Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Алтайский государственный университет»

Отчёт

по производственной эксплуатационной практике
«Разработка информационной системы для работы с базой данных»

Выполнили: студенты 595 группы
_____ Осипов А.В.
_____ Лаптев А.В.
Проверил: руководитель практики
_____ Кошманов Д.С.

Барнаул, 2022 г.

Оглавление

Введение

В современном мире существует множество информационных систем, предназначенных для разных нужд и использующих в качестве хранилища данных выделенные вычислительные комплексы, называемые базами данных. Основное предназначение баз данных – это оптимальное хранение больших объёмов информации и предоставление доступа к ним, в то время как задачами по обработке этой информации занимаются другие элементы информационной системы. Данные элементы в свою очередь принято делить на две роли: клиент и сервер. Клиенты занимаются в основном выводом информации для конечных пользователей и приёмом новых данных от них, в то время как на сервер ложится основная вычислительная нагрузка при обработке данных, которые могут поступать сразу от нескольких клиентов.

Большинство систем управления базами данных (СУБД) предоставляют интерфейс доступа к данным для вычислительных машин любого типа, что в теории позволяет клиентам напрямую обращаться к базе данных для ввода-вывода необходимой им информации, однако на практике такой подход используется очень редко из-за непрактичности и риска безопасности хранимых данных, поэтому даже для простых задач, не требующих больших вычислительных ресурсов, для распределённого хранения и защиты данных используют промежуточные серверы, через которые совершаются все допустимые операции над базой данных, запрашиваемые клиентами.

В рамках данной работы будет решаться задача о разработке информационной системы с простой реляционной базой данных, хранящей информацию в виде как минимум двух связанных таблиц с разными видами данных, а также клиентское и серверное приложения.

В качестве требований, выдвинутых к серверной части продукта, выступают:

- использование базы данных с СУБД PostgreSQL;
- написание серверного приложения на языке программирования C++ с применением открытых библиотек;
- проектирование программного интерфейса (API) сервера согласно парадигме REST;
- поддержка в качестве платформы для работы сервера отечественной операционной системы AstraLinux.

Для клиентской части выдвинуты следующие требования:

- написание клиента на языке программирования C++ с применением открытых библиотек (при необходимости);
- использование фреймворка Qt 5 для создания графического интерфейса клиента;
- поддержка в качестве платформы отечественной операционной системы AstraLinux.

Решение этой задачи можно разбить на следующие этапы:

1. Создание функциональной схемы (модели) обмена данными между частями информационной системы (ИС).
2. Обзор технологий, требуемых к использованию в работе, а также других средств, применяющихся при разработке аналогичных программных решений.
3. Выбор средств для реализации компонентов ИС и организации взаимодействия между ними.
4. Создание серверного приложения и развёртка среды для его работы
5. Проверка работы серверного приложения и его взаимодействия с другими компонентами ИС.

1. ТЕХНОЛОГИИ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ СИСТЕМ

1.1 Описание реализуемой системы

Как уже было обозначено в введении в качестве основных требований к выполнению практического задания является использование в разрабатываемой информационной системе клиент-серверной архитектуры и базы данных, причём клиент не должен иметь прямого доступа к базе данных и выполнять все действия с ней через сервер. Таким образом формируется цепь связи, по которой клиент отправляет запрос серверу через внешнюю сеть (например, локальную сеть предприятия или через Интернет), который в свою очередь обрабатывается сервером и перенаправляется базе данных по внутренней сети. Ответ от базы данных в виде запрашиваемых данных или отчёта об изменении её состояния отправятся клиенту по той же цепи, но в обратном направлении.

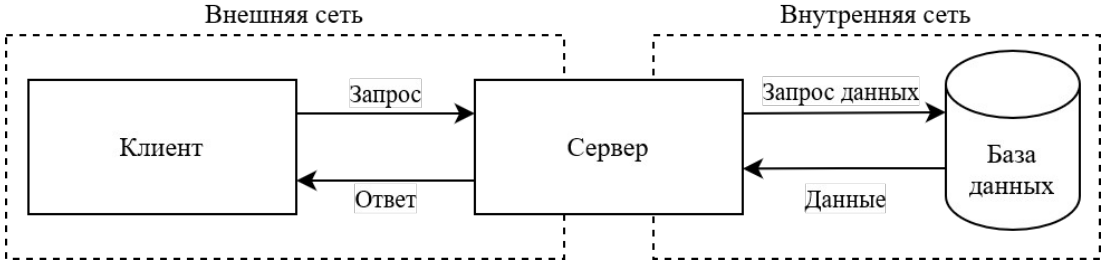

Рис. 1.1. Схема обмена данными в информационной системе

Сервер в такой цепи играет роль посредника между клиентом и базой данных: он занимается преобразованием запросов и ответов в требуемый формат, предотвращает несанкционированные операции над базой данных со стороны клиента, а также выполняет задачу по кэшированию часто запрашиваемой информации, что уменьшает нагрузку на СУБД и время отклика на такие запросы.

Предназначение создаваемой информационной системы и структура её базы данных будет следующими: система будет заниматься хранением информации о посещаемости студентов в университете. Для этого в базе данных создаётся две связанных таблицы: таблица основных данных

о студентах (ФИО и номер группы) и таблица записей о посещаемости (в виде отметки «присутствовал» или «отсутствовал» в определённое время). Связь этих таблиц будет осуществятся по идентификационному номеру (ИД) студента в базе данных, который будет являться одновременно основным ключом записи в таблице студентов и частью составного ключа записи о посещаемости наравне с датой и временем отмечаемого занятия. Данная связь имеет тип «один-ко-многим», поскольку одному студенту может соответствовать несколько записей о посещаемости. Дополнительно в модель данных включается таблица групп студентов, поскольку хранение номеров групп студентов в виде строчного поля (номера групп не всегда могут быть представлены в виде чисел в строгом смысле этого слова) является не совсем оптимальным и нарушает принципы нормализации данных.

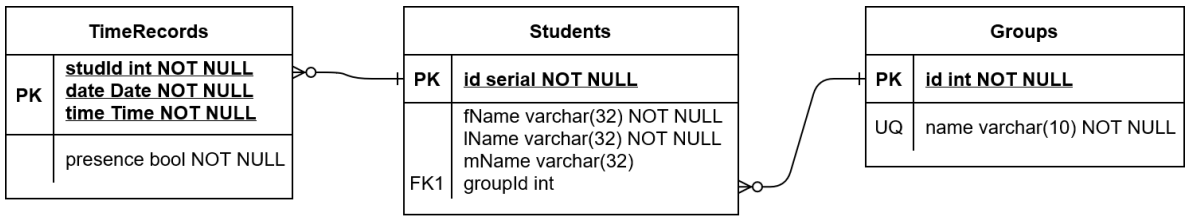

Рис. 1.2. Модель данных ИС, выполнения на языке UML

Клиент может запрашивать у сервера списки групп и состоящих в них студентов, просматривать, создавать и редактировать записи о посещаемости студентов (с фильтрацией по дате и времени занятия и по группе студентов), а также добавлять новых студентов и группы в базу данных или удалять их. Сервер должен удостовериться в правильности подаваемых запросов и перевести их на понятный для СУБД язык – SQL, а СУБД – выполнить их или уведомить сервер о невозможности их выполнения с дальнейшим уведомлением клиента об успешности транзакции.

1.2 Программный интерфейс серверного приложения (API)

Для организации обменных процедур и взаимодействия между двумя программами используют специальный набор соглашений ввода-вывода, называемый программным интерфейсом приложения. В технической литературе, в том числе на русском языке, для обозначения этого понятия чаще используют англоязычное сокращение «API» (Application Programming Interface). Суть API заключается в выполнении действий одной программой по запросу, который она принимает от другой программы по отдельному каналу ввода-вывода. В ответ принимающая программа может отправить информацию по выполнению запроса, либо же встречный запрос второй программе. Таким образом взаимодействие двух программ, которые могут быть запущены на разных компьютерах и быть написанными на разных языках и для разных платформ, практически ничем не отличается от межпроцессного взаимодействия во многозадачных операционных системах.

Большинство API рассчитаны на взаимодействие программ через стандартные сетевые протоколы, что обеспечивает их универсальность и простоту масштабирования систем с их применением. В эпоху развития веб-технологий для вызовов API всё чаще стал использоваться протокол HTTP, который первоначально предназначался для передачи гипертекста, но позволяет также передавать и другие виды данных. Удобная система адресации и разделения типов запросов стали дополнительными причинами распространённости данного протокола в организации межпрограммного взаимодействия, в том числе и вне World Wide Web, и вдохновили на создание парадигмы REST, которая будет рассмотрена в другой главе.

1.3 Использование протокола HTTP для обмена информацией

HTTP (от англ. Hypertext Transfer Protocol) – это двухсторонний протокол связи, который позволяет передавать гипертекст и другие данные, необходимые

для функционирования различных веб-ресурсов, включая вызовы функций Web API. В основе HTTP лежит запросов сообщений в текстовом виде, который можно разделить на три части[1]:

1. Стартовая строка, описывающая тип запроса, URI-адрес запрашиваемого ресурса (веб-страницы или функции API) и версию протокола HTTP (в настоящий момент самой распространённой версией является 1.1). В стартовой строке ответа указывается только версия протокола и статус выполнения запроса.

2. Заголовок сообщения, который может содержать различные сведения о его назначении, содержании, отправителе и получателе.

3. Тело сообщения, которое несёт основную полезную нагрузку в виде текстовых или бинарных данных. Тело сообщения чаще всего используется для передачи аргументов функции API или данных для отображения и обработки на стороне клиента.

Метод HTTP-запроса, задаваемый в его стартовой строке, определяет вид взаимодействия отправителя (клиента) с ресурсом. Чаще всего используются методы GET (получение информации из ресурса) и POST (передача информации ресурсу), но нередко применяются также заголовки с более конкретным предназначением, такие как PUT (выгрузка данных в ресурс) или DELETE (удаление ресурса). В ответ на запрос с любым из этих методов HTTP-сервер должен выполнить соответствующее действие и вернуть клиенту ответ с указанием статуса запроса и возвратных данных. Даже если запрос не удалось обработать корректно, в большинстве случаев сервер отсылает в теле ответа строку с указанием статуса и причины ошибки запроса, как правило, дублируя информацию из стартовой строки ответа. Полученная из ответа информация в дальнейшем так же подлежит обработке со стороны клиента и выводу конечному пользователю информационной системы.

1.4 Формирование тела HTTP-сообщения с помощью JSON

В отличие от формы содержания стартовой строки и заголовка HTTP-сообщения, его тело не имеет стандартизованной формы передачи данных. Ранее данную роль исполнял язык разметки гипертекста HTML, однако когда применение протокола HTTP вышло далеко за пределы его первоначального предназначения, использование данного формата для многих целей стало просто неудобным. Тогда же появились альтернативные форматы передачи текстовых данных, более удобные для их считывания и обработки. Одним из таких форматов является JSON (сокр. от англ. «Javascript Object Notation»), который был разработан американским программистом Дугласом Крокфордом на основе языка программирования Javascript образца 1999 года [2] и до сих пор широко используется для хранения и передачи структурированных данных.

```
"firstName": "John",
"lastName": "Smith",
"isAlive": true,
"age": 27,
"address": {
  "streetAddress": "21 2nd Street",
  "city": "New York",
  "state": "NY",
  "postalCode": "10021-3100"
},
"phoneNumbers": [
  {
    "type": "home",
    "number": "212 555-1234"
  },
  {
    "type": "office",
    "number": "646 555-4567"
  }
],
"children": [],
"spouse": null
```


Рис. 1.3. Пример текстовых данных в формате JSON

Являясь «потомком» Javascript, JSON использует тот же синтаксис и терминологию описания типов данных, которые можно передавать с помощью этого формата. Данные, описываемые в JSON, могут представлять из себя десятичные целые и дробные числа (с разделительной точкой), символьные строки в двойных кавычках или литералы логических значений «true» и «false» («истина» и «ложь») без кавычек. Также любой JSON-файл в своей основе содержит хотя бы одну сложную структуру данных:

упорядоченный массив или объект (ассоциативный массив), которые могут свободно включаться друг в друга наравне со скалярными данными. Массивы в JSON обрамляются квадратными скобками, а объекты – фигурными. Элементы этих структур разделяются запятыми. Ключ свойства (поля) объекта обязательно должен быть строкой и отделяться от его значения двоеточием. Такой простой формат синтаксиса и его полная совместимость с Javascript делает обработку JSON легко реализуемой практически на любом языке программирования. Впрочем, существует множество готовых библиотек для обработки JSON, часть из которых перечислена на его официальном сайте.

1.5 Парадигма REST: основные принципы и преимущества использования

Существует множество способов организации взаимодействия компонентов информационной системы, который имеют разные преимущества и недостатки, однако для грамотной и эффективной организации масштабируемых систем с большим количеством пользователей и интегрируемых подсистем специалисты выделяют универсальные рекомендации, которые актуальны для многих ресурсов в сети Интернет и в других крупных компьютерных сетях. Набор данных рекомендаций в последствие был сформирован в единый архитектурный стиль (парадигму) разработки ПО, получившее название REST (от англ. Representational State Transfer – «передача репрезентативного состояния»). Данная парадигма основана на пяти основных требованиях для REST-систем и шестом дополнительном:

1. Модель «клиент-сервер». Все элементы системы должны делится на эти две роли и выполнять только связанную со своей ролью часть работы. Клиент должен предоставлять удобный интерфейс ввода-вывода для пользователя и заниматься лишь подготовительной обработкой данных, в то время как основные вычислительные ресурсы сервера должны быть

направлены на глубокую обработку и безопасное хранение уже форматированной информации.

2. Отсутствие состояния (англ. stateless). При взаимодействии клиента с сервером последний не должен хранить какую-либо информацию об его текущем состоянии. Вместо этого каждый запрос к серверу должен содержать такой пакет данных, чтобы он мог продолжить взаимодействие с клиентом на любом этапе процесса, не имея при этом данных о предыдущих операциях клиента. Такой подход позволяет хранить на стороне клиента устойчивые состояния процесса, актуальные только для него, и не нагружать этой информацией сервер. В свою очередь на сервер возлагается функция управления переходами между этими состояниями, когда клиент делает соответствующий запрос с указанием состояния, из которого он хочет выйти.

3. Кэширование. Сервер должен обеспечивать кэширование часто запрашиваемых данных. Ответы сервера, в свою очередь, должны иметь явное или неявное обозначение как кэшируемые или некэшируемые с целью предотвращения получения клиентами устаревших или неверных данных в ответ на последующие запросы. Правильное использование кэширования способно частично или полностью устранить некоторые клиент-серверные взаимодействия, ещё больше повышая производительность и масштабируемость системы.

4. Единообразие интерфейса. Каждый ресурс системы по возможности должен иметь такой же формат обращения, как и другие её ресурсы. На базовом уровне это единообразие поддерживается при использовании протокола HTTP, поскольку все ресурсы в нём имеют заранее обозначенный формат запроса, а также возможность унификации адреса и методов взаимодействия с сервером. Также «хорошим тоном» считается указание метода обработки поступающего запроса, в первую очередь языка, на котором описано тело запроса, чтобы сервер мог правильно реагировать

на разные варианты подачи одного и того же запроса. Делать это можно через URI-параметры запроса или стандартный HTTP-заголовок «Content-Type», который описывает MIME-тип поступающего сообщения.

Другим методом унификации интерфейса является представление в ответе на запрос ресурса списка возможных взаимодействий с ним (в виде ссылок на другие ресурсы или допустимых параметров запроса к текущему), что позволяет получать клиенту актуальный список возможных взаимодействий по мере обращения к серверу. Такая модель управления состояниями приложения называется HATEOAS (от англ. Hypermedia as the Engine of Application States) и основана на свойствах гипермедиа (англ. hypermedia) – динамически и недетерминировано подгружаемых медиаресурсов, схожих по этим свойствам с гипертекстом.

5. Слоистая архитектура. Сервер на программном и (или) физическом уровне не должен быть представлен в виде монолитного узла, выполняющего все возложенные на него функции. Вместо этого для масштабируемых систем должна применяться многослойная архитектура, где разные функции сервера выполняются разными частями системы (внутренней сети) серверного приложения. Взаимодействие системы с клиентами происходит только через её верхний слой – промежуточные серверы, на которых лежит функция по обработке запросов, распределению нагрузки между остальными слоями системы и их логическому экранированию. Клиент не может определить (и не обязан определять), что за одним сервером скрывается целая сеть устройств для распределённого выполнения выданной им задачи, что обеспечивает гибкость внутренней архитектуры сервера и безопасность обрабатываемых данных.

Описанная в пункте 1.1 схема обмена данных является примером простейшей многослойной серверной архитектуры. В ней клиент работает не напрямую с базой данных, а с сервером, который переправляет запросы к ней.

Если на место одной базы данных поставить распределённый кластер баз данных, то алгоритм взаимодействия клиента с сервером не поменяется, поскольку оно реализовано отдельно от взаимодействия сервера с базой данных (или кластером БД).

6. Код по требованию. В некоторых случаях сервер может не только управлять стандартными функциями клиента посредством HATEOAS и ответов на stateless-запросы, но и расширять его с помощью загружаемых дополнений (апплетов, плагинов, скриптов) для клиента, тем самым обеспечивая конечному пользователю весь необходимый ему функционал. По такому принципу работают, к примеру, расширяемые онлайн-редакторы или постоянно обновляемые приложения. Однако, такой подход является не всегда допустимым и легко реализуемым, поэтому данное требование считается необязательным. К примеру, в рамках данной задачи нет смысла реализовывать расширяемость по требованию, поскольку весь необходимый функционал клиента обозначен заранее.

1.6 Создание клиента с графическим интерфейсом (GUI)

Помимо создание серверной части, в задачи работы также разработка клиента на основе графической библиотеки Qt 5. Основное предназначение клиента в данной работе – это обеспечение графического интерфейса конечного пользователя и преобразование вводимых данных в понятный серверу формат. Реализация клиента является независимой от реализации сервера и не обязана использовать ту платформу и те же технологии разработки, что и сервер. Главное, чтобы клиент мог корректно работать с API сервера, используя обозначенный его разработчиками протокол обмена данных и формат сообщений. В этом и заключаются одни из главных преимуществ клиент-серверной архитектуры – гибкость и меньшая зависимость от конкретных платформ.

