МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт компьютерных наук
Кафедра автоматизированных систем управления

ОТЧЕТ

о производственной практике

"Проектно-технологическая практика"

в ООО "ОНСОФТ"

Студент АС-21-1 Станиславчук С.М.

Руководитель от кафедры Болдырихин О.В.

Руководитель ВКР
канд. техн. наук Гаев Л.В.

Руководитель от предприятия Кисова А.Е.

Липецк 2025 г.

Липецкий государственный технический университет

Институт компьютерных наук

Кафедра автоматизированных систем управления

ЗАДАНИЕ НА ПРОИЗВОДСТВЕННУЮ ПРАКТИКУ

Студенту Станиславчуку Сергею Михайловичу группы АС-21-1

Направление (специальность) 09.03.01 "Информатика и вычислительная техника"

Изучить информацию по общей программе практики:

1. Структуру, деятельность, продукты для анализа бизнес-процессов, задачи отдела разработки.

2. Используемые на предприятии и в подразделении средства автоматизации.

3. Действующий порядок разработки и использования средств автоматизации: техническую политику отдела разработки в области систем автоматизации, инструменты реализации технической политики, требования к компонентам систем автоматизации и т.п.

Выполнить индивидуальное задание по разработке сервера приложения информационной системы учета медицинских инструментов на C++:

1. Ознакомиться с фреймворком Crow.

2. Разработать документацию серверного API.

3. Написать REST API.

4. Написать API реализующий бизнес-логику в системе.

Руководитель практики от ЛГТУ Старший преподаватель Болдырихин О.В.

Задание принял к исполнению студент Станиславчук С.М.

Аннотация

С. 29. Ил. 6. Литература 8 назв. Прил. 3.

Документ включает в себя описание характеристик предприятия и выполнение задания, выданного руководителем практики. Содержание указанных разделов соответствует стандартам ЕСПД (Единая система программной документации) соответствующих наименований. В работе рассмотрены основные этапы выполнения задания, включая разработку сервера на языке C++ с использованием фреймворка Crow.

Оглавление

Введение

В ходе прохождения практики было знакомство с предприятием ООО «ОНСОФТ». В процессе взаимодействия с сотрудниками отдела разработки ретейл решений было получено задание по созданию программы для учета медицинских инструментов. Для организации разработки было принято решение использовать разделение на задачи. Таким образом были выделены следующие блоки для реализации:

1. Изучить фреймворк Crow.
2. Написать серверную документацию.
3. Реализовать REST API и бизнес-процесс системы.

В результате практики было знакомство с предприятием сведения о котором приводятся в очтете.

В ходе прохождения проектно-технологической практики была проведена первичная разработки темы ВКР «АИС для учета медицинских инструментов».

1. Краткое описание подразделения – места практики

1.1 Характеристика деятельности организации

Организация ООО “ОНСОФТ” занимается разработкой программного обеспечения [1]. Их услуги охватывают все этапы разработки программного обеспечения, включая анализ требований, проектирование архитектуры, разработку, тестирование, внедрение и поддержку.

Услуги предоставляемые компанией ООО «ОНСОФТ»:

- разработка и поддержка высоконагруженных web приложений;
- разработка программного обеспечения для ККТ;
- разработка и развитие retail решений;
- аутстаффинг;
- технический аудит.

1. 2 Характеристика подразделения

Прохождение практики в компании ООО «ОНСОФТ» проходило в отделе разработки ретейл решений. Этот отдел занимается разработкой программного обеспечения для всего спектра retail оборудования: весы, принтеры, сканеры, QR-дисплеи:

- кроссплатформенные драйвера;
- android приложения и сервисы;
- веб-системы мониторинга и управления парком оборудования;
- системы разворачивания и доставки обновлений;
- драйвера библиотек подключаемого оборудования 1С, с возможностью последующей сертификации.

Используемые технологические инструменты (стек):

- Python, TypeScript, JavaScript, Kotlin, C++
- React, Vue, Vite
- PostgreSQL, Redis

Количество сотрудников в отделе ритейл решений: 6.

2. Описание выполнения общей программы

2.1. Фреймворк Crow.

Crow — это минималистичный фреймворк для создания веб-серверов и REST API на C++. Он вдохновлён Flask (Python) и предлагает удобный синтаксис для маршрутизации HTTP-запросов.

Основные цели Crow:

- Простота (без сложных зависимостей)
- Высокая производительность
- Встроенная работа с JSON
- Поддержка асинхронности через `boost::asio`

Фреймворк Crow построен по принципу **маршрутизации HTTP-запросов**, где каждый маршрут привязывается к определённому обработчику.

Основные компоненты:

- **Маршрутизатор (Router)** – анализирует HTTP-запрос и направляет его к соответствующему обработчику.
- **Обработчики (Handlers)** – функции, выполняющие обработку запроса и формирующие ответ.
- **Модуль асинхронности** – использует `boost::asio` для работы в многопоточном режиме.
- **JSON-парсер** – позволяет легко работать с JSON-данными в запросах и ответах.

2.2. Обзор Open API

OpenAPI представляет собой способ представления структуры, включая его эндпоинты, параметры запросов, типы данных, а также возможные ответы. Это позволяет автоматизировать многие процессы

разработки, такие как генерация документации, тестирование, создание SDK для работы с ним. Open API также помогает улучшить взаимодействие между различными сервисами, делая их описание более понятным и доступным. Это достигается тем, что методы, параметры, модели и другие элементы посредством OpenAPI интегрируются с программным обеспечением сервера и всё время с ним синхронизируются.

