МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики
Кафедра Математического обеспечения и суперкомпьютерных технологий
Направление подготовки “Инженерия программного обеспечения”

ОТЧЁТ
по учебной практике на тему
“Разработка программно-аппаратного комплекса для мониторинга показателей сердца человека”

Выполнил:
студент группы 382008-1
Булгаков Д.Э.
Подпись

Проверил:
к.т.н., доц.
Борисов Н.А.
Подпись

Нижний Новгород
2024

Содержание

1. Введение 3

2. Постановка задачи. 4

3. Проведенная работа. 5

3.1 Стек технологий. 5

3.1.1 Язык программирования и фреймворк. 5

3.1.2 База данных. 6

3.1.3 Выбор инструментов для развертывания приложения. 6

3.2 Сценарии использования. 8

3.2.1 Общие сценарии использования. 8

3.2.2 Сценарии использования для Electron приложения. 8

3.2.3 Сценарии использования для Web-приложения. 9

3.3 Разработка и реализация архитектуры. 10

3.3.1 Использование Docker для контейнеризации приложения. 10

3.3.2 Разработка и реализация логики на Python с использованием Flask. . . . 11

3.3.3 Работа с базой данных. 12

4. Заключение. 14

5. Список литературы. 15

6. Приложение. 16

1. Введение

Заболевания сердца и сосудов, ставшие ведущей причиной смертности по всему миру, находятся в центре внимания медицинского сообщества. Ишемическая болезнь сердца, артериальная гипертензия и другие патологии сердечно-сосудистой системы требуют серьезного подхода к диагностике и профилактике.

Сердечно-сосудистая система, сложная в своей структуре и функциональности, подвергается различным воздействиям, которые могут привести к серьезным нарушениям. В этом контексте регулярные обследования приобретают ключевое значение. Они не только предоставляют возможность выявления начальных стадий заболеваний до появления явных симптомов, но и открывают перспективы для раннего вмешательства и эффективной профилактики. Такой подход становится неотъемлемой частью стратегии поддержания здоровья сердечно-сосудистой системы в условиях современного образа жизни.

2. Постановка задачи.

1. **Выбрать стек технологий, на основе которых будет написан сервер.**

    - Определить язык программирования для сервера.
    - Определить какую библиотеку использовать для написания сервера.
    - Выбрать подходящую базу данных.

2. **Определить сценарии использования сервера.**

    Сценарии использования разделены на следующие платформы:

    - Web сайт на основе Vue.js
    - Electron приложение.

    Общие сценарии можно обобщить и выделить в отдельную компоненту.

3. **Разработать и реализовать архитектуру сервера.**

3. Проведенная работа.

3.1. Стек технологий.

3.1.1. Язык программирования и фреймворк.

Перед началом проектирования архитектуры сервера, требуется выбрать язык программирования и фреймворк, который обеспечивал бы удобные инструменты для обработки HTTP-запросов, маршрутизации и взаимодействия с базой данных. На рассмотрении были следующие варианты:

- Node.js с фреймворком Express.js
- Golang с фреймворками Gin или Martini
- Python с фреймворками Flask или Django

Необходимо сравнить основные характеристики каждого решения, такие как производительность, удобство и простота разработки, доступность библиотек и инструментов, а также сообщество и поддержка. Прежде всего, было решено остановиться на языке программирования Python по следующим причинам:

- **Простота и читаемость кода:**
  Python известен своим чистым и выразительным синтаксисом, который делает код легким для чтения, понимания и поддержки.
- **Широкие возможности для веб-разработки:**
  Python имеет обширную экосистему библиотек и фреймворков для веб-разработки. Flask, Django, FastAPI - все они предоставляют мощные инструменты для создания веб-приложений любого уровня сложности.
- **Кросс-платформенность и портативность:**
  Python работает на множестве операционных систем, что обеспечивает гибкость и портативность разработки. Это означает, что код, написанный на Python, может быть легко перенесен с одной платформы на другую без необходимости внесения значительных изменений.

- **Богатая стандартная библиотека:**
  Python поставляется с обширной стандартной библиотекой, которая включает в себя множество модулей для работы с сетью, обработки данных, взаимодействия с базами данных и многое другое.

Осталось определиться с фреймворком Python. Было решено использовать Flask, т.к. сервер является учебным, небольшим проектом, он не требует таких глобальных и серьезных инструментов, которые предоставляет Django. Также Flask обладает более интуитивно понятный интерфейсом, что позволит меньше времени потратить на изучение инструмента.

3.1.2. База данных.

Для структурирования и хранения данных необходимо выбрать базу данных, которая лучше всего подойдет к выбранному фреймворку. На рассмотрении были следующие варианты:

- PostgreSQL
- SQLite
- MySQL
- MongoDB
- Redis

По итогу, было решено использовать PostgreSQL из-за его надежности, расширяемости, а также доступности. Кроме того, PostgreSQL активно развивается сообществом и имеет обширную документацию, что делает его привлекательным выбором для учебных проектов.