2. РАЗРАБОТКА И РАЗВЁРТЫВАНИЕ СЕРВЕРНОГО ПРИЛОЖЕНИЯ

2.1 Описание используемых программных средств

В качестве требований к реализации сервера были упомянуты использование СУБД PostgreSQL и языка программирования C++ для разработки серверного приложения, а также поддержка операционной системы AstraLinux в качестве платформы для его запуска. Исходя из этих требований был список программных средств (библиотек, компиляторов, сред разработки), которые будут использованы для выполнения практического задания:

- Отечественная операционная система AstraLinux «Орёл» на основе ядра GNU/Linux и его дистрибутива Debian 9 (без графической оболочки);
- Система управления базами данных PostgreSQL 9.6.20;
- Компилятор G++ 6.3.0 для C++ из набора GNU Compiler Collection;
- Редактор кода Emacs 24.5.1 (версия для командного интерфейса). Данный редактор известен своей большой расширяемостью и поддержкой многих языков программирования и разметки, включая C++ и JSON, а также языка сценариев для встроенного в GNU/Linux сборщика Make.
- Многоцелевая библиотека Poco 1.11.2 для C++ (собрана из открытого исходного кода). Данная библиотека обладает большим количеством функций, включая создание асинхронного HTTP-сервера, чтение и запись данных в формате JSON и работу с различными СУБД, включая PostgreSQL.
- Инструменты управления сборкой Make (для сборки разрабатываемого приложения) и Cmake (для сборки Poco в виде набора динамических библиотек Linux).

Для клиента список будет немного отличаться, поскольку при его создании учитываются иные требования и задачи:

- Отечественная операционная система AstraLinux «Орёл» с графической оболочки Fly;
- Компилятор G++ 6.3.0 для C++ из набора GNU Compiler Collection;
- Многоцелевая библиотека (фреймворк) Qt 5.11. Помимо обеспечения GUI, имеет также набор функций по работе с JSON, отправке и приёму файлов с помощью протокола HTTP, что покрывает все функциональные потребности клиента.
- Интегрированная среда разработки (IDE) Qt Creator 5.11 – официальная IDE для разработки приложений на базе Qt, которая включает в себя редактор кода на C++, графический редактор форм окна и другие продвинутые функции для разработки Qt-приложений, отсутствующие в сторонних средах разработки.
- Инструмент сборки qmake для приложений, созданных в Qt Creator.

2.2 Установка и настройка программных средств в клиенте

Процесс установки AstraLinux на рабочий компьютер во многом аналогичен процессу установки многих других современных операционных систем на базе GNU/Linux. Однако, в отличие от многих других дистрибутивов Linux, официальный установочный образ AstraLinux не содержит «живого» окружения для временной работы и ознакомления с операционной системой, поэтому его запуск с параметрами по умолчанию сразу приводит к запуску графического установщика системы. В ходе установки пользователь производит разметку диска, на котором будет установлена система и задаёт начальные параметры её работы, включая имя пользователя, дату и время, настройки безопасности и начальный набор программ.

Большая часть необходимых для разработки клиента программ уже присутствуют в установочном комплекте AstraLinux, поэтому для их установки не требуется даже подключения к Интернету. Достаточно выбрать пункт «Средства разработки» на этапе выбора пакетов для установки вместе с базовой системой. Однако, для полного их функционирования рекомендуется обновить программные пакеты после установки системы с помощью системной утилиты apt (Aptitude) при подключении к Интернету.

При первом запуске Qt Creator попросит пользователя создать профиль Qt, который использоваться для сборки программ, созданных на основе этого фреймфорка. В большинстве случае программа сама задаст все настройки профиля на основе установленной версии Qt и компилятора C++. Достаточно будет лишь подтвердить создание профиля и создать проект с его использованием.

2.3 Установка и настройка программных средств на сервере

В ходе установки программ для сервера возникли некоторые трудности. В частности, при установке операционной системы AstraLinux пришлось использовать текстовый (псевдографический) установщик вместо графического, поскольку серверное оборудование не поддерживало стабильный вывод графики. Этим же объясняется отказ от установки графической оболочки Fly и использование редактора Emacs, который поддерживает работу из командной оболочки, вместо более современных и распространённых решений, работающих исключительно в графическом режиме.

Также трудности возникли с установкой библиотеки Poco, поскольку она отсутствовала в репозиториях AstraLinux, а опционально подключаемые репозитории Debian 9 содержат устаревшую версию библиотеки, в которой отсутствуют функции для работы с JSON и PostgreSQL. В конечном счёте пришлось осуществить сборку последней версии библиотеки из её исходного

кода с помощью инструмента сборки Cmake.

Установка остальных программ не вызвала особых трудностей, поскольку они присутствовали в репозитории или установочных файлах AstraLinux в удовлетворительной конфигурации. Для установки этих программ использовалась встроенная утилита apt (Aptitude), которая управляет программными пакетами для Linux в дистрибутивах на основе Debian. Большинство установленных программ не требовало дополнительной настройки, за исключением PostgreSQL, в котором было необходимо задать пароль для пользователя СУБД и создать базу данных со всеми необходимыми таблицами.

2.4 Выделение адресного пространства ресурсов HTTP-сервера

Перед дальнейшей разработкой серверной части следует определить список HTTP-ресурсов сервера и стандартизировать методы обращения к ним согласно четвёртому пункту концепции REST. В данном случае можно чётко выделить две группы данных: это данные о группах и студентах и данные о посещаемости. Последний вид данных можно выделить в один ресурс, настраиваемый с помощью URI-параметров, поскольку пользователя навряд ли будет интересовать отметка о посещении конкретного занятия конкретным студентом, зато ему может понадобится фильтрация целого набора записей по группе студентов или по дате и времени занятия. Подобные параметры не принято выносить в основную часть HTTP-адреса, потому что работа с ним должна быть похожа на удалённую работу с файловой системой.

Записи о студентах уже поддаются большой структуризации. В частности можно отдельно выделить уровень работы со списком групп и вложенный в него уровень работы со списком студентов в отдельной группе. Поскольку студент, как правило, постоянно прикреплён к определённой группе и наша база данных содержит лишь базовые сведения о студентах, выделять отдельную

категорию ресурсов для работы с данными о конкретном студенте так же нет особого смысла.

Синтаксис обращения к выделенным ресурсам сервера можно изложить в виде следующей таблицы:

Адрес ресурса	**Предназначение**	**Метод запроса**	
		GET	**POST/PUT/DELETE**
/timerecords	Посещаемость студентов	Фильтры: group=группа date=дата time=время presence=присут.	action – действие: • add – добавить запись • alter – изменить • remove – удалить data – данные для изменения (массив объектов): • id – ИД студента • date – дата занятия • time – время занятия • present – присутствие defaults – значения полей по умолчанию (для уменьшения размера запроса)

Адрес ресурса	**Предназначение**	**Метод запроса**	
		GET	**POST/PUT/DELETE**
/groups	Список групп	–	action – действие: • add – добавить группу • alter (rename) – переименовать • remove – удалить data – данные для изменения (массив объектов): • name – текущее имя группы • newName – новое имя группы (для rename)
/groups/*	Студенты группы «*»	nameFormat=формат ФИО: split – раздельно, full – одной строкой, initals – «Фамилия И.О.»	action – действие: • add – добавить студента • rename – переименовать • move – перевести в другую группу • alter – обобщённая операция изменения (может вести себя как «rename» или как «move», в зависимости от набора данных). • remove – удалить data – данные для изменения (зависит от действия, см. 2.*)
/groups/all	Студенты всех групп	Аналогично /groups/*	Аналогично /groups/*

Табл. 1. Ресурсы HTTP-сервера

Здесь параметры GET-запросов подаются с помощью URI-параметров, а параметры для остальных методов – с помощью JSON-тела запроса. Стандарты не дают чёткого разграничения между сферами применения POST и PUT запросов, поэтому они объединены в один столбец. Столбец DELETE также объединён с остальными изменяющими методами, но для него значение свойства JSON «action» будет всегда равно «remove».

2.5 Алгоритм работы клиента

Клиент представляет из себя графическое приложение на Qt, большая часть дизайна которого настраивается в конструкторе форм Qt Designer. Данный конструктор позволяет задать набор элементов окна, их название в коде программы и их расположение в пределах окна с помощью визуального редактирования. Таким образом большая часть кода, связанного с инициализации графического интерфейса, генерируется автоматически самим фреймворком и не входит непосредственно в состав основной программы.

Взаимодействие клиента с сервером осуществляется с использованием HTTP-запросов последнему при открытии окон, нажатиях на кнопки, выборе данных (группы, даты занятия, время занятия) в предусмотренных для этого виджетах, а также, внесении, удалении и изменении каких-либо данных, в процессе работы клиента.

Для отправки HTTP-запросов со стороны клиента использовались встроенные средства Qt, в частности, модуль Qt Network[3] и его классы QNetworkAccsessManager, QNetworkRequest, QnetworkReply. Данный модуль предоставляет возможности для выполнения различных сетевых операций, в том числе и реализации HTTP-запросов. API доступа к сети построен вокруг объекта QNetworkAccessManager, который содержит общую конфигурацию и настройки для отправляемых им запросов. Он содержит конфигурацию прокси и кэша, а также сигналы, связанные с такими проблемами, и сигналы ответа, которые можно использовать для отслеживания хода выполнения сетевой

операции. После создания объекта QNetworkAccessManager приложение может использовать его для отправки запросов по сети.

Имеется группа стандартных функций, которые принимают запрос и дополнительные данные, и каждая из них возвращает объект QNetworkReply. Возвращенный объект используется для получения любых данных, возвращаемых в ответ на соответствующий запрос. QNetworkRequest, в свою очередь, содержит запрос, который нужно отправить с помощью QNetworkAccessManager. Данный класс одержит информацию, необходимую для отправки запроса по сети.

Для формирования тела HTTP-запроса в JSON формате использовались следующие классы: QJsonDocument, QJsonObject, QJsonArray, QJsonValue – из модуля Qt Core. Класс QJsonDocument предоставляет способ чтения и записи документов JSON. Этот класс, упаковывает полный документ JSON и может читать и записывать этот документ как из текстового представления в кодировке UTF-8, так и из собственного двоичного формата Qt. К документу можно отправить запрос о том, содержит ли он массив или объект. Массив или объект, содержащийся в документе, можно получить, а затем прочитать или изменить. При помощи QJsonArray, QJsonObject, QJsonValue можно инкапсулировать массив, объект и значение JSON, соответственно.

2.5.1 Алгоритм работы главного (начального) окна программы

1. Начало
2. Инициируем объекты для открытия других окон, доступных из главного окна, и объект загрузки доступных групп
3. Вызываем процедуру открытия главного окна
4. Если подан сигнал закрытия главного окна (н-р, нажата кнопка закрытия на верхней панели окна), переход на п.
5. Если нажата одна из трёх кнопок: «Просмотр посещаемости», «Изменение посещаемости» или «Список студентов», открыть соответствующее окно и скрыть главное. При отсутствии нажатий перейти к п. 4.

6. Удаляем все созданные во время работы программы временные файлы.

7. Конец

Рис. 2.1. Блок-схема алгоритма работы главного окна

2.5.2 Алгоритм работы окна просмотра посещаемости

1. Начало
2. Получаем с сервера список групп
3. Вносим полученные группы в выпадающий список (Combobox) для выбора группы
4. Ожидаем выбора группы из списка, а также указания даты и времени занятия.
5. Для выбранной группы запрашиваем список студентов и посещаемость по группе и сохраняем их в файл.
6. Переносим данные о студентах и их посещаемости из полученного файла в таблицу окна.
7. Ожидаем сигнала закрытия окна или выбора другой группы. В последнем случае снова переходим к п. 5.
8. Закрываем текущее окно и снова открываем главное.
9. Конец

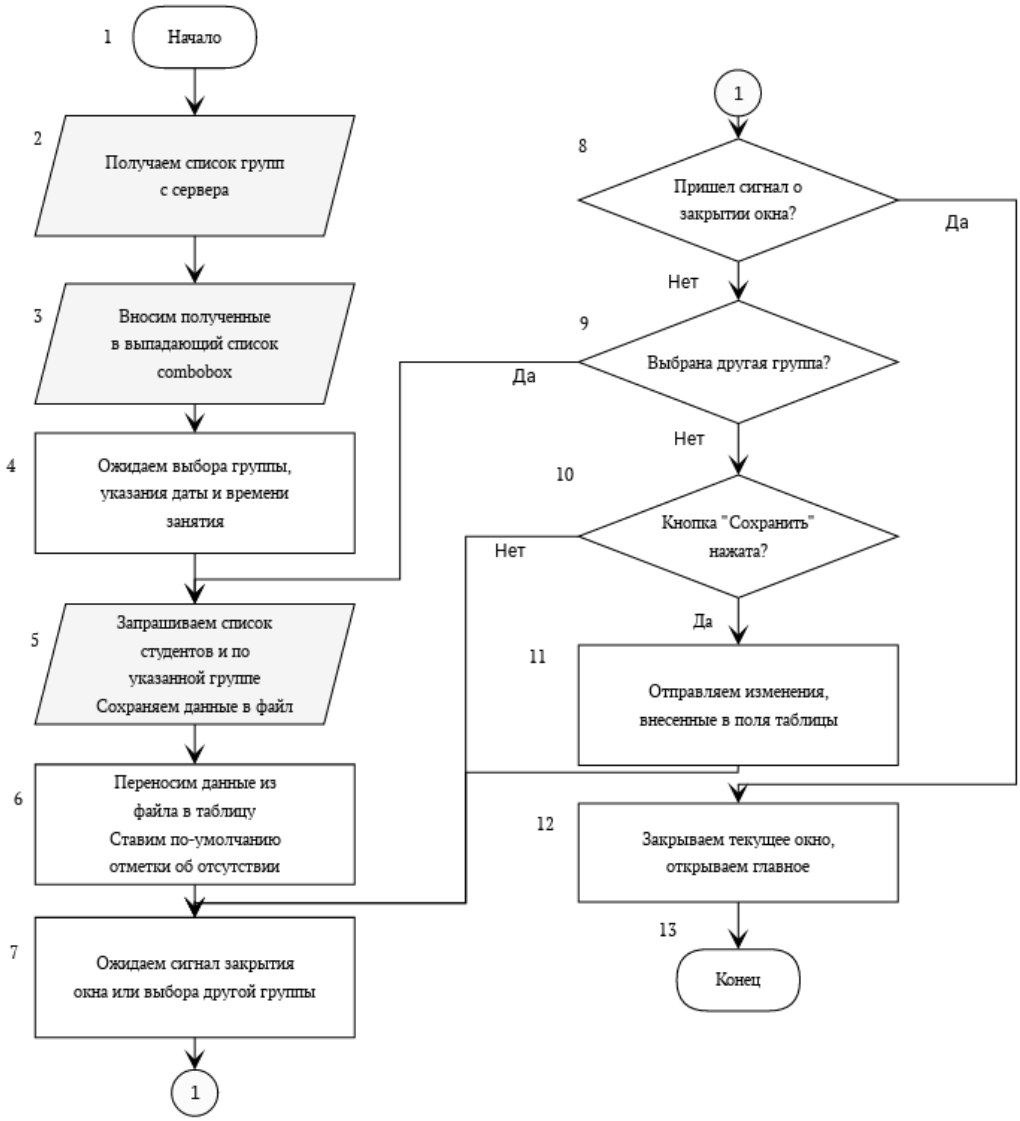

Рис. 2.2. Блок-схема алгоритма работы окна просмотра посещаемости

2.5.3 Алгоритм работы окна изменения посещаемости

1. Начало
2. Получаем с сервера список групп
3. Вносим полученные группы в выпадающий список (Combobox) для выбора группы
4. Ожидаем выбора группы из списка, а также указания даты и времени занятия.
5. Для выбранной группы запрашиваем список студентов и сохраняем его в файл.

6. Переносим данные о студентах из полученного файла в таблицу окна и ставим им по умолчанию отметки об отсутствии на занятии (можно изменить нажатием на галочку).

7. Ожидаем сигнала закрытия окна или выбора другой группы. В последнем случае снова переходим к п. 5.

8. Проверяем нажатие кнопки «Сохранить». При её нажатии происходит отправка изменений, внесённых в изменяемые поля таблицы, и переход к. 7.

9. Закрываем текущее окно и снова открываем главное.

10. Конец

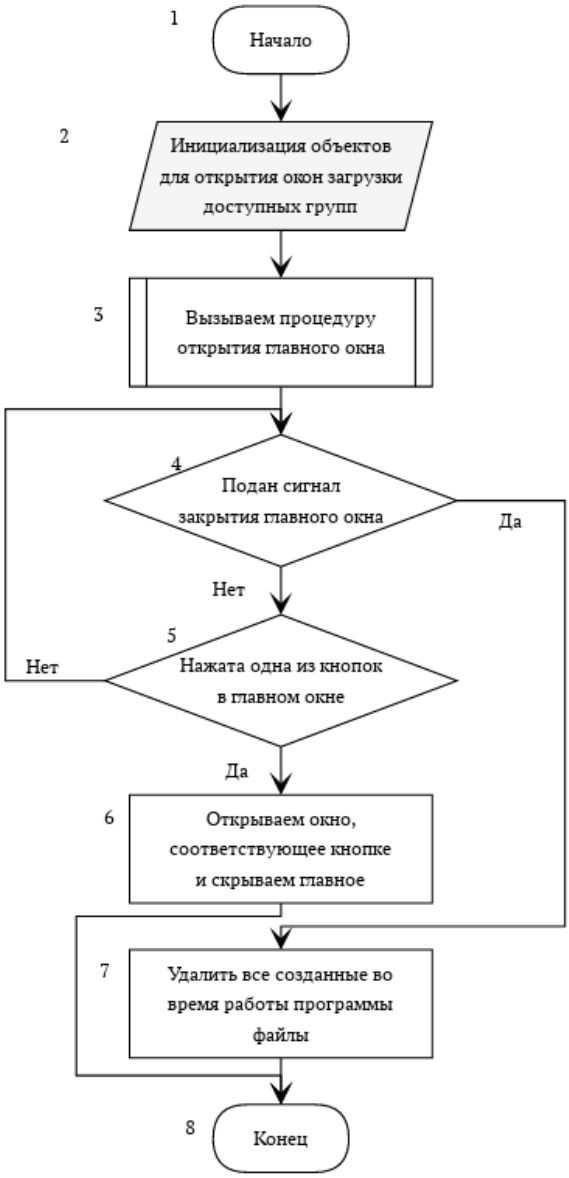

Рис. 2.3. Блок-схема алгоритма работы окна изменения посещаемости

2.5.4 Алгоритм работы окна просмотра и изменения списка студентов

1. Начало
2. Получаем с сервера список групп
3. Вносим полученные группы в выпадающий список (Combobox) для выбора группы
4. Ожидаем выбора группы из списка
5. Для выбранной группы запрашиваем список студентов и сохраняем его в файл.
6. Переносим данные о студентах из полученного файла в таблицу с изменяемыми полями.
7. Ожидаем сигнала закрытия окна или выбора другой группы. В первом случае переходим к п. 18, а в последнем случае снова переходим к п. 5.
8. Проверяем нажатие кнопки «Добавить». При её нажатии открывается форма добавления нового студента. Если кнопка не нажата, перейти к п. 13.
9. Ожидание заполнения формы и её закрытия
10. Если при заполнении формы была нажата кнопка «Отмена», то сразу происходит переход к п. 7
11. Отправка данных заполненной формы на сервер
12. Добавление данных о студенте в таблицу окна, переход к п. 15.
13. Проверяем нажатие кнопки «Изменить». При её нажатии все изменения, внесённые в таблицу, отправляются на сервер; переход к п. 15.
14. Проверяем нажатие кнопки «Удалить». При её нажатии на сервер отправляется запрос на удаление выбранного студента и его удаление из таблицы. Если кнопка не нажата, перейти к п. 7.
15. Проверка успешности отправленного запроса. Если сервер ответил на запрос ошибкой, происходит откат ранее произведённых изменений.
16. Вывод сообщения о статусе произведённой операции, переход к п. 7
17. Закрываем текущее окно и снова открываем главное
18. Конец