Спецификация не зависит от языка программирования, и может быть использована вне протокола HTTP. OpenAPI одновременно применяется для клиента, сервера и соответствующей документации интерфейса, созданного согласно REST.

Спецификация декларативна, и поэтому может быть использована клиентами без знаний особенностей серверной реализации. При этом работать с OpenAPI могут как разработчики, так и рядовые пользователи через готовые инструменты и предоставляемые интерфейсы. В качестве формата используется XML и JSON, но в общем случае может быть выбран и другой язык разметки (например, YAML)

2.3. Обзор REST API

REST API — это архитектурный подход, который устанавливает ограничения для API: как они должны быть устроены и какие функции поддерживать. Это позволяет стандартизировать работу программных интерфейсов, сделать их более удобными и производительными.

Слово REST — акроним от Representational State Transfer, что переводится на русский как «передача состояния представления», «передача репрезентативного состояния» или «передача "самоописываемого“ состояния».

В отличие от, например, SOAP API, REST API — не протокол, а простой список рекомендаций, которым можно следовать или не следовать. Поэтому у него нет собственных методов. С другой стороны, его автор Рой Филдинг создал ещё и протокол HTTP, так что они очень хорошо сочетаются, и REST

обычно используют в связке с HTTP. Хотя нужно помнить: REST — это не только HTTP, а HTTP — не только REST.

3. Описание выполнения индивидуального задания

Прежде чем начать работу с фреймворком Crow, стоит установить git.

Для того, чтобы установить его, следует открыть командную строку Linux и ввести соответствующую команду (Arch Linux): «pacman -Sy git» (рисунок 1).

Рисунок 1 – Установка Git при помощи менеджера пакетов pacman

После завершения установки можно перейти к написанию сервера проекта. Для этого создадим проект crow командами

1. git clone https://github.com/CrowCpp/Crow.git
2. cd Crow
3. mkdir build && cd build
4. cmake ..
5. make
6. sudo make install

Помимо этого, нужна библиотека, работающая с PostgreSQL. Для C++ это libpqxx 7.9.2-1. Устанавливается командой: «pacman -Sy libpqxx».

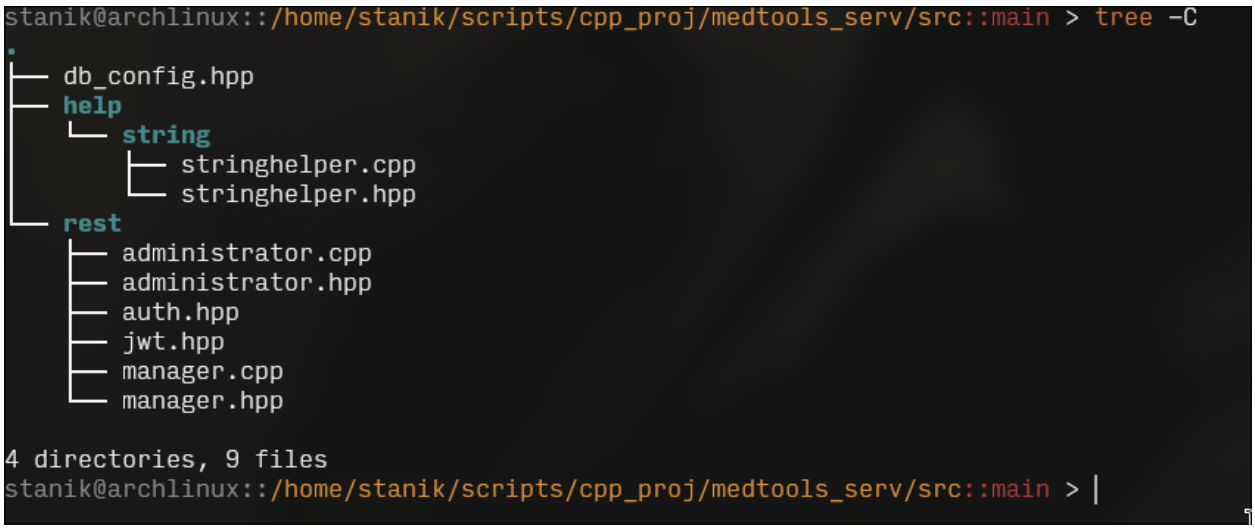


Рисунок 1 – Структура проекта

На данный момент, для выполнения задания буду работать с кластером файлов rest/administrator. Помимо файла, содержащего реализацию REST API для администратора, нужно написать заголовочный файл, в котором описан интерфейс взаимодействия с классом administrator.

Содержимое файла administrator.hpp представлено на рисунке 2.

