РУКОВОДСТВО СИСТЕМНОГО ПРОГРАММИСТА

для системы учета заявок на ремонт климатического оборудования

1. АРХИТЕКТУРА СИСТЕМЫ

Система реализована в виде клиент-серверного приложения, состоящего из трех основных компонентов: базы данных SQLite, бэкенд-сервера на FastAPI и фронтенд-приложения на Streamlit. Бэкенд предоставляет REST API для взаимодействия с базой данных и реализует бизнес-логику приложения. Фронтенд предоставляет веб-интерфейс для пользователей различных ролей. Система использует JWT-токены для аутентификации и авторизации пользователей.

2. ТРЕБОВАНИЯ К СРЕДЕ ВЫПОЛНЕНИЯ

2.1. Аппаратные требования:

- Процессор: 1 ГГц или выше
- Оперативная память: 2 ГБ минимум, 4 ГБ рекомендуется
- Свободное дисковое пространство: 500 МБ
- Сетевая карта для работы в сети

2.2. Программные требования:

- Операционная система: Windows 10/11, Linux (Ubuntu 18.04 и выше), macOS 10.14 и выше
- Python версии 3.8 или выше
- pip (пакетный менеджер Python)
- Доступ в интернет для установки зависимостей

3. УСТАНОВКА И НАСТРОЙКА

3.1. Подготовка среды:

1. Установите Python 3.8 или выше с официального сайта python.org
2. Убедитесь, что pip установлен и обновлен: python -m pip install --upgrade pip
3. Создайте рабочую директорию для проекта, например: C:\RepairSystem\

3.2. Установка зависимостей:

1. Поместите все файлы проекта в рабочую директорию
2. Установите зависимости для бэкенда: pip install -r requirements.txt
3. Установите зависимости для фронтенда: pip install -r frontend_requirements.txt

3.3. Настройка базы данных:

1. Файл базы данных repair_requests.db должен находиться в корневой директории проекта
2. База данных уже содержит необходимые таблицы и тестовые данные
3. При необходимости можно создать новую базу данных, выполнив SQL-скрипт создания таблиц

3.4. Конфигурация приложения:

1. Секретный ключ JWT настраивается в файле main.py в переменной SECRET_KEY
2. Для production-среды необходимо изменить значение SECRET_KEY на более сложное
3. Время жизни токена настраивается в переменной ACCESS_TOKEN_EXPIRE_MINUTES
4. URL API настраивается в файле streamlit_app.py в переменной API_BASE_URL
5. ЗАПУСК СИСТЕМЫ

4.1. Запуск в режиме разработки:

1. Откройте два терминала
2. В первом терминале запустите бэкенд: python run.py или uvicorn main:app --reload --host 0.0.0.0 --port 8000
3. Во втором терминале запустите фронтенд: streamlit run streamlit_app.py
4. Бэкенд будет доступен по адресу: http://localhost:8000
5. Фронтенд будет доступен по адресу: http://localhost:8501
6. Документация API доступна по адресу: http://localhost:8000/docs

4.2. Запуск в production-режиме:

1. Для бэкенда рекомендуется использовать процесс-менеджер (systemd, Supervisor)
2. Для фронтенда Streamlit можно настроить как службу
3. Настройте обратный прокси (Nginx, Apache) для доступа к обоим сервисам
4. СТРУКТУРА БАЗЫ ДАННЫХ

5.1. Таблица users:

- user_id: INTEGER PRIMARY KEY - уникальный идентификатор пользователя
- fio: TEXT - ФИО пользователя
- phone: TEXT - номер телефона

- login: TEXT - логин для входа
- password: TEXT - пароль (в открытом виде, для production рекомендуется хеширование)
- role: TEXT - роль пользователя (Менеджер, Оператор, Специалист, Заказчик)

5.2. Таблица requests:

- request_id: INTEGER PRIMARY KEY - уникальный идентификатор заявки
- start_date: TEXT - дата создания заявки
- climate_tech_type: TEXT - тип климатического оборудования
- climate_tech_model: TEXT - модель оборудования
- problem_description: TEXT - описание проблемы
- request_status: TEXT - статус заявки
- completion_date: TEXT - дата завершения работ
- repair_parts: TEXT - использованные запчасти
- master_id: INTEGER - идентификатор специалиста (внешний ключ к users)
- client_id: INTEGER - идентификатор заказчика (внешний ключ к users)

5.3. Таблица comments:

- comment_id: INTEGER PRIMARY KEY - уникальный идентификатор комментария
- message: TEXT - текст комментария
- master_id: INTEGER - идентификатор специалиста (внешний ключ к users)
- request_id: INTEGER - идентификатор заявки (внешний ключ к requests)
- created_at: TEXT - дата и время создания комментария

6. АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ

6.1. Механизм аутентификации:
Система использует JWT (JSON Web Tokens) для аутентификации. При успешном входе пользователь получает токен, который должен передавать в заголовке Authorization: Bearer <token> при каждом запросе.

6.2. Ролевая модель:

- Менеджер: полный доступ ко всем функциям системы
- Оператор: создание и редактирование заявок, просмотр комментариев
- Специалист: работа с назначенными заявками, добавление комментариев, просмотр своей статистики

- Заказчик: просмотр своих заявок, создание новых заявок

6.3. Регистрация новых пользователей:
Новые пользователи могут зарегистрироваться самостоятельно через интерфейс фронтенда. При регистрации указывается ФИО, телефон, логин, пароль и роль. Система проверяет уникальность логина.