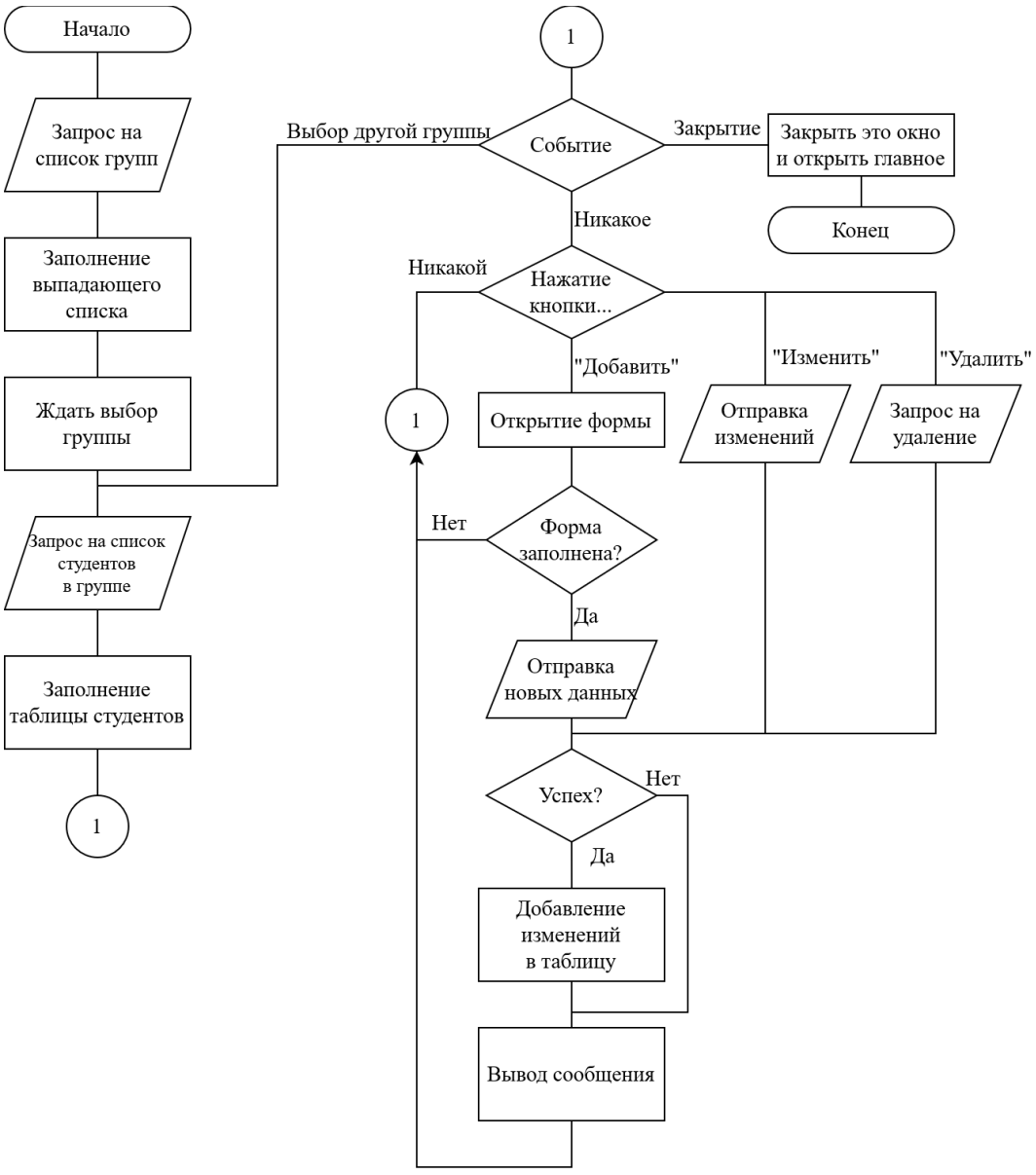

Рис. 2.4. Блок-схема алгоритма работы окна со списком студентов

2.6 Алгоритм работы сервера

Создание серверного приложения будет происходить на основе сетевого модуля Net из библиотеки Poco. Данный модуль обладает возможностями для создания асинхронных серверов для различных протоколов, включая HTTP. Асинхронная работа сервера подразумевает, что обработка запросов к нему будет происходить вне основного программного потока, который может быть занят другими задачами. Вместо этого сервер создаёт для каждого соединения

отдельный программный поток, который будет выполняться параллельно (или псевдо-параллельно, если реальных потоков процессора недостаточно) с основным потоком. Если на сервер придут запросы сразу от нескольких клиентов, все они по возможности будут также обработаны параллельно друг другу. Таким образом обработка запросов происходит с минимальными задержками со стороны сервера, благодаря рациональному распределению процессорного времени между потоками серверного приложения.

В рамках текущей задачи серверное приложение не обладает какими-либо задачами, кроме обработки запросов от клиентов, поэтому при отсутствии соединений оно простаивает в ожидании команды завершения работы (нажатия комбинации клавиш Ctrl+C со стороны пользователя или сигнала SIGTERM со стороны ОС). Поэтому, описанный ниже алгоритм следует рассматривать как алгоритм работы *потока обработки запроса*, а не приложения в целом. Условно этот алгоритм можно поделить на две части: арбитраж запроса и действия со стороны выбранного обработчика запроса. Первая часть алгоритма является общей для всех видов поступающих запросов, в то время как вторая часть зависит от содержания запроса, которое определяется на первом этапе. Суть второго этапа алгоритма может сильно отличаться для разных видов запросов, поэтому рациональнее рассматривать отдельно алгоритм арбитража и отдельные алгоритмы обработки запросов.

2.6.1 Процедура арбитража поступающего запроса

Процесс арбитража происходит следующим образом:

1. Начало
2. Поток получает содержание HTTP-запроса от клиента, включая его метод, адрес запрашиваемого ресурса и тело запроса.
3. Если запрос использует метод, отличный от GET, перейти к п. 9.
4. Если адрес начинается не с «/groups», перейти к п. 7.
5. Если адрес имеет продолжение (второй уровень вложенности),

передать название ресурса на этом уровне и значение URI-параметра «nameFormat» обработчику запросов на получение списка студентов; переход к п. 14.

6. Передать запрос обработчику запросов на получение списка групп, переход к п. 14.

7. Если адрес начинается с «/timerecords», передать запрос обработчику запросов на список посещаемости вместе с фильтрами данных в URI-запросах.

8. Безусловный переход к п. 13.

9. Если адрес начинается не с «/groups», перейти к п. 12.

10. Если адрес имеет продолжение (второй уровень вложенности), передать название ресурса обработчику запросов на изменение списка студентов; переход к п. 14.

11. Передать запрос обработчику запросов на изменение списка групп, переход к п. 14.

12. Если адрес начинается с «/timerecords», передать запрос обработчику запросов на список посещаемости вместе с фильтрами данных в URI-запросах; переход к п. 14.

13. Вернуть ответ с ошибкой 404 (ресурс не найден)

14. Конец

Данная схема арбитража позволяет отсеять запросы, которые отсылаются к заведомо несуществующим ресурсам (включая корневой адрес сервера «/», который не имеет связанного с ним ресурса) и в дальнейшем отдать запрос со всеми его параметрами одному из обработчиков, алгоритм работы которых будет описан ниже.

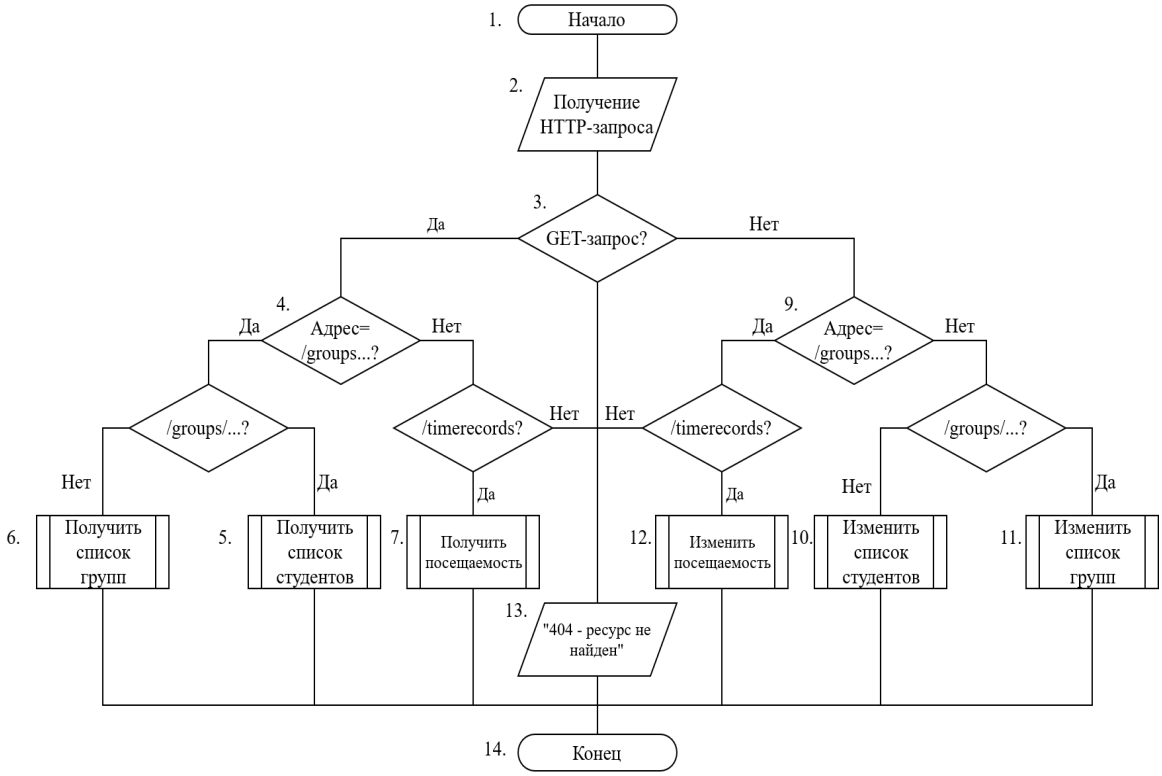

Рис. 2.5. Блок-схема арбитража запроса

2.6.2 Процедура получения списка групп

Отправка клиенту списка групп происходит следующим образом:

1. Начало
2. Получение HTTP-запроса от арбитра
3. Проверка актуальности содержания кэша по флагу устаревания. Если кэш не устарел и не пуст, перейти к п. 6.
4. Обращение к БД за списком номеров всех групп (поле «name»)
5. Очистка кэша от устаревшей информации
6. Запись извлечённых номеров в кэш
7. Запись текущего кэша в тело ответа в формате JSON
8. Конец

Данная процедура является одной из простейших в программе и демонстрируют реализацию одного из ключевых принципов REST – кэширования. Кэшируемые данные хранятся в оперативной памяти сервера,

поэтому скорость доступа к ним будет выше, чем у прямого доступа к данным из БД. Тем не менее, при первом обращении к данным скорость доступа к ним будет ниже, так как кэш сервера до этого был пуст и его необходимо заполнить.

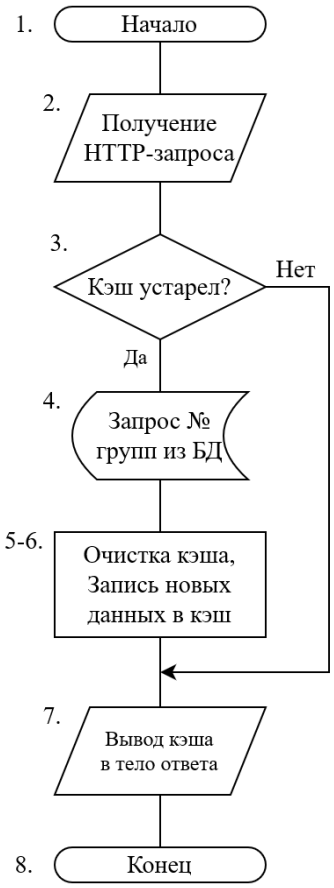

Рис. 2.6. Блок-схема процедуры получения списка групп

Следует обратить внимание, что флаг устаревания кэша, упомянутый в п. 3 алгоритма, всегда устанавливается извне, как правило, процедурой обновления данных в БД, что позволяет пользователю сразу увидеть внесённые им изменения. При запуске программы данный флаг устанавливается автоматически, чтобы сервер мог загрузить в кэш первичные данные при первом их запросе, поэтому для удобства реализации в понятие «устаревания данных» будет включаться также ситуация их отсутствия в кэше.

2.6.3 Процедура получения списка студентов

Процесс отправки клиенту списка студентов из конкретной группы

несильно отличается от процесса отправки списка групп. Основным отличием является необходимость кэширования уже нескольких наборов записей из БД. Для ограничения размера кэша и контроля истории обращений к нему используется структура данных, подобная очереди, но позволяющая обратиться к любую её элементу (а не только к началу или концу списка) и при необходимости переместить этот элемент в конец списка, тем самым отсрочив его выход по переполнению.

В целом алгоритм получения списка студентов и контроля содержимого кэша можно описать следующим образом:

1. Начало
2. Получение от арбитра содержания HTTP-запроса, номера группы и параметров выдачи имён студента (если они заданы)
3. Удаление из кэша давно не запрашиваемых и обновлённых записей
4. Проверка на содержание записи с нужной группой в кэше. Если она там уже есть, переместить её в конец истории запросов и перейти к пункту 7.
5. Обращение к БД за списком студентов по запрашиваемой группе.
6. Запись извлечённых данных в кэш этой группы
7. Чтение данных о составе группы из её кэша
8. Преобразование полей ФИО согласно заданному в запросе формату, если такой параметр был передан в URI
9. Запись информации о студентах в тело ответа в формате JSON
10. Конец

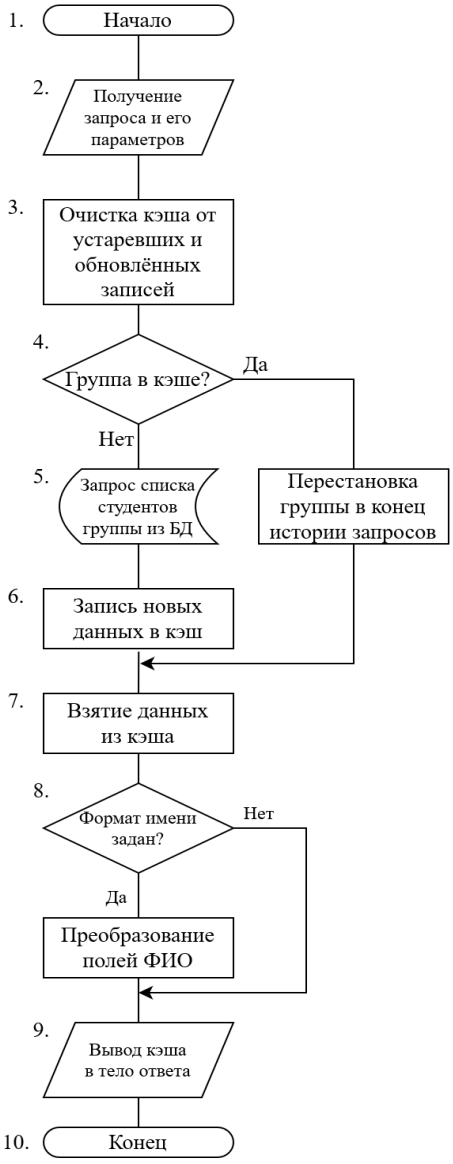

Рис. 2.8. Блок-схема процедуры получения списка студентов

Данный алгоритм применим для всех групп, включая виртуальную группу «all», которая содержит всех студентов в базе данных. На случай обращения к группе «all» или к нескольким группам с записью в один массив, помимо ФИО и ИД, в ответе также дублируется и номер группы каждого студента.

2.6.4 Процедура получения записей о посещаемости

Процедура получения и кэширования записей о посещаемости очень схожа с аналогичной процедурой у списка студентов, но имеет свою специфику.

В частности, существует сразу несколько критериев, по котором можно вести отбор записей для вывода: группа, дата и время занятия, статус посещаемости. Данные параметры могут быть заданы как и вместе, так и раздельно. Также может возникнуть необходимость в отборе целого диапазона значений, например, всех занятий с 10 до 25 мая 2022 года. В условиях такой вариативности проще задавать параметры поиска в виде фильтров. Контроль кэша также будет осуществляться по значениям фильтров у запроса. Фильтры отбора будут иметь следующий синтаксис, универсальный для любого из четырёх критериев отбора записей:

- фильтр без ограничений (по умолчанию) обозначается одиночным символом «*»;
- фильтр точного совпадения представлен строкой, которая полностью совпадает с искомым значением;
- фильтры сравнения обозначаются знаками неравенств («>», «<», «>=» или «<=») перед пороговым значением параметра поиска;
- фильтры диапазона обозначаются двумя пороговыми значениями, разделёнными дефисом. Из-за данного правила значения даты и времени следует подавать с разделителями, отличными от дефисов.
- Фильтры одной категории можно отъединять логическими операторами «ИЛИ» и «И» с помощью знаков «|» и «+» соответственно.

Алгоритм получения посещаемости студентов и контроля содержимого кэша можно описать следующим образом:

1. Начало
2. Получение от арбитра содержания HTTP-запроса и значений фильтров
3. Удаление из кэша давно не запрашиваемых и обновлённых записей

4. Проверка на содержание записи с такими же фильтрами в кэше. Если она там уже есть, переместить её в конец истории запросов и перейти к пункту 7.

5. Обращение к БД за записями о посещаемости, отобранными по заданным фильтрам.