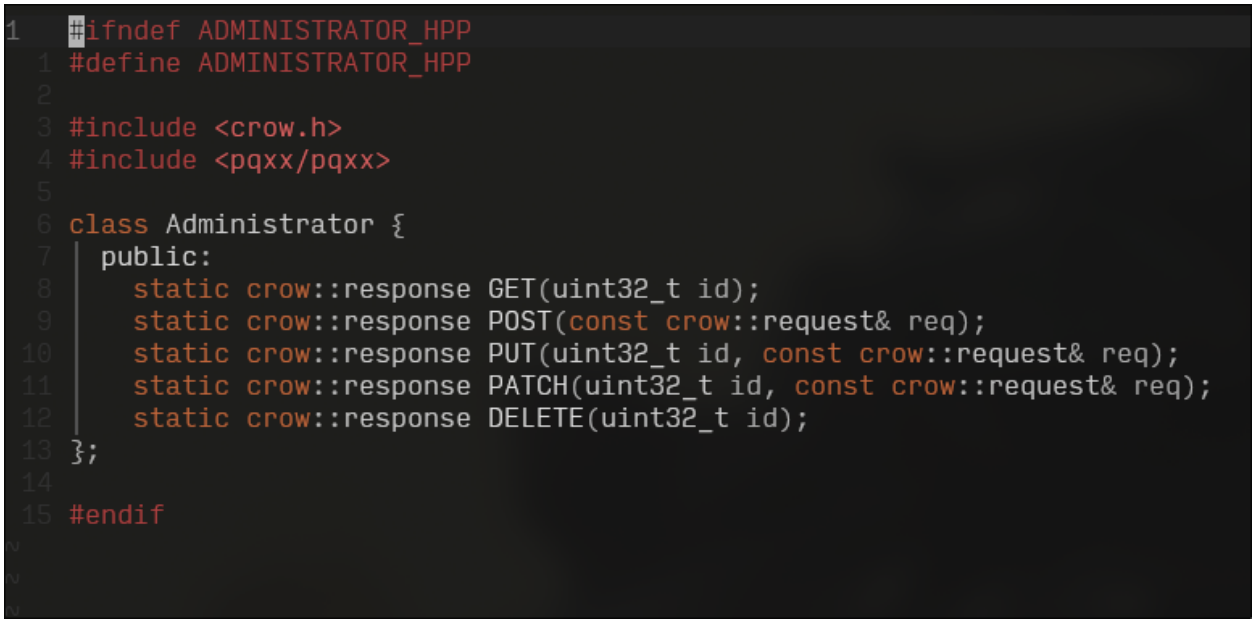


Рисунок 2 – Заголовочный файл для класса «Администратор»

Содержимое файла administrator.cpp, а именно — реализация одного из RESTful запросов — GET, представлено на рисунке 3.

```cpp
#include "./administrator.hpp"
#include "../db_config.hpp"
#include "../help/string/stringhelper.hpp"
#include <exception>

/*
CREATE TABLE Administrator (
    ID SERIAL PRIMARY KEY,
    FirstName TEXT NOT NULL,
    LastName TEXT NOT NULL,
    Email TEXT UNIQUE NOT NULL
);
*/

/// Retrieve by ID
crow::response Administrator::GET(uint32_t id) {
    try {
        pqxx::connection conn(DB_CONN_STR);
        pqxx::work txn(conn);

        pqxx::result res = txn.exec_params("SELECT * FROM Administrator WHERE ID = $1", id);
        txn.commit();

        if (res.empty()) {
            return crow::response(404, "Administrator not found");
        }

        crow::json::wvalue json_res;
        json_res["ID"] = res[0]["id"].as<uint32_t>();
        json_res["FirstName"] = res[0]["first_name"].c_str();
        json_res["LastName"] = res[0]["last_name"].c_str();
        json_res["Email"] = res[0]["email"].c_str();
        json_res["Login"] = res[0]["login"].c_str();
        json_res["Password"] = res[0]["password"].c_str();

        return crow::response(json_res);
    } catch (const std::exception& e) {
        return crow::response(500, std::string("DB error: ") + e.what());
    }
}

/// Create new
```

Рисунок 3 – C++ файл, реализующий класс «Администратор» (GET запрос)

Теперь можно перейти к написанию документации (Open API) для класса «Администратор». Содержимое файла openapi/administrator представлено на рисунке 4.

```yaml
openapi: 3.0.3
info:
  title: Administrator API
  description: API for managing administrators in the system.
  version: 1.0.0

servers:
  - url: http://localhost:8080
    description: Local development server

paths:
  /administrators:
    post:
      summary: Create a new administrator
      description: Adds a new administrator to the database.
      operationId: createAdministrator
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/AdministratorInput"
      responses:
        "201":
          description: Administrator successfully created
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Administrator"
        "400":
          description: Invalid input data
        "500":
          description: Database error

  /administrators/{id}:
    get:
      summary: Get administrator details
      description: Retrieves details of a specific administrator by ID.
      operationId: getAdministrator
      parameters:
        - name: id
          in: path
          required: true
          description: Administrator ID
          schema:
            type: integer
      responses:
        "200":
          description: Administrator found
          content:
```

Рисунок 4 – OpenAPI для работы с «Администратор»

Рассмотрим структуру OpenAPI для администратора.

Заголовок. openapi: 3.0.3 – версия OpenAPI. info содержит название API, его описание и версию.

Сервер API. API будет работать на `http://localhost:8080`

Операции API (paths):

Создание администратора (`POST /administrators`) — для добавления нового администратора в систему. В качестве запроса - данные в формате «JSON», в качестве ответа - «201 created» + JSON-объект администратора.

Получение администратора (`GET /administrators/{id}`) - возвращает информацию о конкретном администраторе по его ID. Например, GET /administrators/5. Ответом будет (если администратор с таким id есть в бд) - «200 OK» + JSON-объект администратора