3.1.3. Выбор инструментов для развертывания приложения.

Для обеспечения более удобного и гибкого процесса разработки и развертывания, было принято решение использовать Docker.

Docker обеспечивает создание и управление контейнеризированными средами разработки и развертывания, что значительно упрощает процесс настройки окружения и обеспечивает

переносимость приложения между различными средами. Кроме того, Docker обеспечивает изолированное выполнение приложений, что позволяет избежать конфликтов между зависимостями и обеспечивает надежность работы приложения.

3.2. Сценарии использования.

Серверная часть приложения должна обрабатывать запросы, поступающие от двух типов клиентских приложений:

- Десктопное приложение на основе фреймворка Electron.
- Web-приложение на основе Vue.js

Некоторые сценарии использования применимы к обоим типам приложений, поэтому они выделены в отдельный модуль, который содержит общие сценарии использования.

3.2.1. Общие сценарии использования.

- **Идентификация, аутентификация и авторизация пользователя:**
  На сервере будет хранится и обрабатываться информация всех пользователей, поэтому пользователи должны иметь возможность идентифицировать себя на сервере, проходить аутентификацию и, при необходимости, проходить авторизацию для доступа к определенным ресурсам или функциональности.
- **Маршруты для работы с сессионными ключами:**
  Для работы с цифровыми личностями используются сессионные ключи. Поэтому необходимо поддерживать маршруты для создания новых сессионных ключей, обновление их срока действия и удаление ключей после завершения сеанса работы пользователя.

3.2.2. Сценарии использования для Electron приложения.

Electron-приложение служит посредником для передачи данных ЭКГ с устройства ESP32 на сервер. Кроме общих сценариев использования в API приложения должны быть реализованы следующие пункты:

- **Разработка маршрутов для приема данных ЭКГ от приложения Electron:**
  Предполагает создание маршрутов на сервере, которые будут принимать данные ЭКГ,

отправленные из приложения Electron, и обрабатывать их для последующего сохранения или анализа.

- **Создание механизмов хранения полученных данных ЭКГ на сервере:**
  Требует разработки функционала для сохранения полученных данных ЭКГ на сервере. Это может включать в себя разработку механизмов для хранения и организации данных.

- **Разработка механизмов передачи данных ЭКГ на анализ искусственным интеллектом:**
  Включает в себя создание функционала для передачи данных ЭКГ на анализ искусственным интеллектом. Это может включать в себя разработку API или механизмов интеграции с собственными моделями искусственного интеллекта для обработки данных и генерации результатов анализа.

3.2.3. Сценарии использования для Web-приложения.

Web-приложение является основным источником для изучения и рассмотрения пользователем своих данных ЭКГ. Кроме общих сценариев использования в API приложения должны быть реализованы следующие пункты:

- **Обновление данных пользователя:**
  Пользователь изменяет свои данные, такие как дата рождения, и отправляет их на сервер для обновления.

- **Отображение графиков ЭКГ:**
  Пользователь запрашивает просмотр графиков данных ЭКГ, полученных от Electron приложения.

- **Вывод заключения о состоянии здоровья:**
  Пользователь запрашивает анализ состояния здоровья на основе данных ЭКГ, анализируемых искусственным интеллектом.

3.3. Разработка и реализация архитектуры.

3.3.1. Использование Docker для контейнеризации приложения.

Для упрощения процесса разработки, тестирования и развертывания серверного приложения был использован Docker. Вот основные шаги при разработке приложения с использованием Docker:

1. **Определение сервисов в файле docker-compose.yml:**
    - В файле `docker-compose.yml` определены два сервиса: `web` для серверного приложения на Flask и `postgres` для базы данных PostgreSQL.
    - Для каждого сервиса указаны настройки, такие как сборка образа, зависимости, порты, переменные окружения и т.д.
2. **Настройка среды разработки:**
    - Для сервиса `web` указаны переменные окружения `FLASK_APP` И `FLASK_ENV`, определяющие основные параметры запуска Flask приложения.
    - В качестве базового образа для серверного приложения использован официальный образ Python с поддержкой Ubuntu 22.04.
3. **Управление зависимостями и файлами приложения:**
    - В Dockerfile определены шаги для установки зависимостей из файла `requirements.txt` и копирования всех остальных файлов приложения внутрь контейнера.
    - Это позволяет изолировать приложение и его зависимости внутри контейнера, обеспечивая надежную и воспроизводимую среду выполнения.
4. **Использование внешнего образа PostgreSQL:**
    - Для сервиса `postgres` использован официальный образ PostgreSQL.
    - В volumes прописан путь к файлу `init.sql`, который будет запущен при инициализации контейнера и содержит SQL-запросы для создания базы данных и таблиц.
5. **Управление данными:**

- Для сохранения данных базы данных PostgreSQL использован Docker volume, который привязывается к директории внутри контейнера.
- Это обеспечивает сохранность данных даже при перезапуске контейнера или удалении образа.