6. Запись извлечённых данных в кэш по значениям фильтров

7. Чтение результатов отбора из кэша посещаемости

8. Запись информации о посещаемости каждого занятия каждым студентом в тело ответа в формате JSON

9. Конец

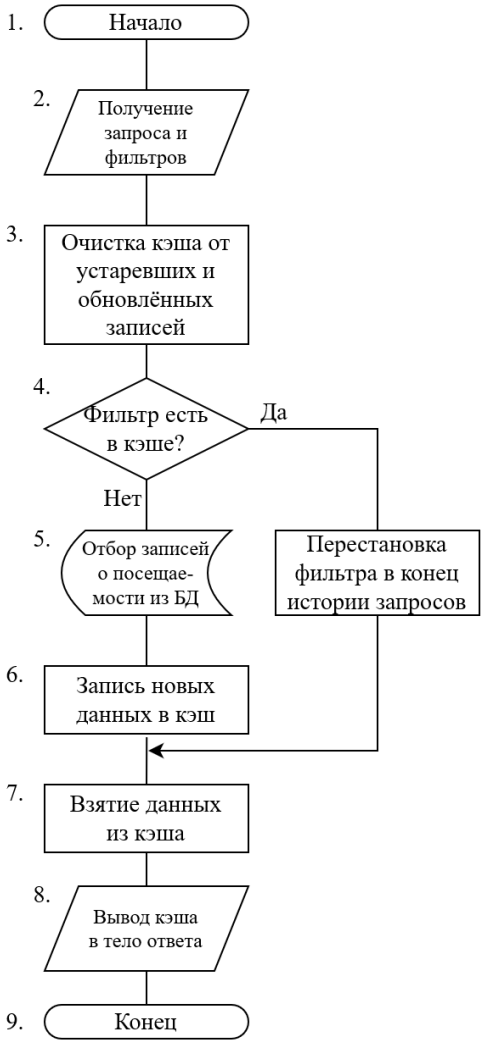

Рис. 2.8. Алгоритм получения записей о посещаемости

2.6.5 Общая процедура изменения таблиц базы данных

Как уже отмечалось в пункте 2.4, процедуры изменения списка групп,

списка студентов отдельной группы и записей посещений имеют схожий синтаксис обращения и отличаются лишь принимаемым набором данных. Это сделано для того, чтобы унифицировать алгоритм обработки запросов на изменение (добавление, удаление) записей в базе данных. Для любой из этих трёх процедур можно выделить три основных свойства принимаемого JSON-объекта:

- action – выполняемое действие. Данное поле является необязательным, но крайне желательным к указанию и может иметь одно из следующих значений:
  - alter – изменение значений в существующей записи по её ключевому значению – ИД студента (группы) или его связка с датой и временем занятия, в зависимости от изменяемой таблицы. Для таблицы групп также поддерживается выбор группы по её уникальному номенклатурному номеру, который фактически является её альтернативным ключом. Данная операция выбирается по умолчанию, если не указана другая из ниже перечисленных операций (идентификаторы операций «rename» и «move» также являются, по сути, синонимами идентификатора «alter»).
  - add – добавление новой записи в базу данных. Данная операция принимает полный набор значений полей новой записи в БД, за исключением ИД нового студента или группы, который присваивается автоматически за счёт использование типа данных serial (автоувеличиваемое целое) в PostgreSQL. Операция может пройти безуспешно, только если передаваемый набор данных оказался недостаточным, либо произошёл сбой в нумерации новых ИД на стороне СУБД (было замечено лишь один раз за весь период работы над решением).
  - remove – удаление записи по основному или альтернативному ключу. В зависимости от настроек СУБД, вместе в записью в одной таблице могут

удаляться связанные с ней записи в других таблицах (управляется значением параметра связанного ключа ON DELETE). Данное действие выбривается всегда для DELETE-запросов, даже если в теле указана другая операция.

- data – собственно передаваемые данные, представляющие с собой массив объектов. Каждый объект содержит необходимые для операции значения ключевых и (или) значащих полей изменяемой таблицы базы данных.

- defaults – значения по умолчанию, которыми заполняются недостающие свойства во всех объектах в массиве data. Использование данного объекта позволяет сократить размер запроса за счёт сведения неизменяемых для всех объектов data свойств в свойства по умолчанию. Данное поле является необязательным, если в его применении нет смысла (н-р для таблиц групп, где подобное сведение сделать невозможно).

Если не углубляться в особенности работы с каждой из таблиц, то для всех них можно составить единый алгоритм обработки запросов на изменение:

1. Начало
2. Получение JSON-объекта из тела HTTP-запроса
3. Взятие очередного объекта из массива data. Если массив пройден полностью, перейти к п. 9.
4. Дополнение объекта отсутствующими свойствами из объекта defaults.
5. Проверка значения свойства actions (или его значения по умолчанию) из корневого объекта. Если оно равно «add», то перейти к п. 6, а если равно «remove» – то к п. 7.
6. Отправка БД запроса на изменение значений на указанные в рассматриваемом объекте. Ключи изменяемых записей берутся оттуда же. Переход к п. 3.
7. Отправка БД запроса на создание новой записи со значениями, указанными в рассматриваемом объекте. Ключи из объекта берутся только для записей о посещаемости, для остальных записей ключ генерируется

автоматически. Переход к п. 3.

8. Отметить в кэше обработчика запроса на получения данных все изменённые записи как устаревшие и необходимые к удалению.

9. Отправка БД запроса на удаление записи с указанным в рассматриваемом объекте ключом, переход к п. 3.

10. Запись в ответе количества невыполненных запросов к БД и (или) возможных ошибок в их содержании.

11. Конец

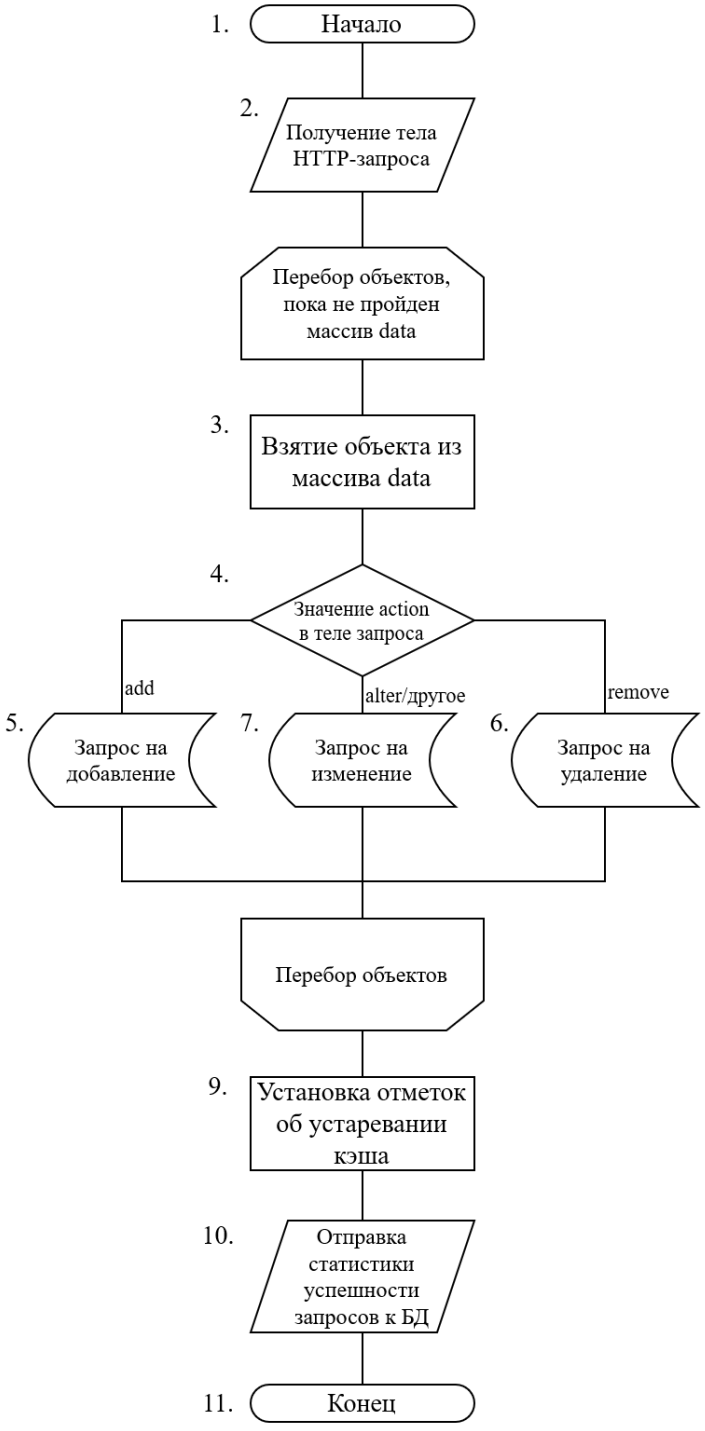

Рис. 2.5. Алгоритм обработки запросов на изменение

2.7 Реализация клиента и сервера

На основе описанных алгоритмов было реализовано две раздельных программы: клиентское приложение и серверное, которые запускаются на разных компьютерах и взаимодействуют друг с другом посредством

локальной сети и протокола HTTP. Клиенту для работы со сервером требуется знать его IP-адрес или доменное имя в сети, в то время как сервер получает все сведения о клиентах по мере их подключения к нему. Таким образом работа сервера не привязана к конкретной сети и конкретным клиентам, как это бывает при двухстороннем соединении «точка-точка». Система управления базой данных, с которой взаимодействуют сервер, запускается на той же машине, что и сам сервер, однако адрес сервера СУБД так же можно изменить в программе основного сервера. Исходный код программ приведён в приложении к практической работе.

Для реализации в сервере системы обработки запросов использовался модуль Net библиотеки Poco (далее – Poco::Net, согласно принятой в C++ нотации пространств имён). Архитектура всех серверных приложений на базе Poco::Net основана на объектно-ориентированной модели языка C++ и шаблоне проектирования «Фабрика»[4]. Роль первичного обработчика (арбитра) запросов берёт на себя *фабрика обработчиков запросов*, которая анализирует содержание запроса и создаёт соответствующий ему объект *обработчика запросов*. В дальнейшем именно выбранный обработчик запросов отвечает за обработку запроса и генерацию ответа на него. Если фабрика не смогла подобрать подходящий обработчик для отправленного запроса, то сервер автоматически выдаёт ошибку нереализованного функционала (501). Фабрика обработчиков, а также сами обработчики реализуются в виде классов, наследующих виртуальные (virtual) методы базовых классов из модуля Poco::Net и перезаписывающие их собственным функционалом. В случае с протоколом HTTP роль базового класса обработчика запросов играет класс Poco::Net::HTTPRequestHandler, а базового класса фабрики – Poco::Net::HTTPRequestHandlerFactory.

В работе сервера также участвуют и другие модули Poco: Poco::JSON, Poco::Dynamic, Poco::Data и его подмножество Poco::Data::PostrgreSQL.

Предназначение первого модуля – чтение и запись JSON-строк для приёма информации от клиента и её передачи обратно. Poco::Dynamic содержит реализацию динамической типизации для C++, которая используется другими модулями библиотеки, а Poco::Data – механизмы работы с базами данных. Использование некоторых функйи из модуля Poco::Data::PostgreSQL нужно для того, чтобы связать функции Poco::Data именно с базой данных на PostgreSQL (для других популярных СУБД существуют аналогичные модули, расширяющие и корректирующие функционал Poco::Data под их особенности). Каждому из модулей Poco соответствует свой набор заголовков, своё пространство имён и своя динамически подключаемая библиотека для платформ Linux и Windows. Для запуска сервера под AstraLinux с самостоятельно собранным набором библиотек требуется либо их установка по стандартному пути (/usr/lib) с помощью утилиты install, либо указание их расположения с помощью переменной окружения LD_LIBRARY_PATH.

3. ТЕСТИРОВАНИЕ ГОТОВОГО ПРОГРАММНОГО РЕШЕНИЯ

3.1 Запуск созданных приложений в ОС AstraLinux

Как уже ранее было сказано, обе программы должны работать под операционной системой AstraLinux. Запуск клиента из системы с графической оболочкой ничем не отличается от запуска других графических программ на платформе GNU/Linux. Главным условием является наличие на компьютере установленной библиотеки Qt 5.11 и разрешения на запуск файла у текущего пользователя, задаваемое путём выставления у файла программы бита исполняемости для текущего уровня доступа. Также между клиентом и сервером, очевидно, должно быть прямое сетевое соединение, не закрытое брандмауэром или параметрами сети. Если не выполняется хотя бы одно из этих условий, необходимо устранить проблему перед запуском программы, иначе она не запустится или не будет выполнять свои функции.

Для запуска сервера требуется произвести аналогичные приготовления, но вместо Qt в качестве основной зависимости выступают библиотека Poco последней версии и СУБД PostgreSQL с предварительно созданной и настроенной базой данных. Так же, в отличие от клиента, сервер работает в консольном режиме и не требует наличия графической оболочки операционной системы. Сервер необходимо запускать с правами суперпользователя (root), поскольку Poco::Net для работы HTTP-серверов использует так называемые сырые сокеты (англ. raw sockets), которые работают непосредственно с поступающими TCP-пакетам и требуют высокого уровня привилегий со стороны ОС. Запуск программы с правами суперпользователя возможен с помощью утилиты sudo, которая позволяет выполнять команды от имени другого пользователя.

3.2 Настройка локальной сети для тестирования ИС

Для того, чтобы клиент мог отправить данные серверу и наоборот требуется нахождение обоих компьютеров в одной сети. Также клиент должен знать, к какому именно адресу он должен обратиться для подключения сервера. Со стороны же сервера главным требованием является наличие открытого порта 80, через который по умолчанию идёт связь с сетевым узлом. Сетевые адреса клиентов сервер получает автоматически по мере их подключения.

Для обеспечения постоянного адреса сервера в сети, который позволяет не перенастраивать клиент при каждом запуске ИС, обычно применяют один из следующих подходов:

- Использование статического IP-адреса, который не меняется на протяжении всего срока службы сервера, что позволяет клиентам всегда подключаться по одному и тому же адресу.

- Использование в сети DNS и назначение серверу доменного имени. При обращении клиента к данному имени DNS преобразует его в актуальный IP-адрес сервера ИС.

Первый подход чаще используется в мелких сетях, поскольку в них проще всего наладить статическую IP-адресацию без риска конфликта нескольких устройств с одинаковым IP-адресом. Второй подход применяется уже в более крупных сетях, по типу Интернета, однако не исключает возможности применения первого способа для более быстрого доступа к серверу. Тем более, что большинство критических узлов сети, как правило, имеет всё же статический IP-адрес, что позволяет использовать оба способа для обращения к ним.

В тестовой конфигурации используется всего два устройства: клиентское и серверное, поэтому наиболее простым способом организации сети будет их прямое соединение друг с другом с помощью проводного интерфейса Ethernet.

В сетевых настройках обоих устройств необходимо указать их статические IP-адреса в созданной сети и маску адреса этой сети. Для локальных сетей малых размеров (до 254 устройств) с адресацией по IPv4 обычно применяют маску адреса 192.168.0.* или 192.168.1.*. В нашем случае клиент имеет адрес 192.168.0.11, а адрес сервер – 192.168.0.10.

3.3 Проверка взаимодействия между клиентом и сервером

При запуске сервера в консоль выводится информация о его состоянии в виде протокола его работы. В протоколе отражаются все запросы к серверу, проводимые в рамках их выполнения операции над базой данных и состояние кэша запросов на получение. Технически протокол разделён на общий поток вывода сообщений (std::cout) и поток вывода сообщений об ошибках (std::cerr). При необходимости содержимое общего протокола можно записать в файл с помощью оператора «>» командного языка Bash. Для записи протокола ошибок нужно использовать оператор «2>». Также сервер отправляет информацию о состояния своей работы клиенту: если он не может выполнить какой-то запрос, то клиент получает ответ со статусом, номер которого начинается на 4 или 5 (в зависимости от ошибки) или количество корректных, но не удачно выполненных запросов на изменение.

При запуске клиента открывается окно с выбором режима работы программы: «просмотр посещаемости», «изменение посещаемости» и «список студентов». При выборе любого из них клиент сразу же запрашивает список групп у сервера по адресу http://192.168.0.10/groups, что отражается в его протоколе как обращение по адресу /groups. Далее, в зависимости от режима работы и выполняемых действий, делают аналогичные обращения и другим ресурсам сервера. В момент, когда в один из этих ресурсов нужно внести изменения, клиент отправляет на адрес этого ресурса POST-запрос с запрашиваемыми изменениям, и соответствующие изменения так же отражаются в протоколе сервера.

Заключение

В результате практической работы была создана информационная система на базе клиент-серверной архитектуры, состоящая из графического клиента на базе фреймворка Qt 5.11, сервера на базе библиотеки Poco и базы данных на базе СУБД PostgreSQL. Студентами были освоены основные технологии и архитектурные принципы организации сетевого обмена между частями информационной системы, возможности использованных программных средств реализации ИС и навыки командной разработки сложных программных комплексов.

Список литературы

1. MDN Web Docs, Сообщения HTTP [эл. текст], 2021, [режим доступа — https://developer.mozilla.org/ru/docs/Web/HTTP/Messages]

2. Дуглас Крокфорд, Официальная страница JSON [эл. текст], 2022, [режим доступа — https://www.json.org/json-ru.html]

3: The Qt Company, Официальная документация Qt 5 (Qt Documentation), 2022

4: Poco Fundation, Официальный справочник функций (Reference Library), 2021

ПРИЛОЖЕНИЕ А

к отчёту по производственной эксплуатационной практике

«Исходный код серверного приложения»

1. Сценарий SQL для развёртывания базы данных в СУБД PostgreSQL

```
CREATE TABLE groups (
	id serial PRIMARY KEY,
	name varchar(10) NOT NULL UNIQUE); --Таблица групп
CREATE TABLE students (
	id serial PRIMARY KEY,
	fName varchar(32) NOT NULL,
	lName varchar(32) NOT NULL,
	mName varchar(32),
	groupId integer NOT NULL,
	FOREIGN KEY (groupId)
		REFERENCES groups(id)
		ON DELETE CASCADE
		ON UPDATE CASCADE); --Таблица студентов
CREATE TABLE timerecords (
	studId integer, date Date, time Time,
	presence NOT NULL DEFAULT false,
	PRIMARY KEY (studId, date, time),
	FOREIGN KEY (studId)
		REFERENCES students(id)
		ON DELETE CASCADE
		ON UPDATE CASCADE); --Таблица посещаемости
```

2. Основная программа серверного приложения (main.cpp)

```
#include <Poco/Util/ServerApplication.h> //Надстройка над main() для серверов
#include <Poco/Data/PostgreSQL/Connector.h>
#include <Poco/Net/HTTPServer.h>
#include <Poco/JSON/Parser.h>
#include <iostream>
#include "handlers/factory.hpp"

using namespace std;
using namespace Poco::Data;
using namespace Poco::Util;

class MyServer: public ServerApplication {
protected:
  int main(const vector<string>&) override {
```

```
    try {
      //Подключение к серверу PostgreSQL
      cout << "Connecting to the database...";
      PostgreSQL::Connector::registerConnector();
      string connStr = "host=127.0.0.1 user=postgres password=postgres
dbname=studentdb";
      Session psql(PostgreSQL::Connector::KEY, connStr);
      cout << "\tDONE!" << endl;
      //Инициализация HTTP-сервера
      cout << "Initializing the HTTP server...";
      auto factory = new ReqFactory(&psql);
      auto srvParams = new HTTPServerParams(); //Параметры сервера
      srvParams->setTimeout(Poco::Timespan(5, 0));
      HTTPServer srv(factory, ServerSocket(80), srvParams); //Сервер
      cout << "\tDONE!" << endl;
      //Запуск сервера
      cout << "Starting up the server...";
      srv.start();
      cout << "\tDONE!" << endl;
      //Ожидание сигнала завершения работы
      cout << "Press Ctrl+C to kill the server" << endl;
      waitForTerminationRequest();
      //Отключение сервера
      cout << "\nShutting down the server...";
      srv.stop();
      cout << "\tDONE!" << endl;
    }
    catch (Poco::Exception ex) {
      string text = ex.displayText();
      cerr << endl << text << endl;
      if (text == "Exception: Permission denied") {
        //Программа запущена с недостаточным количеством прав для запуска
сервера
        cerr << "Ensure you run the program under the root user (sudo !!)" << endl;
      }
      return 1;
    }
    return 0;
  }
};

Poco::JSON::Parser parser; //Парсер JSON (объявлен для других .cpp-файлов)
```

```
int main(int argc, char** argv) {
  MyServer app;
  return app.run(argc, argv);
}
```