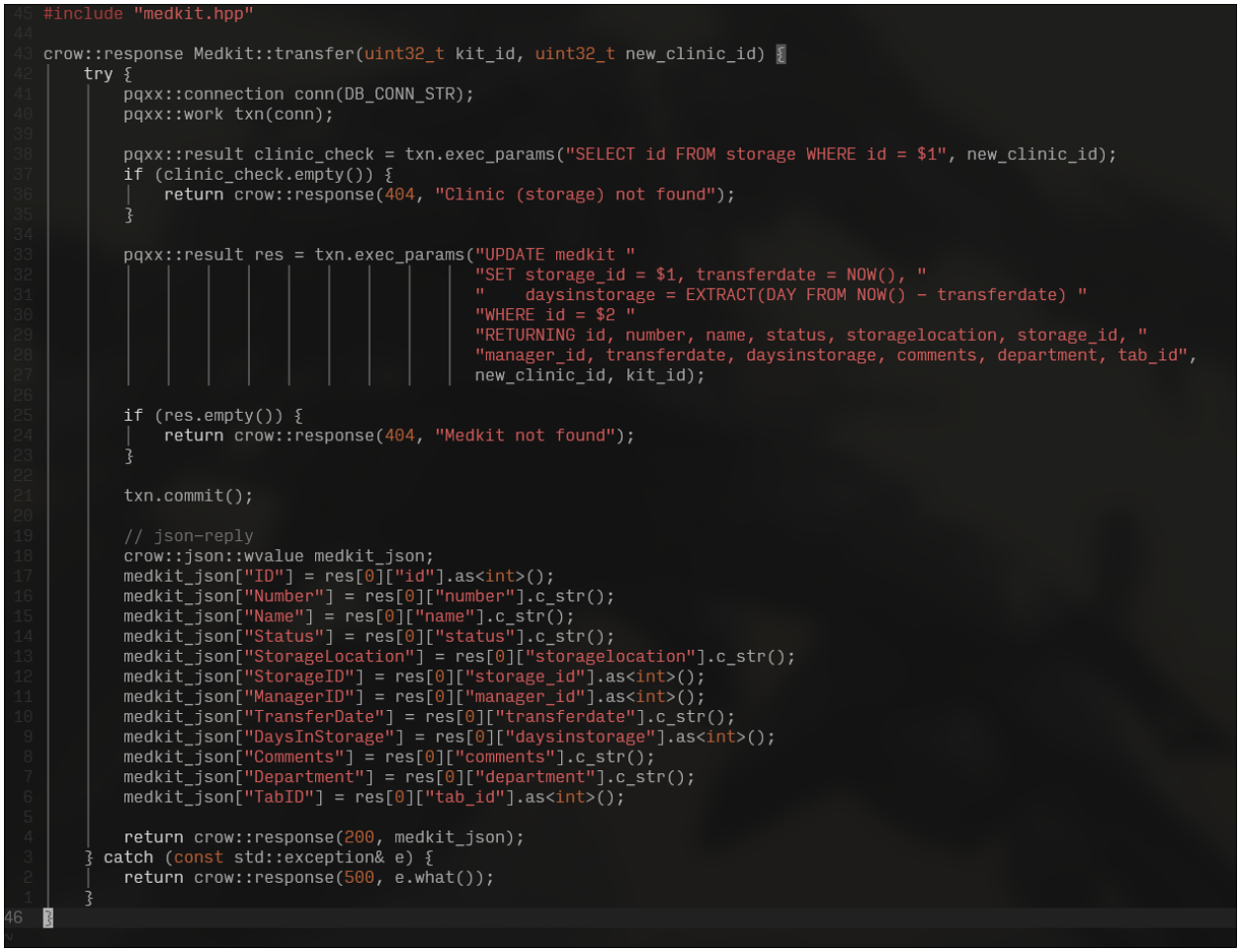
Полное обновление администратора (`PUT /administrators/{id}`) - полностью заменяет данные администратора. В качестве запроса — id администратора и json-тело нового админа, в качестве ответа - «200 OK».

Частичное обновление администратора (`PATCH /administrators/{id}`) - позволяет обновлять только выбранные поля. В качестве запроса — id администратора и json-тело нового админа (поля которого можно выбирать — частичное обновелние), в качестве ответа - «200 OK».

Удаление администратора (`DELETE /administrators/{id}`) - Удаляет администратора из базы данных. Запрос: DELETE /administrators/5. Ответ: `204 No Content` (успешное удаление)

Также в Open API указаны модели (components/schemas) для особых операций. **Administrator** – полная модель администратора. **AdministratorInput** – используется для `POST` и `PUT` запросов (все поля обязательны). **AdministratorPatch** – используется для `PATCH` (все поля опциональны).

И последним этапом я добавлю обработку бизнес-процесса моей системы сервером. Написав для этого серверное API для Medkit. Метод, который я хочу реализовать — это POST-запрос с обновлением клиники. Так как мед. наборы должны перемещаться между клиниками важно, чтобы для этой бизнес-логики был выделен отдельный орган, отвечающий за правильное перемещение медицинского набора в новую клинику. Реализация класса Medkit представлена на рисунке 5.



```cpp
#include "medkit.hpp"

crow::response Medkit::transfer(uint32_t kit_id, uint32_t new_clinic_id) {
    try {
        pqxx::connection conn(DB_CONN_STR);
        pqxx::work txn(conn);

        pqxx::result clinic_check = txn.exec_params("SELECT id FROM storage WHERE id = $1", new_clinic_id);
        if (clinic_check.empty()) {
            return crow::response(404, "Clinic (storage) not found");
        }

        pqxx::result res = txn.exec_params("UPDATE medkit "
                                           "SET storage_id = $1, transferdate = NOW(), "
                                           "    daysinstorage = EXTRACT(DAY FROM NOW() - transferdate) "
                                           "WHERE id = $2 "
                                           "RETURNING id, number, name, status, storagelocation, storage_id, "
                                           "manager_id, transferdate, daysinstorage, comments, department, tab_id",
                                           new_clinic_id, kit_id);

        if (res.empty()) {
            return crow::response(404, "Medkit not found");
        }

        txn.commit();

        // json-reply
        crow::json::wvalue medkit_json;
        medkit_json["ID"] = res[0]["id"].as<int>();
        medkit_json["Number"] = res[0]["number"].c_str();
        medkit_json["Name"] = res[0]["name"].c_str();
        medkit_json["Status"] = res[0]["status"].c_str();
        medkit_json["StorageLocation"] = res[0]["storagelocation"].c_str();
        medkit_json["StorageID"] = res[0]["storage_id"].as<int>();
        medkit_json["ManagerID"] = res[0]["manager_id"].as<int>();
        medkit_json["TransferDate"] = res[0]["transferdate"].c_str();
        medkit_json["DaysInStorage"] = res[0]["daysinstorage"].as<int>();
        medkit_json["Comments"] = res[0]["comments"].c_str();
        medkit_json["Department"] = res[0]["department"].c_str();
        medkit_json["TabID"] = res[0]["tab_id"].as<int>();

        return crow::response(200, medkit_json);
    } catch (const std::exception& e) {
        return crow::response(500, e.what());
    }
}
```

