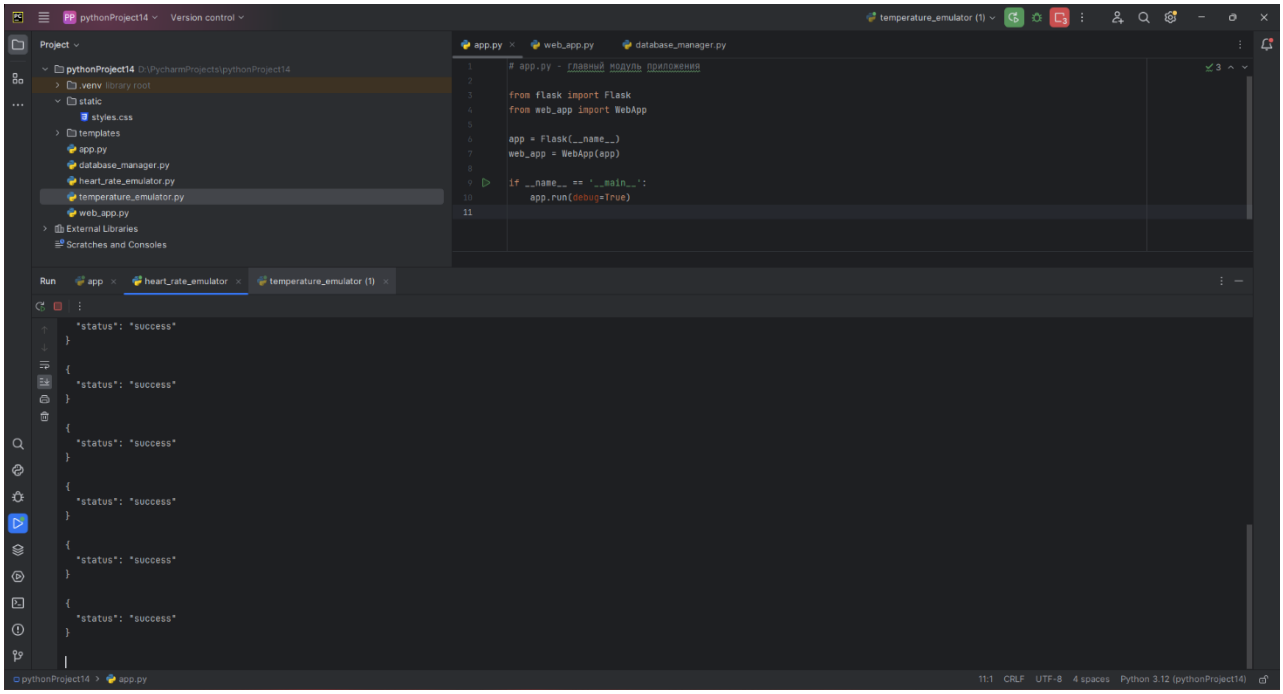

Рисунок 11 – Лог эмулятора умного пульсометра

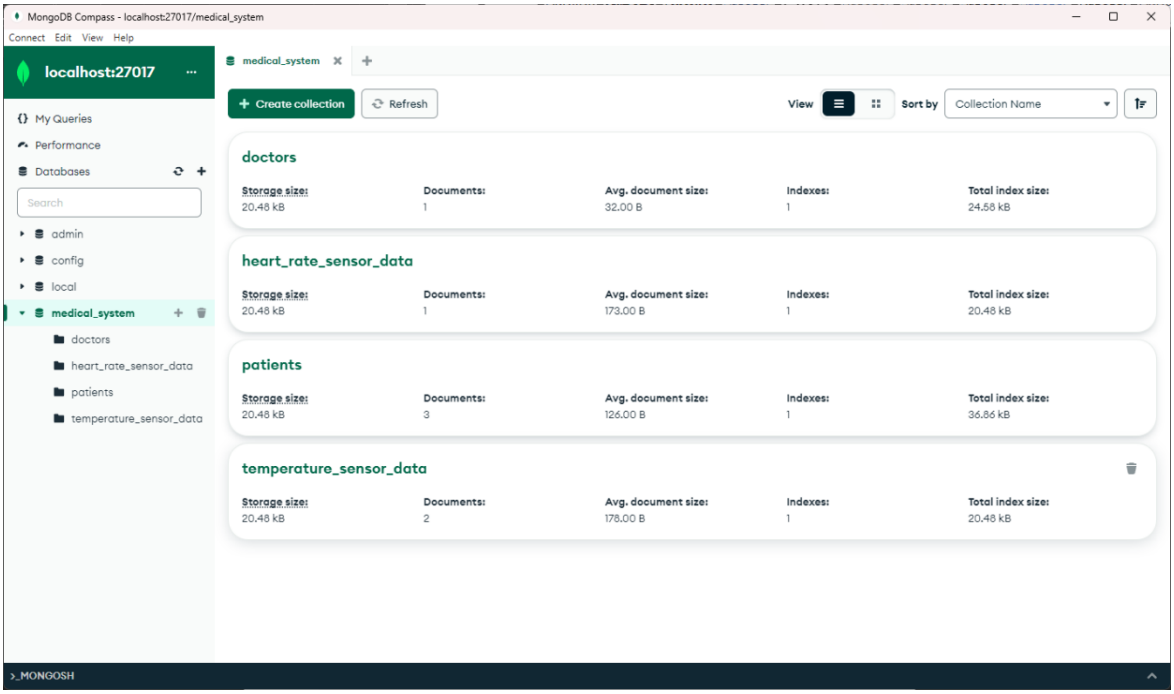

Рисунок 11 – Лог MongoDB Compass

Исходники проекта доступны по [ссылке]