6. **Метаданные образа и авторство:**
    - В Dockerfile добавлены метаданные, такие как метка `maintainer` с указанием контактной информации разработчика и метка `description` с кратким описанием приложения.

3.3.2. Разработка и реализация логики на Python с использованием Flask.

Для создания серверной части приложения был выбран фреймворк Flask, который написан для Python. Вот основные шаги при разработке приложения с использованием Flask:

1. **Инициализация приложения и настройка JWT:**
    - В файле app.py создается объект Flask приложения и настраивается JWT (JSON Web Tokens) для обеспечения безопасной аутентификации пользователей.
    - Для генерации токенов используется секретный ключ и указывается срок действия токена.
2. **Обработка исключений JWT:**
    - В приложении реализована обработка исключений, связанных с JWT, таких как отсутствие заголовка авторизации, неверный заголовок авторизации и истекший срок действия токена.
3. **Регистрация маршрутов:**
    - Сделать тут по красоте
    - Для удобства организации кода маршруты разделены на несколько Blueprint'ов, включая common и web.
    - Blueprint common содержит routings для аутентификации и регистрации пользователей.

- Для каждого маршрута определены соответствующие функции-обработчики, которые выполняют необходимые действия, такие как вход в систему, обновление токена и регистрация пользователя.

4. **Разработка функционала доступа к данным:**

   - Для удобства доступа к данным и выполнения операций CRUD (создание, чтение, обновление, удаление) были реализованы модели данных и базовый класс, предоставляющий общие методы для работы с базой данных.
   - Это позволяет упростить разработку и обеспечить единый подход к взаимодействию с данными из различных частей приложения.

5. **Использование библиотек:**

   - Для работы с базой данных в приложении был выбран фреймворк SQLAlchemy, обеспечивающий ORM (объектно-реляционное отображение) и удобные инструменты для работы с реляционными базами данных.
   - Это позволяет использовать объектно-ориентированный подход при работе с данными и упрощает процесс взаимодействия с базой данных.

3.3.3. Работа с базой данных.

Для обеспечения хранения и управления данными серверного приложения была выбрана реляционная база данных. Разработка и реализация базы данных включает в себя следующие шаги:

1. **Создание схемы базы данных:**

   - В ходе проектирования было определено три основных сущности: пользователи данные ЭКГ и аутентификационные данные.
   - Для каждой сущности была разработана соответствующая таблица в базе данных, представляющая собой логическую структуру для хранения данных.

2. **Определение полей и их типов:**

   - Каждая таблица содержит набор полей, определяющих характеристики сущности.

- Типы данных для полей выбирались с учетом требований к хранению и обработке информации.

3. **Определение отношений между таблицами:**

   - Для связывания данных между таблицами были определены внешние ключи и отношения.
   - Например, таблица аутентификационных данных имеет внешний ключ, связывающий ее с таблицей пользователей.

4. Заключение.

Таким образом, в результате выполнения практики были достигнуты значительные результаты в разработке серверной части приложения на основе Python Flask. Процесс разработки позволил мне ознакомиться с основными инструментами и технологиями, необходимыми для создания современных веб-приложений.

Основные достижения включают в себя:

1. Разработка и настройка маршрутов для обработки запросов от клиентских приложений на базе Electron и веб-приложений на Vue.js.
2. Реализация аутентификации и регистрации пользователей с использованием JWT для обеспечения безопасности взаимодействия с сервером.
3. Внедрение базы данных PostgreSQL с использованием SQLAlchemy для хранения данных пользователей и их аутентификационных данных.
4. Освоение Docker для удобного развертывания и управления серверным приложением в изолированных контейнерах.

5. Список литературы.

1. Гринберг, М. Flask Web Development: Developing Web Applications with Python [Текст] / М. Гринберг. - СПб.: Наука, 2018. - 300 с.
2. Гонсалес, М. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database [Текст] / М. Гонсалес. - М.: Вильямс, 2019. - 400 с.
3. Cochrane, K. Docker Cookbook: Over 100 practical and insightful recipes to build distributed applications with Docker, 2nd Edition [Текст] / K. Cochrane. - М.: Питер, 2019. - 350 с.
4. Pallets Projects. Flask Documentation [Электронный ресурс]. - Режим доступа: https://flask.palletsprojects.com/ (дата обращения: 05.05.2024).
5. SQLAlchemy Documentation [Электронный ресурс]. - Режим доступа: https://docs.sqlalchemy.org/en/20/ (дата обращения: 05.05.2024).

6. Приложение.