Рисунок 5 — Medkit.cpp (метод transfer)

В этом методе я сначала создаю соединение с бд, после чего проверяю новую клинику на наличие в бд, то же самое делаю и для мед. набора. Выполняю коммит транзакции и генерирую json-ответ с новым мед. набором.

4. Постановка задачи выпускной квалификационной работы

4.1 Описание системы. Пользователи системы

Мной была выбрана тема «АИС для учета и заказов мед. инструментов».

Описание системы:

- информационная система предоставляет пользователям удобный функционал для просмотра, редактирования списка наборов, составления отчётности. Пользователи смогут генерировать заказы на интересующие медицинские инструменты. Не придётся использовать сторонние сервисы;
- информационная система имеет клиент-серверную архитектуру;
- информационная система является десктоп-приложением.

Пользователи системы:

Менеджер – имеет возможности просматривать и заказывать мед. оборудование, используя графический интерфейс приложения.

Администратор – просматривает, создает, редактирует мед. оборудование. Может принимать или отклонять заявки на мед. инструменты. Также может создавать, удалять пользователей в системе, редактировать списки менеджеров и мест хранения (склады, клиники).

4.2 Цели разработки, функции системы, диаграмма вариантов использования.

Целью выпускной квалификационной работы является разработка десктоп-приложения, которое позволит создавать и заказывать медицинские наборы.

Функции системы:

1) Управление статусом каждого мед. набора.
2) Упорядочивание списка наборов.
3) Отображение внутреннего наполнения набора (карточка набора).
4) Редактирование и добавление фото и акта к карточке набора.
5) Хранение и отображение актов на наборы.

6) Ведение журнала перемещений набора, автоматическое обновление при каждом перемещении.
7) Создание отчетов по выбранным клиникам и менеджерам.
8) Редактирование списка наборов
9) Создание таблиц (папок)

Диаграмма использования АИС «Учет и заказы медицинских инструментов» представлена на рисунке 5.