3. Фабрика обработчиков HTTP-запросов

3.1. Заголовочный файл (handlers/factory.hpp)

```
#include <Poco/Net/HTTPRequestHandlerFactory.h>
#include <Poco/Data/Session.h>
#include <string>
#include <map>
#ifndef HANDLERS_FACTORY_HPP_INCLUDED
#define HANDLERS_FACTORY_HPP_INCLUDED

using namespace std;
using namespace Poco::Net;
using namespace Poco::Data;

class ReqFactory: public HTTPRequestHandlerFactory {
public:
  ReqFactory(Session* dbSession);
private:
  Session* dbSession = nullptr; //Общий сеанс связи с базой данных
  HTTPRequestHandler* createRequestHandler(const HTTPServerRequest& req)
override; //Метод-фабрикант обработчиков запросов
};
#endif
```

3.2. Реализация класса (handlers/factory.cpp)

```
#include "factory.hpp"
#include "groups.hpp"
#include "students.hpp"
#include "timerecords.hpp"
#include "agent.hpp"
#include "agents/timerecords.hpp"
#include "agents/groups.hpp"
#include "agents/students.hpp"
#include <sstream>
#include <vector>
#include <Poco/URI.h>

using namespace std;
using namespace Poco;
using namespace Poco::Net;
using namespace Poco::Data;
```

```
typedef std::vector<string> svec;
typedef pair<string,string> QueryParameter;
typedef map<string, string> smap;
typedef std::vector<QueryParameter> QueryParameters;

template <class K, class V>
inline void extractIfExists(map<K, V> m, K key, V& to) {
  /*Извлечь значение key из таблицы m в переменную to, если оно там
существует.
    Иначе, оставить переменной to её прежнее значение.*/
  try {
    to = m.at(key);
  } catch (std::out_of_range ex) {
    //Значение переменной to сохраняется
  }
}

template <class T>
bool valid(T v, std::vector<T> values) {
  // Проверить, соотвествует ли параметр одному из разрешённых значений
  auto it = find(values.begin(), values.end(), v);
  return it != values.end();
}

template <class Agent>
AgentRequestHandler* agentHandler(Session* dbSession) {
  //Двойной конструктор для обработчика и его агента
  return new AgentRequestHandler(new Agent(dbSession));
}

//Класс для вывода сообщений об ошибке
class ErrorRequestHandler: public HTTPRequestHandler {
  typedef HTTPResponse::HTTPStatus Status;
public:
  Status status;
  ErrorRequestHandler(Status status) {
    this->status = status;
  }
protected:
  void handleRequest(HTTPServerRequest& req, HTTPServerResponse& res) {
    res.setContentType("text/plain");
    res.setStatusAndReason(status);
    ostream& resBody = res.send();
```

```
    resBody << status << " - " << res.getReason() << endl;
    resBody.flush();
  }
};

//Конструктор класса
ReqFactory::ReqFactory(Session* dbSession){
  this->dbSession = dbSession;
}

//Метод-фабрикант (обработчик URI-адресов)
HTTPRequestHandler* ReqFactory::createRequestHandler(const
HTTPServerRequest& req) {
  //Разбор адреса
  URI uri(req.getURI());
  string method = req.getMethod();
  bool isGET = (method == HTTPRequest::HTTP_GET);
  string path = uri.getPath(); //Относительный адрес ресурса
  cout << "INFO: Accepted " << method << " request for " << path << endl;
  cout.flush();
  vector<string> pathSeg; //Сегменты адреса (по '/')
  uri.getPathSegments(pathSeg);
  size_t pathSegN = pathSeg.size();
  const QueryParameters params = uri.getQueryParameters(); //GET-параметры в
виде списка пар <ключ, значение>
  const smap paramMap(params.begin(), params.end()); //Таблица GET-параметров
(для более удобной обработки)
  cout << "INFO: The request has " << pathSegN << " path segments and " <<
paramMap.size() << " parameters" << endl;

  //Обработка адреса
  if (pathSegN == 0)
    goto push_404;
  if (pathSeg[0] == "groups"){
    switch (pathSegN){
    case 2:{
      string groupName = path.substr(8); //Название запрашиваемой группы
      if (isGET) {
          string nameFormat; //Формат имён студентов в ответе сервера
          extractIfExists(paramMap, (const string)"nameFormat", nameFormat);
          if (!valid(nameFormat, {"split", "initials", "full"}))
            nameFormat = StudentsRequestHandler::FORMAT_SPLIT; //Значение по
умолчанию
```

```
        return new StudentsRequestHandler(dbSession, groupName, nameFormat);
      } else {
        if (StudentsRequestHandler::groupNames.size() == 0)
          StudentsRequestHandler(dbSession).cacheUpdate(); //Обновляем кэш для
получения информации о группах
        AgentRequestHandler* ah = agentHandler<StudentsAgent>(dbSession);
        if (pathSeg[1] != StudentsRequestHandler::GROUPS_ALL)
          ah->getAgent()->defaults.set("group", pathSeg[1]);
        return ah;
      }
    }
    case 1:
      if (isGET)
        return new GroupsRequestHandler(dbSession);
      else
        return agentHandler<GroupsAgent>(dbSession);
    }
  } else if (pathSeg[0] == "timerecords") {
    if (isGET) {
      TimeRecordFilters filters; //Фильтры GET-запроса к журналу посещаемости
      //Заполнение фильтров из параметров запроса
      string* fields[] = {&filters.group, &filters.date, &filters.time, &filters.presence};
      static const string keys[] = {"group", "date", "time", "presence"};
      for (int i = 0; i < 4; i++)
      extractIfExists(paramMap, keys[i], *fields[i]);
    //Выполнение запроса
      return new TimeRecordsRequestHandler(dbSession, filters);
    } else {
      return agentHandler<TimeRecordAgent>(dbSession);
    }
  }
 push_404:
  cout << "WARNING: The request does not suit for any request handler. Returning
404..." << endl;
  return new ErrorRequestHandler(HTTPResponse::HTTP_NOT_FOUND);
}
```

4. Базовый обработчик HTTP-запросов на получение данных с кэшированием

4.1. Заголовочный файл (handlers/cached.hpp)

```
#include <Poco/Net/HTTPRequestHandler.h>
#include <Poco/Net/HTTPServerRequest.h>
#include <Poco/Net/HTTPServerResponse.h>
#include <Poco/Data/Session.h>
```

```
#include <Poco/JSON/Array.h>
#include <Poco/JSON/Object.h>
#include <ctime> //Для фиксации времени тех или иных событий в дочерних классах
#ifndef HANDLERS_CACHED_HPP_INCLUDED
#define HANDLERS_CACHED_HPP_INCLUDED

using namespace std;
using namespace Poco::Data;
using namespace Poco::Net;
using namespace Poco::JSON;

/*ПРИМЕЧАНИЕ: Все новые методы, кроме конструктора класса, а также наследуемый метод handleRequest() являются чисто виртуальными (pure virtual) и должны быть перезаписаны дочерними классами для их полной реализации.*/
class CachedRequestHandler: public HTTPRequestHandler {
public:
  CachedRequestHandler(Session* dbSession);
  static Session* dbSession; //Сеанс связи с БД
  virtual Array cacheArray() = 0; //Данные из кэша в формате JSON-массив
  virtual Object* getResponseObject() = 0; //Вернуть JSON-объект для ответа на запрос
protected:
  virtual bool cacheUpdate() = 0; //Обновить кэш (true, если успешно)
  static Object* responseObject(const Array& data, const Array& actions, const Object& requiredProperties, const string& filter); //Сгенерировать JSON-объект для ответа на запрос
  static void respondError(HTTPServerResponse& res, HTTPResponse::HTTPStatus status); //Выдать клиенту ошибку запроса
  virtual void handleRequest(HTTPServerRequest& req, HTTPServerResponse& res); //Обработка запросов (задан стандартный способ)
};
#endif
```

4.2. Реализация класса (handlers/cached.cpp)

```
#include "cached.hpp"

// Определние статичных (static) членов класса
Session* CachedRequestHandler::dbSession = nullptr;

// Определение конструктора класса
CachedRequestHandler::CachedRequestHandler(Session* dbSession) {
  this->dbSession = dbSession;
}

Object* CachedRequestHandler::responseObject(const Array& data, const Array&
actions, const Object& requiredProperties, const string& filter){
  Object* obj = new Object();
  obj->set("data", data); //Данные ответа
  obj->set("actions", actions); //Список доступных действий с данными
  obj->set("requiredProperties", requiredProperties); //Список аргументов,
требуемых для того или иного действия (или для всех сразу)
  obj->set("timeRecords", "/timerecords" + filter); //Ссылка на таблицу
посещаемости по данному запросу (с необходимыми фильтрами)
  return obj;
}
typedef HTTPResponse::HTTPStatus Status;
void CachedRequestHandler::respondError(HTTPServerResponse& res, Status
status) {
  res.setStatusAndReason(status);
  ostream& resBody = res.send();
  resBody << status << " - " << res.getReason() << endl;
  resBody.flush();
}
void CachedRequestHandler::handleRequest(HTTPServerRequest& req,
HTTPServerResponse& res) {
  try {
    cacheUpdate(); //Обновление кэша из базы данных
    //Отправка ответа в формате JSON
    res.setContentType("application/json");
    ostream& resBody = res.send();
    cout << "INFO: Obtaining the requested data from the cache..." << endl;
    Object* resObj = getResponseObject();
    resObj->stringify(resBody);
    delete resObj;
    resBody << endl; //Для "крассивого" вывода в консольных программах типа
curl
```

```
    resBody.flush();
    cout << "INFO: Request done!" << endl;
  } catch (Poco::Exception ex) {
    //Ошибка формирования ответа на запрос (сервер отсылает ошибку 500)
    respondError(res, HTTPResponse::HTTP_INTERNAL_SERVER_ERROR);
    cerr << "Request handling error!" << endl;
    cerr << ex.displayText() << endl;
  }
}
```

5. Обработчик запросов на получение списка групп

5.1. Заголовочный файл (handlers/groups.hpp)

```
#include "cached.hpp"
#include <vector>
#ifndef HANDLERS_GROUPS_HPP_INCLUDED
#define HANDLERS_GROUPS_HPP_INCLUDED

struct Group {
  int id; //ИД группы в базе данных (БД)
  string name; //Название (номер) группы
};

class GroupsRequestHandler: public CachedRequestHandler {
public:
  static bool cacheOutdated; //Данные в кэша устарели или отсутствуют
  using CachedRequestHandler::CachedRequestHandler; // Использовать
родительский конструктор
  bool cacheUpdate() override; //Обновить кэш списка вручную (true, если
успешно)
  Array cacheArray() override; //Список групп из кэша в формате JSON
  Object* getResponseObject() override; //Ответ на запрос списка групп
private:
  static std::vector<Group> cache; //Кэш списка групп
};
#endif
```

5.2. *Реализация класса (handlers/groups.cpp)*

```
#include "groups.hpp"
#include <Poco/JSON/Parser.h>
#include <sstream>

using namespace Poco::Data::Keywords;
using namespace Poco::JSON;
extern Parser parser;
```

```
//Объявление статичных (static) членов класса
std::vector<Group> GroupsRequestHandler::cache;
bool GroupsRequestHandler::cacheOutdated = true;

//Реализация виртуальных методов родительских классов
Array GroupsRequestHandler::cacheArray() {
  Array arr; //Массив JSON для номеров групп
  for (Group g: cache)
    arr.add(g.name);
  return arr;
}

bool GroupsRequestHandler::cacheUpdate() {
  if (!cacheOutdated)
    return true;
  try {
    if (dbSession == nullptr)
        throw ConnectionFailedException("DB session is not specified");
    Statement query(*dbSession); //Объекта запроса базы данных
    Group g; //Промежуточные представление данных о группе
    /*Запись и выполнение запроса выборки (SELECT) с указанием полей
      для хранения данных и шага обхода запрашиваемой таблицы групп
(groups)*/
    query << "SELECT * FROM groups ORDER BY name;", into(g.id), into(g.name),
range(0, 1);
    while (!query.done()) {
      query.execute();
      cache.push_back(g);
    }
    //Уведомление об успешном обновлении кэша
    cacheOutdated = false;
    time_t cachedTime = time(nullptr);
    cout << "INFO: Successfully updated group list cache on " <<
ctime(&cachedTime);
  } catch (Poco::Exception ex) {
    //Предупреждение об ошибке обновления кэша (сервер хранит старые
данные)
    cerr << ex.displayText() << endl;
    cout << "WARNING: Last cache update failed! The response data may be
outdated." << endl;
    return false;
  }
  return true;
```

```
}
Object* GroupsRequestHandler::getResponseObject() {
  static Array actions;
  static Object required;
  if (actions.size() == 0) {
    for (auto action: {"add", "rename", "remove"}) {
      actions.add(action);
      Array props;
      props.add("name");
      if (action == "rename")
          props.add("newName");
      required.set(action, props);
    }
  }
  return responseObject(cacheArray(), actions, required, "");
}
```

6. Обработчик запросов на получение списка студентов

6.1. Заголовочный файл (handlers/students.hpp)

```
#include "cached.hpp"
#include <vector>
#include <map>
#include <list>
#include <Poco/JSON/Object.h>
#ifndef HANDLERS_STUDENTS_HPP_INCLUDED
#define HANDLERS_STUDENTS_HPP_INCLUDED

using namespace Poco::JSON;

struct Student { //Информация о студенте
  int id; //ИД студента в базе данных (БД)
  string fName, lName, mName; //ФИО студента (раздельно)
  string fullName(bool initials); //ФИО студента (слитно)
  size_t groupId; //ИД группы студента
  Object JSONObject(string nameFormat); //Перевод в формат JSON
};

typedef std::vector<Student> Students;
typedef map<string, Students> StudentCache;
typedef list<string> StudentCacheQueue;

class StudentsRequestHandler: public CachedRequestHandler {
  typedef const char* strconst; //Тип для строковых констант
  friend Student; //Предоставить структуре Student доступ ко всем членам класса
```

```
  friend class StudentsAgent; //Предоставить агенту доступ ко всем членам класса
  friend class ReqFactory;
public:
  StudentsRequestHandler(Session* dbSession, string group = GROUPS_ALL, string
nameFormat = FORMAT_SPLIT);
  static bool cacheOutdated; //Данные в кэше устарели или отсутствуют
  static bool groupExists(string); //Проверка на существование группы (из кэша
групп)
  bool cacheUpdate() override; //Обновить кэш списка вручную (true, если
успешно)
  Array cacheArray() override; //Данные кэша в формате JSON
  Array cacheArray(string group); //Данные по конкретной группе в формате JSON
  static Students getStudentsFromDB(string group = GROUPS_ALL); //Вернуть
список студентов из БД
  static Students getStudentsFromDB(int groupId);
  Object* getResponseObject() override; //Ответ на запрос списка групп
  // Параметры обработки запроса
  string group, nameFormat;
  // Форматы имён и прочие константы
  static constexpr strconst GROUPS_ALL = "all"; //Ключ для вывода студентов из
всех групп
  static constexpr int GROUPS_NULL = -1; //ID для отсутствующей группы
  static constexpr int GROUPS_ALL_ID = -2; //ID для ключа GROUPS_ALL
  static constexpr strconst FORMAT_SPLIT = "split"; //Выводить значения ФИО
раздельно (fName, lName, mName)
  static constexpr strconst FORMAT_FULL = "full"; //Выводить ФИО одной
строкой (name) полностью
  static constexpr strconst FORMAT_INITIALS = "initials"; //Выводить ФИО одной
строкой (name) с инициалами
private:
  static StudentCache cache; //Кэш списка студентов в каждой группе (ключ "all"
хранит всех студентов в БД)
  static StudentCacheQueue cacheOrder; //Очередь с порядком добавления
элементов в кэш
  static const int CACHE_SIZE = 10; //Максимальное количество записей, которое
может храниться в кэше
  static vector<string> groupNames; //Названия всех групп (не только из кэша)
  static vector<int> groupIds; //ИД всех групп (параллельно их названиям в
groupNames)
  static int groupIdByName(string name); //ИД группы по её названию
  static const string groupNameById(int id); //Название группы по её ИД
  void handleRequest(HTTPServerRequest& req, HTTPServerResponse& res)
override;
```

```
};
#endif
```