7. API ИНТЕРФЕЙС

7.1. Основные эндпоинты:

- POST /token - получение JWT токена
- POST /register - регистрация нового пользователя
- GET /requests - получение списка заявок
- POST /requests - создание новой заявки
- GET /requests/{id} - получение информации о заявке
- PUT /requests/{id} - обновление заявки
- DELETE /requests/{id} - удаление заявки
- GET /requests/{id}/comments - получение комментариев к заявке
- POST /requests/{id}/comments - добавление комментария
- GET /stats/* - получение статистики

7.2. Форматы данных:
Все запросы и ответы используют формат JSON. Даты передаются в формате ISO 8601 (YYYY-MM-DD).

8. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ

8.1. Управление заявками:

- Создание заявок с указанием типа оборудования, модели, описания проблемы
- Назначение специалистов на заявки
- Изменение статусов заявок
- Добавление информации о запчастях
- Отметка о завершении работ

8.2. Работа с комментариями:

- Добавление комментариев к заявкам
- Просмотр истории комментариев
- Привязка комментариев к конкретным специалистам

8.3. Статистика и отчетность:

- Количество выполненных заявок
- Среднее время выполнения заявок
- Статистика по типам неисправностей
- Статистика по специалистам (для менеджеров)

8.4. Оценка качества работы:

- Генерация QR-кода для перехода к форме оценки
- Ссылка на Google Forms для сбора отзывов

9. МОДИФИКАЦИЯ СИСТЕМЫ

9.1. Добавление новых ролей:
Для добавления новой роли необходимо:

1. Добавить роль в перечень допустимых значений в файле schemas.py
2. Обновить проверки прав доступа в файле main.py
3. Добавить обработку новой роли в интерфейсе streamlit_app.py

9.2. Изменение бизнес-логики:
Изменения в бизнес-логике вносятся в файлы:

- models.py - работа с базой данных
- main.py - обработка API запросов
- schemas.py - структуры данных

9.3. Расширение функционала:
Для добавления нового функционала необходимо:

1. Создать новые таблицы в базе данных при необходимости
2. Добавить соответствующие методы в models.py
3. Создать схемы данных в schemas.py
4. Добавить эндпоинты в main.py
5. Реализовать интерфейс в streamlit_app.py

6. РЕЗЕРВНОЕ КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЕ

10.1. Резервное копирование базы данных:
Рекомендуется регулярно создавать резервные копии файла repair_requests.db. Для этого можно использовать стандартные средства операционной системы или создать скрипт автоматического копирования.

10.2. Восстановление из резервной копии:
Для восстановления базы данных необходимо остановить приложение, заменить файл repair_requests.db резервной копией и перезапустить приложение.

11. МОНИТОРИНГ И ДИАГНОСТИКА

11.1. Логирование:
FastAPI предоставляет встроенное логирование. Для настройки детального логирования можно использовать стандартный модуль logging Python.

11.2. Мониторинг производительности:
Для мониторинга производительности API можно использовать инструменты типа Prometheus с экспортером для FastAPI.

11.3. Диагностика проблем:
При возникновении проблем рекомендуется:

1. Проверить логи приложения
2. Убедиться в доступности базы данных
3. Проверить корректность JWT токенов
4. Проверить права доступа к файлам
5. БЕЗОПАСНОСТЬ

12.1. Рекомендации по безопасности:

1. Замените секретный ключ JWT в production-среде
2. Реализуйте хеширование паролей вместо хранения в открытом виде
3. Настройте HTTPS для защиты передаваемых данных
4. Регулярно обновляйте зависимости
5. Ограничьте доступ к серверу по SSH

12.2. Защита от атак:

- Реализована защита от SQL-инъекций через параметризованные запросы
- Используется JWT для защиты от CSRF
- Реализована проверка прав доступа на уровне API

13. ОБСЛУЖИВАНИЕ И ПОДДЕРЖКА

13.1. Регулярное обслуживание:

1. Мониторинг свободного места на диске
2. Проверка логов на ошибки
3. Обновление зависимостей
4. Резервное копирование данных

13.2. Устранение неисправностей:
Типичные проблемы и их решения:

- Ошибка подключения к базе данных: проверить наличие файла repair_requests.db и права доступа
- Ошибки аутентификации: проверить корректность JWT токена
- Медленная работа: проверить нагрузку на сервер, оптимизировать запросы к базе данных

14. МАСШТАБИРУЕМОСТЬ

14.1. Вертикальное масштабирование:
При увеличении нагрузки можно:

1. Увеличить ресурсы сервера (CPU, RAM)
2. Оптимизировать запросы к базе данных
3. Внедрить кэширование

14.2. Горизонтальное масштабирование:
Для горизонтального масштабирования необходимо:

1. Вынести базу данных на отдельный сервер
2. Настроить несколько экземпляров бэкенда
3. Использовать балансировщик нагрузки
4. ИНТЕГРАЦИЯ С ДРУГИМИ СИСТЕМАМИ

15.1. Возможности интеграции:
Система предоставляет REST API, что позволяет интегрировать ее с другими системами, такими как:

- CRM системы
- Системы бухгалтерского учета
- Мессенджеры для уведомлений
- Email-рассылки

15.2. Форматы обмена данными:
Для интеграции используются стандартные форматы:

- JSON для обмена данными
- REST для взаимодействия
- JWT для аутентификации

16. ТЕСТИРОВАНИЕ И ОТЛАДКА

16.1. Модульное тестирование:
Для тестирования отдельных компонентов можно использовать стандартные инструменты Python (unittest, pytest).

16.2. Интеграционное тестирование:
Для тестирования API можно использовать Postman или аналогичные инструменты.

16.3. Тестирование производительности:
Для тестирования производительности можно использовать инструменты типа locust или Apache JMeter.

Руководство составлено в соответствии с требованиями технического задания и стандартами разработки программного обеспечения. Система готова к эксплуатации и дальнейшему развитию в соответствии с потребностями заказчика.