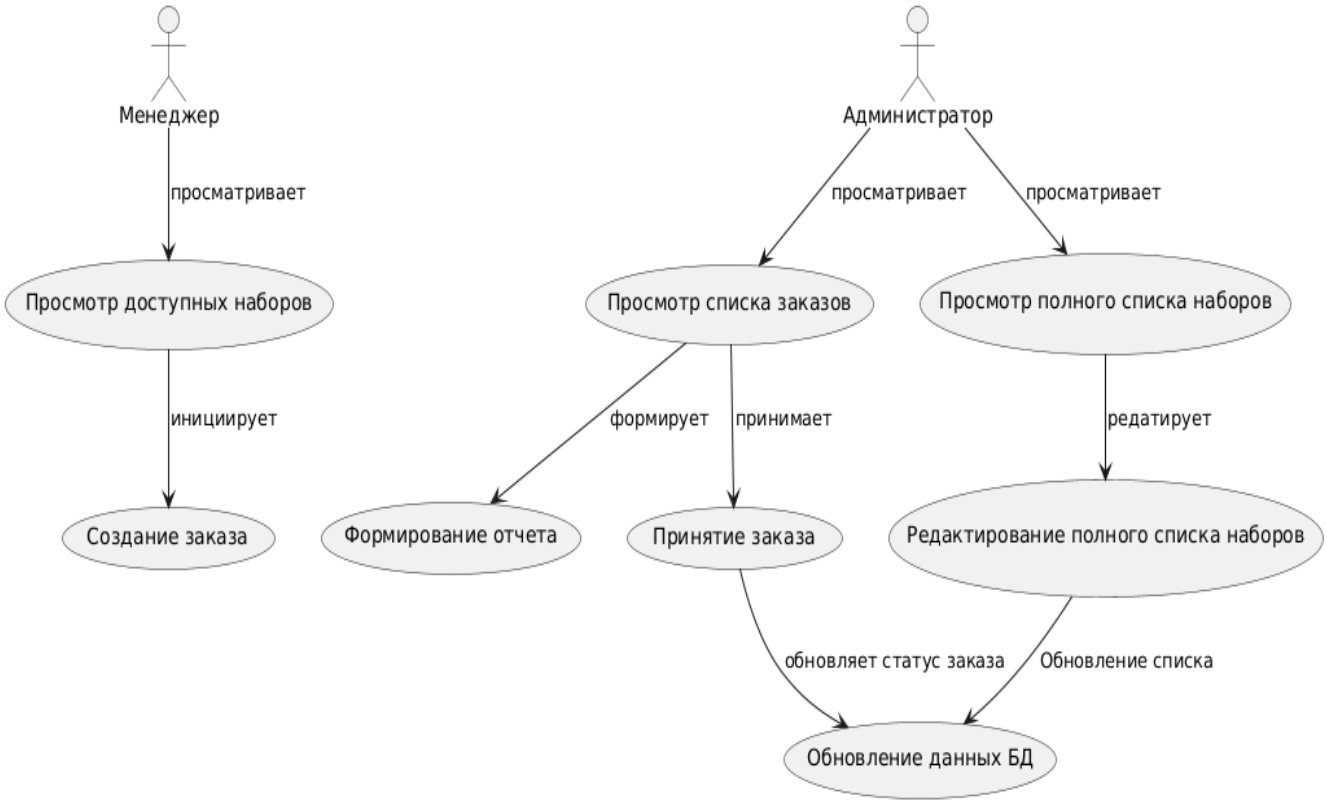


Рисунок 6 – Диаграмма использования

4.3 Сравнение с отечественными аналогами

Изучая существующие приложения, были найдены только веб-сервисы, требующие постоянного подключения к сети интернет: Инвентарь360 [4], FaceKit [5], Eqman [6]. Разрабатываемое мной приложение не будет требовать постоянного подключения к сети: только в момент идентификации и аутентификации. Такой функционал как редактирование списков, добавление и удаление новых элементов будет всё еще сохранять свое состояние, используя сохранение на локальном устройстве пользователя, а

при восстановлении соединения данные будут синхронизованы с удаленной базой данных.

4.4 Решаемые задачи

4.4.1 Учет хранимых медицинских инструментов

Ведение актуального учета всех медицинских инструментов, находящихся на хранении в клинике или на складе. Это включает информацию о каждом инструменте: тип, серийный номер, состояние, срок аренды, дата поступления, история использования и другие параметры.

Система регистрирует каждый инструмент в базе данных, обновляет информацию о его состоянии и предоставляет доступ к данным в режиме реального времени.

4.4.2 Заказы медицинских инструментов

Система предоставляет интерфейс для создания заказов на поставку инструментов. Заказы могут быть обработаны пользователем с учетом наличия инструментов на складе и их местоположения.

Система регистрирует каждый инструмент в базе данных, обновляет информацию о его состоянии и предоставляет доступ к данным в режиме реального времени.

4.4.3 Журналирование перемещений инструментов

Ведение подробного журнала всех перемещений медицинских инструментов между клиниками, складами и другими объектами. Это включает информацию о дате, времени, месте отправки и получения, а также о лицах, ответственных за перемещение.

Система автоматически фиксирует каждое перемещение инструментов, сохраняя данные в журнале. Это позволяет отслеживать историю движения инструментов и контролировать их местоположение.

5. Описание информации, полученной на практике для выполнения ВКР

В ходе прохождения практики в отделе разработки мной были получены практические навыки разработки сервера приложений с использованием фреймворка Crow и языка программирования C++. Данное техническое решение обладает высокой производительностью, что позволит эффективно обрабатывать большое количество одновременных запросов с минимальными задержками.

Также в ходе практики был получен опыт работы с **RESTful API** и **OpenAPI**, что позволило разработать более структурированное и стандартизированное приложение.

Характеристика предметной области:

Медицинские инструменты являются неотъемлемой частью работы любой клиники или больницы. Однако, учитывая их высокую стоимость, клиники нередко предпочитают арендовать инструменты, а не приобретать их в собственность. Это связано с тем, что многие инструменты являются многоразовыми, и их состояние поддерживается специальным персоналом. Однако процесс учета, транспортировки и контроля состояния инструментов часто занимает значительное время, что может негативно сказаться на оперативности оказания медицинской помощи.

В условиях, когда каждая минута на счету, особенно в экстренных ситуациях, сокращение времени ожидания поставки медицинских инструментов может стать решающим фактором для спасения жизни пациента. Поэтому разработка программы, которая автоматизирует процессы учета, транспортировки и контроля состояния медицинских инструментов, является актуальной и крайне важной задачей.

Заключение

По итогам прохождения производственной практики в отделе разработки было выполнено индивидуальное задание, связанное с разработкой REST API на C++ с использованием фреймворка Crow. В ходе

работы были получены навыки проектирования и реализации сервера, а также документирования API с помощью OpenAPI.

Библиографический список

1. ОН-СОФТ [Электронный ресурс] / Режим доступа к данным: https://on-soft.ru/, свободный.

2. Crow framework [Электронный ресурс] / Режим доступа к данным: https://crowcpp.org/master/guides/app/, свободный.

3. Инвентарь 360 [Электронный ресурс] / Режим доступа к данным: http://inventory360.ru/, свободный.

4. Facekit [Электронный ресурс] / Режим доступа к данным: https://facekit.ru/, свободный.

5. Eqman [Электронный ресурс] / Режим доступа к данным: https://eqman.co/, свободный.

6. Методические указания к производственной практике и выпускной квалификационной работе бакалавра [Текст] – Липецк: Издательство Липецкого государственного технического университета, 2024. – 32 с