6.2. Реализация класса (handlers/students.cpp)

```
#include "students.hpp"
#include "agents/students.hpp"
#include <sstream>
#include <codecvt>
#include <Poco/JSON/Array.h>
#include <Poco/JSON/Parser.h>
#include <Poco/Data/RecordSet.h>
#include <Poco/Dynamic/Var.h>

using namespace Poco::Data::Keywords;
using Poco::Dynamic::Var;
extern Parser parser;

//Методы структуры Student
string Student::fullName(bool initials) {
  static wstring_convert<codecvt_utf8_utf16<wchar_t>> converter; //Переводчик
UTF-8 -> UTF-16
  ostringstream name; //Строковый поток полного имени
  name << lName << ' ';
  if (initials) {
    wstring initials[2]; //Для корректной обработки кириллицы используются
"широкие" строки (wstring)
    for (int i = 0; i < 2; i++){
      wstring wn = converter.from_bytes(i == 0 ? fName: mName);
      if (wn.substr(0, 2) == L"Дж")
          initials[i] = L"Дж"; //Исключение из правил составления инициалов
      else initials[i] += wn[0];
      name << converter.to_bytes(initials[i]) << '.';
    }
  } else {
    name << fName << ' ' << mName;
  }
  return name.str();
}

Object Student::JSONObject(string nameFormat) {
  Object obj;
  //Заполнение свойств объекта
  obj.set("id", id);
  obj.set("group", StudentsRequestHandler::groupNameById(groupId));
```

```
  if (nameFormat == StudentsRequestHandler::FORMAT_SPLIT) {
    obj.set("fName", fName);
    obj.set("lName", lName);
    obj.set("mName", mName);
  } else {
    bool init = (nameFormat == StudentsRequestHandler::FORMAT_INITIALS);
    obj.set("name", fullName(init));
  }
  return obj;
}

//Объявление статичных (static) членов класса
bool StudentsRequestHandler::cacheOutdated = true;
StudentCache StudentsRequestHandler::cache;
StudentCacheQueue StudentsRequestHandler::cacheOrder;
std::vector<string> StudentsRequestHandler::groupNames;
std::vector<int> StudentsRequestHandler::groupIds;

//Объявление конструктора
StudentsRequestHandler::StudentsRequestHandler(Session* dbSession, string group,
string nameFormat): CachedRequestHandler(dbSession) {
  this->group = group;
  this->nameFormat = nameFormat;
};

//Собственные методы класса
inline bool StudentsRequestHandler::groupExists(string group) {
  if (group == GROUPS_ALL) //Проверка на техническую группу "all"
    return true;
  return find(groupNames.begin(), groupNames.end(), group) != groupNames.end();
}

int StudentsRequestHandler::groupIdByName(string name) {
  if (name == GROUPS_ALL)
    return GROUPS_ALL_ID;
  for (size_t i = 0; i < groupNames.size(); i++) {
    if (groupNames[i] == name)
      return groupIds[i];
  }
  return GROUPS_NULL; //Группа не найдена
}

const string StudentsRequestHandler::groupNameById(int id) {
```

```
  if (id == GROUPS_ALL_ID)
    return GROUPS_ALL;
  for (size_t i = 0; i < groupIds.size(); i++) {
    if (groupIds[i] == id)
      return groupNames[i];
  }
  return ""; //Группа не найдена
}

Array StudentsRequestHandler::cacheArray(string group){
  Array arr; //Массив JSON для объектов студентов
  Students studs; //Список студентов
  try {
    studs = cache[group];
  } catch (std::exception ex) {
    //Обновить кэш и повторно извлечь список студентов из него
    swap(this->group, group); //Временная замена
    cacheUpdate();
    swap(this->group, group);
    studs = cache[group];
  }
  for (Student stud: studs) {
    //cout << "Adding student ID " << stud.id << " from " << group << endl;
    arr.add(stud.JSONObject(nameFormat));
  }
  return arr;
}

Students StudentsRequestHandler::getStudentsFromDB(int groupId) {
  if (dbSession == nullptr)
    throw ConnectionFailedException("DB session is not specified");
  Statement statement(*dbSession);
  //Текст запроса
  string query = "SELECT * FROM students";
  if (groupId == GROUPS_ALL_ID)
    query += ";";
  else
    query += " WHERE groupId = " + to_string(groupId) + ";";
  cout << groupId << endl;
  Students studs; //Список студентов
  statement << query, now;
  RecordSet rs(statement);
  for (Row row: rs) {
```

```
    Student stud = {row.get(0), row.get(1), //id, fName
                    row.get(2), row.get(3), row.get(4)}; //lName, mName, groupId
    studs.push_back(stud);
  }
  return studs;
}

Students StudentsRequestHandler::getStudentsFromDB(string group) {
  return getStudentsFromDB(groupIdByName(group));
}

//Реализация виртутальных методов родительских классов
Array StudentsRequestHandler::cacheArray() {
  Array arr; //Массив итоговых данных
  for (string group: cacheOrder) {
    //Парсируем информацию по конкретным группам
    Array studs = cacheArray(group);
    Object gobj;
    gobj.set("group", group);
    gobj.set("students", studs);
    arr.add(gobj);
  }
  return arr;
}

bool StudentsRequestHandler::cacheUpdate() {
  try {
    if (dbSession == nullptr)
        throw ConnectionFailedException("DB session is not specified");
    if (cacheOutdated || groupNames.size() == 0) {
      //Очистка кэша от устаревших данных
      groupNames.clear();
      groupIds.clear();
      cacheOrder.clear();
      cache.clear();
      //Получаем список всех групп
      Statement query(*dbSession);
      query << "SELECT * FROM groups;", into(groupIds), into(groupNames), now;
      time_t cachedTime = time(nullptr);
      cout << "INFO: Successfully updated group list on " << ctime(&cachedTime);
      cacheOutdated = false;
    }
    auto order = find(cacheOrder.begin(), cacheOrder.end(), group);
```

```
    if (order != cacheOrder.end()) {
      //Группа уже есть в кэше, обновить её положение в очереди
      cacheOrder.erase(order);
      cacheOrder.push_back(group);
    }
    else if (groupExists(group)) {
      //Если группа вообще существует в списке
      if (cacheOrder.size() >= CACHE_SIZE) {
          //Очистка кэша от данных, к которым давно не обращались
          cache.erase(cacheOrder.front());
          cacheOrder.pop_front();
      }
      //Добавление списка группы в кэш
      Students dbData = getStudentsFromDB(group);
      cache[group] = dbData;
      if (cacheOrder.size() == CACHE_SIZE - 1)
          cout << "WARNING: The student data cache is going to be overflowed. Old
data will be erased until they are required again." << endl;
      cacheOrder.push_back(group);
      time_t cachedTime = time(nullptr);
      cout << "INFO: Cached student data for group " << group << " on " <<
ctime(&cachedTime);
    }
  } catch (Poco::Exception ex) {
    cerr << "Student cache update error!" << endl;
    cerr << ex.displayText() << endl;
    return false;
  }
  return true;
}

Object* StudentsRequestHandler::getResponseObject() {
  cout << "Formating the response..." << endl;
  static Array actions;
  static Object required;
  if (actions.size() == 0) {
    StudentsAgent sa(dbSession); //Для доступа к вирутальным методам
    for (auto action: sa.PURPOSE_ALL) {
      actions.add(action);
      Array props;
      for (auto prop: sa.requiredProperties(action))
          props.add(prop);
      required.set(action, props);
```

```
    }
  }
  try {
    Object* obj = responseObject(cacheArray(group), actions, required, "?group=" +
group);
    cout << "Checking response...";
    //Извлекаем массив ч/з парсинг его JSON-представления (из-за неопознанной
ошибки при извлечении массива напрямую)
    Var data = obj->get("data");
    data = parser.parse(data.toString());
    obj->set("data", data);
    //Возврат объекта
    return obj;
  } catch (std::exception ex) {
    cerr << ex.what() << endl;
  }
  return new Object();
}

void StudentsRequestHandler::handleRequest(HTTPServerRequest& req,
HTTPServerResponse& res) {
  cacheUpdate(); //Обновляем кэш из базы данных
  //Проверка на существование группы
  if (!groupExists(group)) {
    //Группа не найдена, отправка пустого сообщения с кодом 404
    cout << "WARNING: Attempt to require data for a missing group " << group << ".
";
    cout << "Returning 404..." << endl;
    respondError(res, HTTPResponse::HTTP_NOT_FOUND);
  } else {
    //Отправка ответа в формате JSON
    CachedRequestHandler::handleRequest(req, res);
  }
}
```

7. Обработчик запросов на получение записей о посещаемости

7.1. Заголовочный файл (handlers/timerecords.hpp)

```
#include "cached.hpp"
#include <vector>
#include <map>
#include <list>
#include <Poco/JSON/Object.h>
#ifndef HANDLERS_TIMERECORDS_HPP_INCLUDED
#define HANDLERS_TIMERECORDS_HPP_INCLUDED
```

```
using namespace Poco::JSON;
using Poco::DateTime;

struct TimeRecord { //Информация о посещаемости студента
  int id; //ИД студента в базе данных (БД)
  string date; //Дата занятия
  string time; //Время занятия
  bool present = false; //Отметка о присутствии студента в указанное время
  Object JSONObject(); //Перевод в JSON
};

struct TimeRecordFilters {
  //Параметры фильтрации результатов GET-запроса
  string group = GROUP_ANY;
  string date = DATE_ANY;
  string time = TIME_ANY;
  string presence = PRESENCE_ANY;
  // Типы фильтров
  enum FilterType {FILTER_GROUP, FILTER_DATE, FILTER_TIME,
FILTER_PRESENCE};
  // Константы фильтров
  typedef const char* strconst; //Тип для строковых констант
  static constexpr strconst GROUP_ANY = "*"; //Не фильтровать записи по группе
  static constexpr strconst DATE_ANY = "*"; //Не фильтровать записи по дате
  static constexpr strconst TIME_ANY = "*"; //Не фильтровать записи по времени
  static constexpr strconst PRESENCE_ANY = "*"; //Не фильтровать записи по
присутствию/отсутствию студента
  static constexpr strconst PRESENCE_YES = "true"; //Фильтр записей по
присутствию на занятии
  static constexpr strconst PRESENCE_NO = "false"; //Фильтр записей по
отсутствию на занятии
  //Операторы и преобразования
  inline const string toString(bool skipWildcards = true) const; //Запись параметров
фильтрации в одну строку
  const bool operator<(const TimeRecordFilters other) const; //Для сортировки в
std::map();
  const bool operator==(const TimeRecordFilters other) const; //Проверка на
равенство двух наборов фильтров
  //Формирование SQL-запросов и их частей
  static string SQLCondition(string filter, FilterType type); //Перевод фильтра в
предикат SQL
  static inline string uniteSQLConditions(string filter, char delim, string op,
```

```
FilterType); //Объединение предикатов
  string SQLQuery(); //Создание SQL-запроса для таблицы "timerecords" с учётом
всех фильтров
};

typedef std::vector<TimeRecord> TimeRecords;
typedef map<TimeRecordFilters, TimeRecords> TimeCache;
typedef list<TimeRecordFilters> TimeCacheQueue;

class TimeRecordsRequestHandler: public CachedRequestHandler {
public:
  TimeRecordsRequestHandler(Session* dbSession, TimeRecordFilters filters);
  static bool cacheOutdated; //Данные в кэше устарели или отсутствуют
  bool cacheUpdate() override; //Обновить кэш списка вручную (true, если
успешно)
  Array cacheArray() override; //Данные кэша в формате JSON
  Array cacheArray(TimeRecordFilters filter); //Данные по фильтру в формате
JSON
  Object* getResponseObject() override; //Ответ на запрос списка групп
  //Параметры обработки запроса
  TimeRecordFilters filters;

private:
  static TimeCache cache; //Кэш записей по времени
  static TimeCacheQueue cacheOrder; //Очередь с порядком добавления элементов
в кэш
  static const int CACHE_SIZE = 30; //Максимальное количество записей, которое
может храниться в кэше
};
#endif
```

7.2. Реализация класса (handlers/timerecords.cpp)

```
#include "timerecords.hpp"
#include "agents/timerecords.hpp"
#include <sstream>
#include <Poco/JSON/Parser.h>
#include <Poco/Dynamic/Var.h>

//Вспомгательные функции
inline bool startsWith(string s, string begin) {
  //Проверка на совпадение начала строки s со строкой begin
  s.resize(begin.size()); //Выравниваем строки по длине
  return s == begin;
}
```

```
//Методы структур
Object TimeRecord::JSONObject() {
  Object obj;
  obj.set("id", id);
  obj.set("date", date);
  obj.set("time", time);
  obj.set("present", present);
  return obj;
}

typedef TimeRecordFilters::FilterType FilterType;
inline string TimeRecordFilters::uniteSQLConditions(string filter, char delim, string
op, FilterType type) {
  istringstream ss(filter);
  string united = "("; //Объединённый предикат
  string sub; //Временная переменная для подфильтра
  while(getline(ss, sub, delim)) {
    if (united != "(") //Перед первым предикатом op не ставится
      united += op;
    united += SQLCondition(sub, type);
  }
  return united + ")";
}

string TimeRecordFilters::SQLCondition(string filter, FilterType type) {
  string param; //Сравниваемый параметр
  switch(type) {
  case FILTER_GROUP:
    param = "groups.name";
    break;
  case FILTER_DATE:
    param = "date";
    break;
  case FILTER_TIME:
    param = "time";
    break;
  case FILTER_PRESENCE:
    param = "presence";
    break;
  }
  //Предикаты "больше-меньше"
  for (string pred: {"<=", ">=", "<", ">"}) {
```

```
    if (startsWith(filter, pred)) {
      filter = filter.substr(pred.size());
      if (type != FILTER_PRESENCE)
          filter = "\'" + filter + "\'";
      return param + pred + filter;
    }
  }
  //Предикат диапазона
  int betweenSep = filter.find('-');
  if (betweenSep > -1) {
    string rrange = filter.substr(betweenSep + 1);
    string lrange = filter.substr(0, betweenSep);
    if (type != FILTER_PRESENCE) {
          lrange = "\'" + lrange + "\'";
          rrange = "\'" + rrange + "\'";
    }
    return param + " BETWEEN " + lrange + " AND " + rrange;
  }
  //Предикаты, объединённые дизъюнкцией
  if (filter.find('|') != string::npos)
    return uniteSQLConditions(filter, '|', " OR ", type);
  //Предикаты, объединённые конъюнкцией
  if (filter.find('+') != string::npos)
    return uniteSQLConditions(filter, '+', " AND ", type);
  //Предикат одинарного равенства
  if (type != FILTER_PRESENCE)
    filter = "\'" + filter + "\'";
  return param + "=" + filter;
}

string TimeRecordFilters::SQLQuery() {
  string query = "SELECT studId, date, time, presence FROM students, groups,
timerecords";
  query += " WHERE students.id = studId AND groups.id = groupId";
  if (group != GROUP_ANY)
    query += " AND " + SQLCondition(group, FILTER_GROUP);
  if (date != DATE_ANY)
    query += " AND " + SQLCondition(date, FILTER_DATE);
  if (time != TIME_ANY)
    query += " AND " + SQLCondition(time, FILTER_TIME);
  if (presence != PRESENCE_ANY)
    query += " AND " + SQLCondition(presence, FILTER_PRESENCE);
  query += ";";
```

```
  return query;
}

inline const string TimeRecordFilters::toString(bool skipWildcards) const {
  if (!skipWildcards) //Если разрешён вывод фильтров типа "*" (любое значение)
    return "group=" + group + "&date=" + date + "&time=" + time + "&presence=" +
presence;
  string ret; //Возвращаемые данные
  static const string params[] = {"group", "date", "time", "presence"};
  const string* fields[] = {&group, &date, &time, &presence};
  for (int i = 0; i < 4; i++){
    const string field = *fields[i];
    if (field != "*") {
      if (!ret.empty())
          ret += "&";
      ret += params[i] + "=" + field;
    }
  }
  return ret;
}

const bool TimeRecordFilters::operator<(const TimeRecordFilters other) const {
  return this->toString() < other.toString();
}

const bool TimeRecordFilters::operator==(const TimeRecordFilters other) const {
  return (this->date == other.date) && (this->time == other.time) &&
    (this->group == other.group) && (this->presence == other.presence);
}

//Объявление статичных (static) членов класса
TimeCache TimeRecordsRequestHandler::cache;
TimeCacheQueue TimeRecordsRequestHandler::cacheOrder;
bool TimeRecordsRequestHandler::cacheOutdated = true;

//Конструктор класса
TimeRecordsRequestHandler::TimeRecordsRequestHandler(Session* dbSession,
TimeRecordFilters filters): CachedRequestHandler(dbSession) {
  this->filters = filters;
};

//Собственные методы класса
Array TimeRecordsRequestHandler::cacheArray(TimeRecordFilters filter) {
```

```
  Array arr; //Массив для объектов JSON
  TimeRecords tr; //Записи о посещаемости
  try {
    tr = cache[filter];
  } catch (std::exception ex) {
    //Обновить кэш и повторно извлечь записи из него
    swap(this->filters, filter); //Временная замена
    cacheUpdate();
    swap(this->filters, filter);
    tr = cache[filter];
  }
  for (TimeRecord t: tr)
    arr.add(t.JSONObject());
  //Возврат JSON-строки
  return arr;
}

//Реализация виртуальных методов родительских классов
Array TimeRecordsRequestHandler::cacheArray() {
  using Poco::Dynamic::Var; //Динамический тип переменных
  Array arr; //Массив для объектов JSON
  for (auto cached: cache) {
    TimeRecordFilters f = cached.first;
    Object obj;
    obj.set("filters", f.toString());
    obj.set("data", cacheArray(f));
    arr.add(obj);
  }
  return arr;
}
bool TimeRecordsRequestHandler::cacheUpdate() {
  using namespace Poco::Data::Keywords;
  try {
    if (dbSession == nullptr)
        throw ConnectionFailedException("DB session is not specified");
    if (cacheOutdated) {
      //Очистка кэша от устаревших данных
      cache.clear();//TODO: сделать выборочную очистку
      cacheOrder.clear();
      time_t cachedTime = time(nullptr);
      cout << "INFO: Outdated cache clear on " << ctime(&cachedTime);
      cacheOutdated = false;
    }
```

```
    auto order = find(cacheOrder.begin(), cacheOrder.end(), filters);
    if (order != cacheOrder.end()) {
      //Результат уже есть в кэше, обновить его положение в очереди
      cacheOrder.erase(order);
      cacheOrder.push_back(filters);
    }
    else {
      if (cacheOrder.size() >= CACHE_SIZE) {
          //Очистка кэша от данных, к которым давно не обращались
          cache.erase(cacheOrder.front());
          cacheOrder.pop_front();
      }
      //Получение результатов из БД и добавление их в кэш
      TimeRecords tr;
      string query = filters.SQLQuery();
      Statement statement(*dbSession);
      TimeRecord t; //Временный объект для хранения данных о посещаемости
      statement << query, into(t.id), into(t.date), into(t.time),
          into(t.present), range(0, 1);
      while (!statement.done()) {
          statement.execute();
          tr.push_back(t);
      }
      cache[filters] = tr;
      assert(tr.size() == cache[filters].size());
      if (cacheOrder.size() == CACHE_SIZE - 1)
          cout << "WARNING: The time record cache is going to be overflowed. Old
data will be erased until they are required again." << endl;;
      time_t cachedTime = time(nullptr);
      cacheOrder.push_back(filters);
      cout << "INFO: Cached time records for filters \'" << filters.toString() << "\' on "
<< ctime(&cachedTime);
    }
  } catch (Poco::Exception ex) {
    cerr << "ERROR: Time record cache update error!" << endl;
    cerr << ex.displayText() << endl;
    return false;
  }
  return true;
}

Object* TimeRecordsRequestHandler::getResponseObject() {
  static Array actions;
```

```
  static Object required;
  if (actions.size() == 0) {
    for (auto action: {"add", "alter", "remove"}) {
      actions.add(action);
      TimeRecordAgent tra(dbSession); //Для доступа к виртуальным методам
      Array props;
      for (auto prop: tra.requiredProperties())
          props.add(prop);
      if (action != "remove")
          props.add("present");
      required.set(action, props);
    }
  }
  string f = filters.toString();
  if (!f.empty())
    f.insert(0, "?");
  return responseObject(cacheArray(filters), actions, required, f);
}
```

8. Базовый обработчик запросов на изменение посредством агентов взаимодействия с базой данных

8.1. Заголовочный файл (handlers/agent.hpp)

```
#include "agents/agent.hpp" //Полиморфный шаблон для всех агентов
#include <Poco/Net/HTTPRequestHandler.h>
#include <Poco/Net/HTTPServerRequest.h>
#include <Poco/Net/HTTPServerResponse.h>
#include <Poco/Data/Session.h>
#include <Poco/JSON/Object.h>
#ifndef HANDLERS_AGENT_HPP_INCLUDED
#define HANDLERS_AGENT_HPP_INCLUDED

using namespace std;
using namespace Poco::Net;
using Poco::Data::Session;
using Poco::JSON::Object;

class AgentRequestHandler: public HTTPRequestHandler{
  typedef HTTPServerRequest Request;
  typedef HTTPServerResponse Response;
public:
  //Конструкторы и деструкторы класса
  AgentRequestHandler(DBAgent* agent);
  ~AgentRequestHandler();
  //Управление агентами
```

```
  DBAgent* setAgent(DBAgent* agent); //Назначить уже существующего агента
  DBAgent* getAgent(); //Получить указатель на текущего агента
  void deleteAgent(); //Удалить агента
  //Параметры обработки запроса
  bool getDefaults = true; //Получать из запроса значения по умолчанию
protected:
  DBAgent* agent = nullptr; //Агент обработчика запросов
  //Обработка HTTP-запросов
  void badRequest(Response& res, const string& error = ""); //Ошибка
некорректного запроса
  void handleRequest(Request& req, Response& res); //Обработка запроса
};
#endif
```

8.2. Реализация класса (handlers/agent.cpp)

```
#include "agent.hpp"
#include <Poco/Data/DataException.h>
#include <Poco/JSON/Parser.h>
#include <Poco/Dynamic/Var.h>

using namespace Poco::JSON;
using Poco::Dynamic::Var;
extern Parser parser;

//Вспомогательные функции
void fillLackingValues(Object& a, const Object& b) {
  //Взять из объекта b те значения, которых нет в объекте a
  for (auto it = b.begin(); it != b.end(); it++) {
    if (!a.has(it->first))
      a.set(it->first, it->second);
  }
}

//Конструкторы и деструктор обработчика запросов
AgentRequestHandler::AgentRequestHandler(DBAgent* agent) {
  setAgent(agent);
}

AgentRequestHandler::~AgentRequestHandler() {
  deleteAgent();
}

//Работа с агентами
DBAgent* AgentRequestHandler::setAgent(DBAgent* agent) {
```

```
  deleteAgent(); //Удалить старого агента из памяти
  this->agent = agent;
  return agent;
}

DBAgent* AgentRequestHandler::getAgent() {
  return agent;
}

void AgentRequestHandler::deleteAgent() {
  if (agent != nullptr)
    delete agent;
}

//Стандартный обработчик HTTP-запросов с применением агентов
void AgentRequestHandler::badRequest(HTTPServerResponse& res, const string&
error) {
  //Отправка ошибки некорректного запроса (400) с указанием ошибки
синтаксиса
  res.setStatusAndReason(HTTPResponse::HTTP_BAD_REQUEST);
  ostream& resBody = res.send();
  if (error != "") {
    res.setContentType("text/plain");
    resBody << "Bad request with the following " << error; //...Exception:...
    resBody << endl;
    //Отправка предупреждения в консоль сервера
    cerr << "ERROR: The request is bad (400), so the handler threw " << error;
//...Exception...
    cerr << endl;
  }
  resBody.flush();
}

void AgentRequestHandler::handleRequest(HTTPServerRequest& req,
HTTPServerResponse& res) {
  try {
    string method = req.getMethod();
    if (method == HTTPRequest::HTTP_GET)
      return; //Данный обработчик не обрабатывает GET-запросы
    if (agent == nullptr)
      throw ConnectionFailedException("Agent is not set");

    //Получение тела запроса
```

```
istream& reqBody = req.stream();
string reqBodyStr;
Var result; //Результат парсинга JSON из тела запроса
try {
  while (!reqBody.eof()) {
      string s;
      reqBody >> s;
      reqBodyStr += s;
  }
  result = parser.parse(reqBodyStr);
} catch (JSONException ex) {
  badRequest(res, ex.displayText());
  cerr << "Full request body:" << endl << reqBodyStr << endl;
  return;
}

Object::Ptr resultObj = result.extract<Object::Ptr>();

//Получение парамтеров по умолчанию (опционально)
if (getDefaults) {
  Object defaults;
  if (resultObj->has("defaults"))
      defaults = *resultObj->getObject("defaults");
  //Перенастройка агента под полученные параметры)
  if (defaults.size() > 0)
      fillLackingValues(agent->defaults, defaults);
}
//Определние выполняемого действия по типу и содержанию запроса
string action = "alter"; //Выполняемое действие
bool isDELETE = (method == HTTPRequest::HTTP_DELETE);
if (isDELETE) {
  action = "remove"; //Для DELETE-запроса всегда равно "remove"
} else if (resultObj->has("action")) {
  action = resultObj->getValue<string>("action");
}

if (!(action == "add" || action == "remove"))
  action = "alter"; //Исправление action на значение по умолчанию

//Выполнение выбранного действия (act)
Array::Ptr data = resultObj->getArray("data");
size_t fails = 0; //Счётчик невыполненных действий
for (Var item: *data) {
```

```
      Object::Ptr obj = item.extract<Object::Ptr>();
      try {
          bool success = agent->act(action, *obj);
          if (!success) fails++;
      }
      catch (Poco::Exception ex) {
          cerr << ex.displayText() << endl;
          fails++;
      }
    }

    //Определние статуса ответа
    typedef HTTPResponse::HTTPStatus Status;
    Status status = HTTPResponse::HTTP_OK; //Статус ответа по умолчанию
    if (action == "add") {
      if (fails == 0)
          status = HTTPResponse::HTTP_CREATED;
      else
          status = HTTPResponse::HTTP_ACCEPTED;
    }
    else if (action == "remove") {
      if (fails == 0 && !isDELETE)
          status = HTTPResponse::HTTP_NO_CONTENT;
    }
    else //status == "alter"
      if (fails != 0)
          status = HTTPResponse::HTTP_ACCEPTED;

    //Отправка ответа
    res.setStatusAndReason(status);
    res.setContentType("text/plain");
    ostream& resBody = res.send(); //Поток тела ответа
    ostringstream failString; //Строковый поток с числом ошибок выполнения
запроса
    failString << fails << " fail" << (fails != 1 ? "s": "");
    if (fails == 0)
      resBody << "Success!" << endl;
    else
      resBody << failString.str() << endl;
    resBody.flush(); //Завершение отправки тела ответа
    cout << "INFO: Request done with " << failString.str() << endl;
  } catch (Poco::Exception ex) {
    cerr << "Time altering (agent) request handling error!" << endl;
```

```
    cerr << ex.displayText() << endl;
    //Отправка ошибки 501

res.setStatusAndReason(HTTPResponse::HTTP_INTERNAL_SERVER_ERROR);
    res.send().flush();
  }
}

void DBAgent::prepare() {
  sql->reset(*dbSession);
}
```

9. Базовый класс агента взаимодействия с БД

9.1. Заголовочный файл (handlers/agents/agent.hpp)

```
#include <Poco/Data/Session.h>
#include <Poco/Data/Statement.h>
#include <Poco/JSON/Object.h>
#include <Poco/Net/NetException.h> //Для вызова "сетевых" исключений из-под методов агента
#include <vector>
#ifndef HANDLERS_AGENTS_AGENT_HPP_INCLUDED
#define HANDLERS_AGENTS_AGENT_HPP_INCLUDED

using namespace std;
using namespace Poco::Data;
using namespace Poco::JSON;
using namespace Poco::Net;

class DBAgent {
  friend class ReqFactory;
public:
  //Конструкторы и деструктор класса
  DBAgent(Session* dbSession, const Object& defaults = Object());
  virtual ~DBAgent();
  //Параметры внесения изменений
  bool addIfNotExist = true; //При работе метода alter() добавлять новые записи, если они не существуют в БД
  bool ignoreIdleRemoval = true; //Считать попытки удаления несуществующих записей успешным удалением
  Object defaults; //Значения свойств JSON-объектов по умолчанию
  //Виртуальные методы обработчика (при подаче аргументов через JSON-объекты)
  //Возвращаемые значения: true = операция прошла успешно, false = ошибка
  virtual bool add(const Object& obj) = 0; //Добавить запись
```

```
  virtual bool alter(const Object& obj) = 0; //Изменить запись
  virtual bool remove(const Object& obj) = 0; //Удалить запись
  virtual bool act(const string& method, const Object& obj); //Выполнить одно из
выше объявленных действий, указанное в строке method
  //Методы проверки целостности и дополнения данных
  virtual bool complete(Object& obj); //Дополняет объект недостающими данными
(true, если данных по умолчанию хватило для заполнения)
  bool full(const Object& obj); //Проверяет полноту объекта без дополнения
  //Возращаемые параметры объекта
  virtual const std::vector<string> requiredProperties(); //Список необходимых
свойств JSON-объекта
  Session* getDBSession(); //Вернуть dbSession
protected:
  Session* dbSession = nullptr; //Сессия связи с БД
  Statement* sql = nullptr; //Общий объект выражения SQL
  void prepare(); //Подготовиться к следующей транзакции
};
#endif
```

9.2. Реализация класса (handlers/agents/agent.cpp)

```
#include "agent.hpp"
#include <Poco/Dynamic/Var.h>

using Poco::Dynamic::Var;

DBAgent::DBAgent(Session* dbSession, const Object& defaults) {
  this->dbSession = dbSession;
  this->sql = new Statement(*dbSession);
  this->defaults = defaults;
}

DBAgent::~DBAgent() {
  if (sql != nullptr)
    delete sql;
}

bool DBAgent::complete(Object& obj) {
  for (string prop: requiredProperties()) {
    Var defaultValue = defaults.get(prop); //isEmpty() = true, если ключ prop не задан
    if (!obj.has(prop) && !defaultValue.isEmpty())
      obj.set(prop, defaultValue);
  }
  return full(obj);
}
```

```
bool DBAgent::full(const Object& obj) {
  for (string prop: requiredProperties())
    if (!obj.has(prop))
      return false; //Объект имеет неполный набор свойств
  return true; //Объект имеет все необходимые свойства
}

bool DBAgent::act(const string& method, const Object& obj) {
  if (method == "add")
    return add(obj);
  if (method == "remove")
    return remove(obj);
  return alter(obj); //Метод по умолчанию
}

const std::vector<string> DBAgent::requiredProperties() {
  return {}; //По умолчанию у агента нет требований к свойствам JSON-объектов
}

Session* DBAgent::getDBSession() {
  return dbSession;
}
```

10. Агент взаимодействия с таблицей групп

10.1. Заголовочный файл (handlers/agents/groups.hpp)

```
#include "agent.hpp"
#ifndef HANDLERS_AGENT_GROUPS_HPP_INCLUDED
#define HANDLERS_AGENT_GROUPS_HPP_INCLUDED

using namespace std;
using namespace Poco::Data;

class GroupsAgent: public DBAgent {
  friend class AlterGroupsRequestHandler;
protected:
  inline void outdateCache(); //Объявить кэш GET-запросов устаревшим
public:
  //Конструктор класса
  using DBAgent::DBAgent;
  //Добавление новой группы в БД (true при успехе)
  bool add(string name, int* idStorage = nullptr); //ID добавленной группы
записывается по указателю idStorage
```

```
  bool add(const Object& obj) override;
  //Изменение названия группы в БД (true при успехе)
  bool alter(int id, string newName); //По ИД группы в БД
  bool alter(string name, string newName); //По старому названию
  bool alter(const Object& obj) override;
  //Удаление группы из БД (true при успехе, вместе с группой удаляются все
студенты в ней)
  bool remove(int id); //По ИД группы в БД
  bool remove(string name); //По названию группы
  bool remove(const Object& obj) override;
};
#endif
```

10.2. Реализация класса (handlers/agents/groups.cpp)

```
#include "groups.hpp"
#include "../groups.hpp"
#include <Poco/Data/Statement.h>
#include <Poco/Data/RecordSet.h>

using namespace Poco::Data::Keywords;

void GroupsAgent::outdateCache() {
  GroupsRequestHandler::cacheOutdated = true;
}

bool GroupsAgent::add(string name, int* idStorage) {
  prepare();
  cout << "Adding group " << name << " to the database..." << endl;
  *sql << "INSERT INTO groups (name) VALUES ($1);", use(name);
  size_t qnt = sql->execute(); //Кол-во добавленных строк (1 или 0)
  if (idStorage != nullptr && qnt != 0) {
    RecordSet rs(*sql);
    int id = rs.row(0).get(0).convert<int>();
    *idStorage = id;
    cout << "INFO: The new group ID (" << id << ") was written to pointer " <<
idStorage << endl;
  }
  if (qnt != 0)
    outdateCache();
  return qnt != 0;
}

bool GroupsAgent::add(const Object& obj) {
```

```
  if (obj.has("name"))
    return add(obj.getValue<string>("name"));
  throw MessageException("Insufficent data to complete the request", 1);
}

bool GroupsAgent::alter(int id, string newName) {
  prepare();
  cout << "Renaming group #" << id << " to " << newName << endl;
  *sql << "UPDATE groups SET name = $1 WHERE id = $2;", use(newName),
use(id);
  outdateCache();
  return sql->execute() != 0;
}

bool GroupsAgent::alter(string name, string newName) {
  prepare();
  cout << "Renaming group " << name << " to " << newName << endl;
  *sql << "UPDATE groups SET name = $1 WHERE name = $2;", use(newName),
use(name);
  outdateCache();
  return sql->execute() != 0;
}

bool GroupsAgent::alter(const Object& obj) {
  if (obj.has("newName")) {
    string newName = obj.getValue<string>("newName");
    if (obj.has("name"))
      return alter(obj.getValue<string>("name"), newName);
    if (obj.has("id"))
      return alter(obj.getValue<int>("id"), newName);
  }
  throw MessageException("Insufficent data to complete the request", 1);
}

bool GroupsAgent::remove(int id) {
  prepare();
  cout << "Removing group #" << id << " from the database..." << endl;
  *sql << "DELETE FROM groups WHERE id = $1;", use(id);
  outdateCache();
  return sql->execute() != 0;
}

bool GroupsAgent::remove(string name) {
```

```
  prepare();
  cout << "Removing group " << name << " from the database..." << endl;
  *sql << "DELETE FROM groups WHERE name = $1;", use(name);
  return sql->execute() != 0;
}

bool GroupsAgent::remove(const Object& obj) {
  if (obj.has("name"))
    return remove(obj.getValue<string>("name"));
  if (obj.has("id"))
    return remove(obj.getValue<int>("id"));
  throw MessageException("Insufficent data to complete the request", 1);
}
```

11. Агент взаимодействия с таблицей студентов

11.1. Заголовочный файл (handlers/agents/students.hpp)

```
#include "agent.hpp"
#include "../students.hpp"
#ifndef HANDLERS_AGENT_STUDENTS_HPP_INCLUDED
#define HANDLERS_AGENT_STUDENTS_HPP_INCLUDED

using namespace std;
using namespace Poco::Data;

class StudentsAgent: public DBAgent {
protected:
  inline void outdateCache(); //Объявить кэш GET-запросов устаревшим
public:
  //Конструктор класса
  using DBAgent::DBAgent;
  //Парамтеры работы с таблицей групп при выполнении запросов
  bool addGroupIfNotExists = false; //Создавать новую группу для студента, если
она не существует в БД
  bool removeGroupIfEmpty = false; //Удалять группу, если в ней не осталось
студентов
  //Дополнение объектов
  bool complete(Object& obj) override; //Дополнение без предназначения
  bool complete(Object& obj, const string& purpose); //Дополнение для конкретной
операции
  bool full(const Object& obj, const string& purpose); //Проверка на полноту
данных для конкретной операции
  //Добавление нового студента (true при успехе)
  bool add(int groupId, string fName, string lName, string mName = "");
```

```
  bool add(string group, string fName, string lName, string mName = ""); //ИД
группы по её имени
  bool add(const Object& obj) override;
  //Изменение данных студента (true при успехе)
  bool rename(int id, string fName, string lName, string mName = "");
//Переименовать студента
  bool move(int id, int groupId); //Переместить студента в другую группу
  bool move(int id, string group); //ИД группы по её имени
  bool alter(const Object& obj) override; //rename() + move()
  //Удаление студента из БД (true при успехе)
  bool remove(int id);
  bool remove(const Object& obj) override;
  //Константы
  static constexpr int NULL_GROUP = -1;
  static constexpr const char* PURPOSE_ADD = "add"; //Добавление студента
  static constexpr const char* PURPOSE_RENAME = "rename"; //Переименование
студента
  static constexpr const char* PURPOSE_MOVE = "move"; //Перевод студента в
другую группу
  static constexpr const char* PURPOSE_ALTER = "alter"; //Полное изменение
данных о студенте
  static constexpr const char* PURPOSE_REMOVE = "remove"; //Удаление
студента
  static const char* PURPOSE_ALL[5]; //Список всех действий
  //Списки обязательных свойств JSON-объектов
  const std::vector<string> requiredProperties() override; //Общий список
обязательных свойств
  const std::vector<string> requiredProperties(const string& purpose); //Список
обязательных свойств для конкретной операции
protected:
  //Поиск группы по её имени
  static int groupIdByName(string name); //ИД группы по её имени. При неудаче
возращеает NULL_GROUP;
  int resolveGroupName(string name); //Получить ИД существующей группы по
её имени или создать новую с этим именем (если это разрешено). При неудаче
поднимет исключение MessageException.
  inline void manualComplete(Object& obj); //Уникальные для этого класса
дополнения без циклов
};
#endif
```

11.2. Реализация класса (handlers/agents/students.cpp)

```
#include "groups.hpp"
```

```
#include "../groups.hpp"
#include <Poco/Data/Statement.h>
#include <Poco/Data/RecordSet.h>

using namespace Poco::Data::Keywords;

void GroupsAgent::outdateCache() {
  GroupsRequestHandler::cacheOutdated = true;
}

bool GroupsAgent::add(string name, int* idStorage) {
  prepare();
  cout << "Adding group " << name << " to the database..." << endl;
  *sql << "INSERT INTO groups (name) VALUES ($1);", use(name);
  size_t qnt = sql->execute(); //Кол-во добавленных строк (1 или 0)
  if (idStorage != nullptr && qnt != 0) {
    RecordSet rs(*sql);
    int id = rs.row(0).get(0).convert<int>();
    *idStorage = id;
    cout << "INFO: The new group ID (" << id << ") was written to pointer " <<
idStorage << endl;
  }
  if (qnt != 0)
    outdateCache();
  return qnt != 0;
}

bool GroupsAgent::add(const Object& obj) {
  if (obj.has("name"))
    return add(obj.getValue<string>("name"));
  throw MessageException("Insufficent data to complete the request", 1);
}
bool GroupsAgent::alter(int id, string newName) {
  prepare();
  cout << "Renaming group #" << id << " to " << newName << endl;
  *sql << "UPDATE groups SET name = $1 WHERE id = $2;", use(newName),
use(id);
  outdateCache();
  return sql->execute() != 0;
}

bool GroupsAgent::alter(string name, string newName) {
  prepare();
```

```
  cout << "Renaming group " << name << " to " << newName << endl;
  *sql << "UPDATE groups SET name = $1 WHERE name = $2;", use(newName),
use(name);
  outdateCache();
  return sql->execute() != 0;
}

bool GroupsAgent::alter(const Object& obj) {
  if (obj.has("newName")) {
    string newName = obj.getValue<string>("newName");
    if (obj.has("name"))
      return alter(obj.getValue<string>("name"), newName);
    if (obj.has("id"))
      return alter(obj.getValue<int>("id"), newName);
  }
  throw MessageException("Insufficent data to complete the request", 1);
}

bool GroupsAgent::remove(int id) {
  prepare();
  cout << "Removing group #" << id << " from the database..." << endl;
  *sql << "DELETE FROM groups WHERE id = $1;", use(id);
  outdateCache();
  return sql->execute() != 0;
}

bool GroupsAgent::remove(string name) {
  prepare();
  cout << "Removing group " << name << " from the database..." << endl;
  *sql << "DELETE FROM groups WHERE name = $1;", use(name);
  return sql->execute() != 0;
}

bool GroupsAgent::remove(const Object& obj) {
  if (obj.has("name"))
    return remove(obj.getValue<string>("name"));
  if (obj.has("id"))
    return remove(obj.getValue<int>("id"));
  throw MessageException("Insufficent data to complete the request", 1);
}
```