7. Git Documentation [Электронный ресурс] / Режим доступа к данным: https://git-scm.com/doc, свободный.

Приложение А

Код REST API администратора

Administrator.cpp

```
#include "./administrator.hpp"
#include "../db_config.hpp"
#include "../help/string/stringhelper.hpp"
#include <exception>

/*
CREATE TABLE Administrator (
   ID SERIAL PRIMARY KEY,
   FirstName TEXT NOT NULL,
   LastName TEXT NOT NULL,
   Email TEXT UNIQUE NOT NULL
);
*/

/// Retrieve by ID
crow::response Administrator::GET(uint32_t id) {
   try {
      pqxx::connection conn(DB_CONN_STR);
      pqxx::work txn(conn);

      pqxx::result res = txn.exec_params("SELECT * FROM Administrator WHERE ID = $1", id);
      txn.commit();

      if (res.empty()) {
         return crow::response(404, "Administrator not found");
      }

      crow::json::wvalue json_res;
      json_res["ID"] = res[0]["id"].as<uint32_t>();
      json_res["FirstName"] = res[0]["first_name"].c_str();
      json_res["LastName"] = res[0]["last_name"].c_str();
      json_res["Email"] = res[0]["email"].c_str();
      json_res["Login"] = res[0]["login"].c_str();
      json_res["Password"] = res[0]["password"].c_str();

      return crow::response(json_res);
   } catch (const std::exception& e) {
      return crow::response(500, std::string("DB error: ") + e.what());
   }
}

/// Create new
crow::response Administrator::POST(const crow::request& req) {
   try {
      auto body = crow::json::load(req.body);
      if (!body || !body.has("FirstName") || !body.has("LastName") || !body.has("Email")) {
         return crow::response(400, "Invalid request: Missing required fields");
      }

      pqxx::connection conn(DB_CONN_STR);
```

```
        pqxx::work txn(conn);

        // dao
        // json -> jsno dao
        // bussiness dao
        // data access

        // handler
        // open api - for docs
        // i/o
        // key cloak, face2

        txn.exec_params("INSERT INTO Administrator (FirstName, LastName, "
                        "Email) VALUES ($1, $2, $3)",
                        std::string(body["FirstName"].s()), std::string(body["LastName"].s()),
                        std::string(body["Email"].s()));
        txn.commit();

        return crow::response(201, "Administrator created successfully");
    } catch (const std::exception& e) {
        return crow::response(500, std::string("Database error: ") + e.what());
    }
}

/// FULL Update
crow::response Administrator::PUT(uint32_t id, const crow::request& req) {
    try {
        auto body = crow::json::load(req.body);
        if (!body || !body.has("FirstName") || !body.has("LastName") || !body.has("Email")) {
            return crow::response(400, "Invalid request: Missing required fields");
        }

        pqxx::connection conn(DB_CONN_STR);
        pqxx::work txn(conn);

        pqxx::result res = txn.exec_params("UPDATE Administrator SET FirstName = $1, LastName = $2, Email = "
                                           "$3 WHERE ID = $4 RETURNING ID",
                                           std::string(body["FirstName"]), std::string(body["LastName"].s()),
                                           std::string(body["Email"].s()), id);
        txn.commit();

        if (res.empty()) {
            return crow::response(404, "Administrator not found");
        }

        return crow::response(200, "Administrator updated successfully");
    } catch (const std::exception& e) {
        return crow::response(500, std::string("Database error: ") + e.what());
    }
}

// Partially update
crow::response Administrator::PATCH(uint32_t id, const crow::request& req) {
    try {
        auto body = crow::json::load(req.body);
        if (!body) {
            return crow::response(400, "Invalid JSON");
        }
```

```
        pqxx::connection conn(DB_CONN_STR);
        pqxx::work txn(conn);

        std::vector<std::string> updates;
        pqxx::params params;

        if (body.has("FirstName")) {
            updates.push_back("FirstName = $" + std::to_string(updates.size() + 1));
            params.append(std::string(body["FirstName"].s()));
        }
        if (body.has("LastName")) {
            updates.push_back("LastName = $" + std::to_string(updates.size() + 1));
            params.append(std::string(body["LastName"].s()));
        }
        if (body.has("Email")) {
            updates.push_back("Email = $" + std::to_string(updates.size() + 1));
            params.append(std::string(body["Email"].s()));
        }

        if (updates.empty()) {
            return crow::response(400, "No fields to update");
        }

        // Добавляем ID в параметры
        updates.push_back("ID = $" + std::to_string(updates.size() + 1));
        params.append(id);

        std::string query = "UPDATE Administrator SET " + StringHelper::join(updates, ", ") + " WHERE ID = $" +
                            std::to_string(updates.size()) + " RETURNING ID";

        pqxx::result res = txn.exec_params(query, params);

        if (res.empty()) {
            return crow::response(404, "Administrator not found");
        }

        txn.commit();
        return crow::response(200, "Administrator updated successfully");
    } catch (const std::exception& e) {
        return crow::response(500, std::string("Database error: ") + e.what());
    }
}

// Remove item
crow::response Administrator::DELETE(uint32_t id) {
    try {
        pqxx::connection conn(DB_CONN_STR);
        pqxx::work txn(conn);

        pqxx::result res = txn.exec_params("DELETE FROM Administrator WHERE ID = $1 RETURNING ID", id);
        txn.commit();

        if (res.empty()) {
            return crow::response(404, "Administrator not found");
        }

        return crow::response(200, "Administrator deleted successfully");
    } catch (const std::exception& e) {
```

```
        return crow::response(500, std::string("Database error: ") + e.what());
    }
}
```

Administrator.hpp

```
#ifndef ADMINISTRATOR_HPP
#define ADMINISTRATOR_HPP

#include <crow.h>
#include <pqxx/pqxx>

class Administrator {
  public:
    static crow::response GET(uint32_t id);
    static crow::response POST(const crow::request& req);
    static crow::response PUT(uint32_t id, const crow::request& req);
    static crow::response PATCH(uint32_t id, const crow::request& req);
    static crow::response DELETE(uint32_t id);
};

#endif
```

Приложение Б

Документация (OpenAPI) для администратора

administrator.yaml

```
openapi: 3.0.3
info:
  title: Administrator API
  description: API for managing administrators in the system.
  version: 1.0.0

servers:
  - url: http://localhost:8080
    description: Local development server

paths:
  /administrators:
    post:
      summary: Create a new administrator
      description: Adds a new administrator to the database.
      operationId: createAdministrator
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/AdministratorInput"
      responses:
        "201":
          description: Administrator successfully created
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Administrator"
        "400":
          description: Invalid input data
        "500":
          description: Database error

  /administrators/{id}:
    get:
      summary: Get administrator details
      description: Retrieves details of a specific administrator by ID.
      operationId: getAdministrator
      parameters:
        - name: id
          in: path
          required: true
          description: Administrator ID
          schema:
            type: integer
      responses:
        "200":
          description: Administrator found
          content:
            application/json:
              schema:
```

```
          $ref: "#/components/schemas/Administrator"
    "404":
      description: Administrator not found

put:
  summary: Replace an administrator
  description: Updates all fields of an existing administrator.
  operationId: updateAdministrator
  parameters:
    - name: id
      in: path
      required: true
      description: Administrator ID
      schema:
        type: integer
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/AdministratorInput"
  responses:
    "200":
      description: Administrator updated successfully
    "400":
      description: Invalid input data
    "404":
      description: Administrator not found

patch:
  summary: Partially update an administrator
  description: Updates only specified fields of an administrator.
  operationId: patchAdministrator
  parameters:
    - name: id
      in: path
      required: true
      description: Administrator ID
      schema:
        type: integer
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/AdministratorPatch"
  responses:
    "200":
      description: Administrator updated successfully
    "400":
      description: Invalid input data
    "404":
      description: Administrator not found

delete:
  summary: Delete an administrator
  description: Removes an administrator from the database.
  operationId: deleteAdministrator
  parameters:
```

```
      - name: id
        in: path
        required: true
        description: Administrator ID
        schema:
          type: integer
    responses:
      "204":
        description: Administrator deleted successfully
      "404":
        description: Administrator not found

components:
  schemas:
    Administrator:
      type: object
      properties:
        ID:
          type: integer
          description: Unique identifier of the administrator
        FirstName:
          type: string
          description: Administrator's first name
        LastName:
          type: string
          description: Administrator's last name
        Email:
          type: string
          format: email
          description: Administrator's email address

    AdministratorInput:
      type: object
      required:
        - FirstName
        - LastName
        - Email
      properties:
        FirstName:
          type: string
          description: Administrator's first name
        LastName:
          type: string
          description: Administrator's last name
        Email:
          type: string
          format: email
          description: Administrator's email address

    AdministratorPatch:
      type: object
      properties:
        FirstName:
          type: string
          description: Administrator's first name (optional update)
        LastName:
          type: string
          description: Administrator's last name (optional update)
        Email:
```

type: string
format: email
description: Administrator's email address (optional update)

Приложение В

Реализованная бизнес-логика трансфера мед. инструмента

Medkit.hpp

```
#pragma once

// for cpp file
#include "../db_config.hpp"
#include <crow.h>
#include <pqxx/pqxx>

class Medkit {
  public:
    // Move medkit from prev clinic to new one with transferdate update
    static crow::response transfer(uint32_t kit_id, uint32_t new_clinic_id);
};
```

Medkit.cpp

```
#include "medkit.hpp"

crow::response Medkit::transfer(uint32_t kit_id, uint32_t new_clinic_id) {
    try {
        pqxx::connection conn(DB_CONN_STR);
        pqxx::work txn(conn);

        pqxx::result clinic_check = txn.exec_params("SELECT id FROM storage WHERE id = $1", new_clinic_id);
        if (clinic_check.empty()) {
            return crow::response(404, "Clinic (storage) not found");
        }

        pqxx::result res = txn.exec_params("UPDATE medkit "
                                   "SET storage_id = $1, transferdate = NOW(), "
                                   "    daysinstorage = EXTRACT(DAY FROM NOW() - transferdate) "
                                   "WHERE id = $2 "
                                   "RETURNING id, number, name, status, storagelocation, storage_id, "
                                   "manager_id, transferdate, daysinstorage, comments, department, tab_id",
                                   new_clinic_id, kit_id);

        if (res.empty()) {
            return crow::response(404, "Medkit not found");
        }

        txn.commit();

        // json-reply
        crow::json::wvalue medkit_json;
        medkit_json["ID"] = res[0]["id"].as<int>();
        medkit_json["Number"] = res[0]["number"].c_str();
        medkit_json["Name"] = res[0]["name"].c_str();
        medkit_json["Status"] = res[0]["status"].c_str();
        medkit_json["StorageLocation"] = res[0]["storagelocation"].c_str();
        medkit_json["StorageID"] = res[0]["storage_id"].as<int>();
        medkit_json["ManagerID"] = res[0]["manager_id"].as<int>();
        medkit_json["TransferDate"] = res[0]["transferdate"].c_str();
        medkit_json["DaysInStorage"] = res[0]["daysinstorage"].as<int>();
        medkit_json["Comments"] = res[0]["comments"].c_str();
```

```
        medkit_json["Department"] = res[0]["department"].c_str();
        medkit_json["TabID"] = res[0]["tab_id"].as<int>();

        return crow::response(200, medkit_json);
    } catch (const std::exception& e) {
        return crow::response(500, e.what());
    }
}
```