12. Агент взаимодействия с записями о посещаемости

12.1. *Заголовочный файл (handlers/agents/timerecords.hpp)*

```
#include "agent.hpp"
```

```
#include "../agent.hpp"
#ifndef HANDLERS_AGENT_TIMERECORDS_HPP_INCLUDED
#define HANDLERS_AGENT_TIMERECORDS_HPP_INCLUDED

class TimeRecordAgent: public DBAgent {
protected:
  inline void outdateCache(); //Объявить кэш GET-запросов устаревшим
public:
  //Конструктор класса
  using DBAgent::DBAgent;
  //Добавление строк в БД (true при успехе)
  bool add(int id, string date, string time, bool present); //Полный вызов
  bool add(const Object& obj) override; //Вывоз с JSON-объектом
  //Изменение строк в БД (true при успехе)
  bool alter(int id, string date, string time, bool present); //Полный вызов
  bool alter(const Object& obj) override; //Вывоз с JSON-объектом
  //Удаление записей из БД (true при успехе)
  bool remove(int id, string date, string time); //Полный вызов
  bool remove(const Object& obj) override; //Вывоз с JSON-объектом
  //Проверка JSON-объектов
  const std::vector<string> requiredProperties() override; //Необх. свойства
поступающих объектов
};
#endif
```

12.2. Реализация класса (handlers/agents/timerecords .cpp)

```
#include "timerecords.hpp"
#include "../timerecords.hpp"
#include <Poco/Net/NetException.h>
#include <Poco/Dynamic/Var.h>

using namespace Poco::Data::Keywords;
using Poco::Dynamic::Var;
typedef std::vector<int> ivec;

//Вспомогательные функции
inline void TimeRecordAgent::outdateCache() {
  TimeRecordsRequestHandler::cacheOutdated = true;
}

const std::vector<string> TimeRecordAgent::requiredProperties() {
  return {"id", "date", "time"};
}
```

```
//Добавление
bool TimeRecordAgent::add(int id, string date, string time, bool present) {
  try {
    prepare();
    *sql << "INSERT INTO timerecords VALUES ($1, $2, $3, $4);", use(id),
      use(date), use(time), use(present), now;
    outdateCache();
    return true;
  } catch(ExecutionException ex) {
    //Ничего не делает, т.к. отчёты об ошибках в запросах обычно выводит СУБД
  }
  return false;
}

bool TimeRecordAgent::add(const Object& obj) {
  Object cObj = obj; //Дополненная версия объекта
  bool completed = complete(cObj); //Дополняем объект при необходимости
  if (!completed)
    throw MessageException("Insufficent data to complete the request", 1);
  if (obj.has("present")) { //В объекте указаны изменения посещаемости
    int id = obj.getValue<int>("id");
    string d = obj.getValue<string>("date");
    string t = obj.getValue<string>("time");
    bool p = obj.getValue<bool>("present");
    return add(id, d, t, p);
  }
  return false; //Нет данных об оценке
}

//Изменение
bool TimeRecordAgent::alter(int id, string date, string time, bool present) {
  try {
    prepare();
    *sql << "UPDATE timerecords SET presence = $1 WHERE studId = $2 AND date
= $3 AND time = $4;", use(present), use(id), use(date), use(time);
    size_t altered = sql->execute();
    if (altered == 0) { //Изменений не было => нет записи в БД
      if (addIfNotExist)
          return add(id, date, time, present); //Создать запись в БД
      return false; //Невозможно изменить несуществующую запись
    }
    outdateCache();
    return true;
```

```
  } catch(ExecutionException ex) {
    //Ничего не делает, т.к. отчёты об ошибках в запросах обычно выводит СУБД
  }
  return false;
}

bool TimeRecordAgent::alter(const Object& obj) {
  Object cObj = obj; //Дополненная версия объекта
  bool completed = complete(cObj); //Дополняем объект при необходимости
  if (!completed)
    throw MessageException("Insufficent data to complete the request", 1);
  if (obj.has("present")) { //В объекте указаны изменения посещаемости
    int id = obj.getValue<int>("id");
    string d = obj.getValue<string>("date");
    string t = obj.getValue<string>("time");
    bool p = obj.getValue<bool>("present");
    return alter(id, d, t, p);
  }
  return true; //Нет данных об изменениях => нечего изменять
}

//Удаление
bool TimeRecordAgent::remove(int id, string date, string time) {
  try {
    prepare();
    *sql << "DELETE FROM timerecords WHERE studId = $1 AND date = $2 AND
time = $3;", use(id), use(date), use(time);
    size_t altered = sql->execute();
    if (altered == 0) //Изменений не было => нет записи в БД
      return ignoreIdleRemoval;
    outdateCache();
    return true;
  } catch(ExecutionException ex) {
    //Ничего не делает, т.к. отчёты об ошибках в запросах обычно выводит СУБД
  }
  return false;
}

bool TimeRecordAgent::remove(const Object& obj) {
  Object cObj = obj; //Дополненная версия объекта
  bool completed = complete(cObj); //Дополняем объект при необходимости
  if (!completed)
    throw MessageException("Insufficent data to complete the request", 1);
```

```
  int id = obj.getValue<int>("id");
  string d = obj.getValue<string>("date");
  string t = obj.getValue<string>("time");
  return remove(id, d, t);
}
```

ПРИЛОЖЕНИЕ Б
к отчёту по производственной эксплуатационной практике
«Исходный код клиентского приложения»

1. **Основная программа клиента (main.cpp)**

```
#include "mainwindow.h"
#include <QApplication>
int main(int argc, char *argv[])
{
   QApplication a(argc, argv);
   MainWindow w;
   w.show();
   return a.exec();
}
```

2. **Главное окно программы**

2.1 Заголовочный файл (mainwindow.h)

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "checkattendance.h"
#include "markattendance.h"
#include "studentslist.h"
#include "download.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
   Q_OBJECT

public:
   MainWindow(QWidget *parent = nullptr);
   ~MainWindow();
private slots:
   void on_check_clicked();
   void on_mark_clicked();
   void on_list_clicked();

private:
   Ui::MainWindow *ui;
   CheckAttendance *check;
```

```
    MarkAttendance *mark;
    StudentsList *list;
    Download *download;
};
#endif // MAINWINDOW_H
```

2.2 *Реализация класса (mainwindow.cpp)*

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "checkattendance.h"
#include "markattendance.h"
#include "studentslist.h"
#include "download.h"

#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    mark = new MarkAttendance ();   // Инициируем окно отметки посещаемости
    connect(mark, &MarkAttendance::firstWindow, this,
&MarkAttendance::show);   // Подключаемся к слоту открытия главного окна

    check = new CheckAttendance (); // Инициируем окно просмотра
посещаемости
    connect(check, &CheckAttendance::firstWindow, this, &CheckAttendance::show);

    list = new StudentsList (); // Инициируем окно просмотра посещаемости
    connect(list, &StudentsList::firstWindow, this, &StudentsList::show);

    download = new Download (); // Инициируем новый объект загрузки
    connect(ui->check, SIGNAL(clicked()), download, SLOT(getDataGroup()));  //
получаем данные по нажатию на кнопку
    connect(ui->mark, SIGNAL(clicked()), download, SLOT(getDataGroup()));
    connect(ui->list, SIGNAL(clicked()), download, SLOT(getDataGroup()));
}

MainWindow::~MainWindow()
```

```
{
   delete ui;
   // Удаляем все созданные в процессе работы программы файлы
   QFile *addStud = new QFile (QCoreApplication::applicationDirPath() +
"/addStudent.json");
   addStud->remove();

   QFile *alterStud = new QFile (QCoreApplication::applicationDirPath() +
"/alertStudent.json");
   alterStud->remove();

   QFile questGroup(QCoreApplication::applicationDirPath() + "/group.json");
   questGroup.remove();

   QFile questStud(QCoreApplication::applicationDirPath() + "/student.json");
   questStud.remove();

   QFile questPres(QCoreApplication::applicationDirPath() + "/timerecord.json");
   questPres.remove();

   QFile *delStud = new QFile (QCoreApplication::applicationDirPath() +
"/deleteStudent.json");
   delStud->remove();

   QFile timeStud(QCoreApplication::applicationDirPath() + "/time.json");
   timeStud.remove();

   QFile presentAbsent(QCoreApplication::applicationDirPath() + "/pres.txt");
   presentAbsent.remove();
}

void MainWindow::on_check_clicked()  // Открываем окно просмотра
посещаемости
{
   check->show();
   this->close();
   connect(download, &Download::onReady, check,
&CheckAttendance::readGroup);
}

void MainWindow::on_mark_clicked()  // Открываем окно отметки посещаемости
{
   mark->show();
```

```
    this->close();
    connect(download, &Download::onReady, mark, &MarkAttendance::readGroup);
}

void MainWindow::on_list_clicked()  // Открываем окно со списком студентов
{
    list->show();
    this->close();
    connect(download, &Download::onReady, list, &StudentsList::readGroup);
}
```

3. Объект загрузки

3.1 Заголовочный файл (download.h)

```
#ifndef DOWNLOAD_H
#define DOWNLOAD_H

#include <QObject>
#include <QtNetwork/QNetworkAccessManager>
#include <QtNetwork/QNetworkReply>
#include <QtNetwork/QNetworkRequest>
#include <QUrl>
#include <QFile>
#include <QDebug>

class Download : public QObject
{
    Q_OBJECT
public:
    explicit Download(QObject *parent = 0);
    QString comboGroup;

signals:
    void onReady();

public slots:
    void getDataGroup();
    void getDataStud(QString comboGroup);
    void onResultGroup(QNetworkReply *reply);
    void onResultStud(QNetworkReply *reply);
    void onResultPres(QNetworkReply *reply);

private:
    QNetworkAccessManager *managerGroup;
```

```
    QNetworkAccessManager *managerStud;
    QNetworkAccessManager *managerPers;
};

#endif // DOWNLOAD_H
```

Реализация класса (download.cpp)

```
#include "download.h"
#include <QCoreApplication>

Download::Download(QObject *parent)
    :QObject(parent)
{
    managerGroup = new QNetworkAccessManager ();
    connect(managerGroup, &QNetworkAccessManager::finished, this,
&Download::onResultGroup);

    managerStud = new QNetworkAccessManager ();
    connect(managerStud, &QNetworkAccessManager::finished, this,
&Download::onResultStud);

    managerPers = new QNetworkAccessManager ();
    connect(managerPers, &QNetworkAccessManager::finished, this,
&Download::onResultPres);
}

void Download::getDataGroup()
{
    QUrl studGroup("http://192.168.0.10/groups"); // Стучимся туда, где расположен
файл
    QNetworkRequest requestGroup;
    requestGroup.setUrl(studGroup);
    managerGroup->get(requestGroup);  // Отправляем запрос
}

void Download::getDataStud(QString comboGroup)
{
    if (comboGroup == "Все группы")
    {
        QUrl stud("http://192.168.0.10/groups/all"); // Стучимся туда, где расположен
файл
        QNetworkRequest requestStud;
        requestStud.setUrl(stud);
```

```
        managerStud->get(requestStud);  // Отправляем запрос
    }
    else
    {
        QUrl stud("http://192.168.0.10/groups/" + comboGroup); // Стучимся туда, где
расположен файл
        QNetworkRequest requestStud;
        requestStud.setUrl(stud);
        managerStud->get(requestStud);  // Отправляем запрос
    }

    QUrl studPres("http://192.168.0.10/timerecords"); // Стучимся туда, где
расположен файл
    QNetworkRequest requestPres;
    requestPres.setUrl(studPres);
    managerPers->get(requestPres);  // Отправляем запрос
}

void Download::onResultGroup(QNetworkReply *reply)
{
    if (reply->error()) // Проверка наличия ошибок во время выполнения запроса
    {
        qDebug() << "Error";
        qDebug() << reply->errorString();
    }
    else
    {
        QFile *questGroup = new QFile (QCoreApplication::applicationDirPath() +
"/group.json"); // Создаем новый файл и копируем туда содержимое полученного
файла для дальнейшей обработки
        if(questGroup->open(QFile::WriteOnly))
        {
            questGroup->write(reply->readAll());
            questGroup->close();
        }
        qDebug() << "Download is completed";
        emit onReady();
    }
}

void Download::onResultStud(QNetworkReply *reply)
{
    if (reply->error()) // Проверка наличия ошибок во время выполнения запроса
```

```
    {
        qDebug() << "Error";
        qDebug() << reply->errorString();

    }
    else
    {
        QFile *questStud = new QFile (QCoreApplication::applicationDirPath() +
"/student.json"); // Создаем новый файл и копируем туда содержимое
полученного файла для дальнейшей обработки
        if(questStud->open(QFile::WriteOnly))
        {
            questStud->write(reply->readAll());
            questStud->close();
        }
        qDebug() << "Download is completed";
        emit onReady();
    }
}

void Download::onResultPres(QNetworkReply *reply)
{
    if (reply->error()) // Проверка наличия ошибок во время выполнения запроса
    {
        qDebug() << "Error";
        qDebug() << reply->errorString();

    }
    else
    {
        QFile *questPres = new QFile (QCoreApplication::applicationDirPath() +
"/timerecord.json"); // Создаем новый файл и копируем туда содержимое
полученного файла для дальнейшей обработки
        if(questPres->open(QFile::WriteOnly))
        {
            questPres->write(reply->readAll());
            questPres->close();
        }
        qDebug() << "Download is completed";
        emit onReady();
    }
}
```

Делегат поля с галочкой

Заголовочный файл (checkboxdelegate.h)

```
#ifndef CHECKBOXDELEGATE_H
#define CHECKBOXDELEGATE_H

#include <QStyledItemDelegate>

QT_BEGIN_NAMESPACE
namespace Ui { class CheckBoxDelegate; }
QT_END_NAMESPACE

class CheckBoxDelegate : public QStyledItemDelegate
{
   Q_OBJECT

public:
   CheckBoxDelegate(QObject *parent = nullptr);

private:
   Ui::CheckBoxDelegate *ui;
   void paint(QPainter *painter, const QStyleOptionViewItem &option, const
QModelIndex &index) const;
   QWidget *createEditor(QWidget *parent, const QStyleOptionViewItem &option,
const QModelIndex &index) const;
   void setEditorData(QWidget *editor, const QModelIndex &index) const;
   void setModelData(QWidget *editor, QAbstractItemModel *model, const
QModelIndex &index) const;
   void updateEditorGeometry(QWidget *editor, const QStyleOptionViewItem
&option, const QModelIndex &index) const;
};
#endif // CHECKBOXDELEGATE_H
```

3.2 Реализация класса (checkboxdelegate.cpp)

```
#include "checkboxdelegate.h"

#include <QCheckBox>
#include <QApplication>
#include <QDebug>
#include <QFile>

#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>
```

```
CheckBoxDelegate::CheckBoxDelegate(QObject *parent)
   :QStyledItemDelegate (parent)
{

}

QWidget *CheckBoxDelegate::createEditor(QWidget *parent, const
QStyleOptionViewItem &option, const QModelIndex &index) const
{
   Q_UNUSED(option)
   Q_UNUSED(index)

   //Создаем checkbox editor
   QCheckBox *editor = new QCheckBox(parent);
   return editor;
}

void CheckBoxDelegate::setEditorData(QWidget *editor, const QModelIndex
&index) const
{
   //Установливаем выбрано/не выбрано
   QCheckBox *cb = qobject_cast<QCheckBox *>(editor);
   cb->setChecked(index.data().toBool());
}

void CheckBoxDelegate::setModelData(QWidget *editor, QAbstractItemModel
*model, const QModelIndex &index) const
{
   //Записываем данные в модель
   QCheckBox *cb = static_cast<QCheckBox *>(editor);
   int value = (cb->checkState()==Qt::Checked)? 1 : 0;
   model->setData(index, value, Qt::EditRole);
   QString text;
   if(value == 1)
      text = "true";
   else
      text = "false";

   QFile presentAbsent(QCoreApplication::applicationDirPath() + "/pres.txt");
   if (!presentAbsent.open(QIODevice::Append | QIODevice::Text))
      return;
   QTextStream writeStream(&presentAbsent);
   writeStream << text << "\n";
```

```
    presentAbsent.close();
}

void CheckBoxDelegate::updateEditorGeometry(QWidget *editor, const
QStyleOptionViewItem &option, const QModelIndex &index) const
{
    Q_UNUSED(index);
    QStyleOptionButton checkboxstyle;
    QRect checkbox_rect = QApplication::style()-
>subElementRect(QStyle::SE_CheckBoxIndicator, &checkboxstyle);

    //Центрирование
    checkboxstyle.rect = option.rect;
    checkboxstyle.rect.setLeft(option.rect.x() + option.rect.width()/2 -
checkbox_rect.width()/2);

    editor->setGeometry(checkboxstyle.rect);
}

void CheckBoxDelegate::paint(QPainter *painter, const QStyleOptionViewItem
&option, const QModelIndex &index) const
{
    //Получаем данные
    bool data = index.model()->data(index, Qt::DisplayRole).toBool();

    //Создаем стиль CheckBox
    QStyleOptionButton checkboxstyle;
    QRect checkbox_rect = QApplication::style()-
>subElementRect(QStyle::SE_CheckBoxIndicator, &checkboxstyle);

    //Центрирование
    checkboxstyle.rect = option.rect;
    checkboxstyle.rect.setLeft(option.rect.x() + option.rect.width()/2 -
checkbox_rect.width()/2);
    //Выбрано или не выбрано
    if(data)
        checkboxstyle.state = QStyle::State_On|QStyle::State_Enabled;
    else
        checkboxstyle.state = QStyle::State_Off|QStyle::State_Enabled;

    //Отображаем
    QApplication::style()->drawControl(QStyle::CE_CheckBox, &checkboxstyle,
painter);
```

```
}
```

4. Окно просмотра посещаемости

4.1 Заголовочный файл (checkattendance.h)

```
#ifndef CHECKATTENDANCE_H
#define CHECKATTENDANCE_H

#include <QWidget>
#include <QStandardItem>
#include <QStandardItemModel>

#include "download.h"

namespace Ui {
class CheckAttendance;
}

class CheckAttendance : public QWidget
{
    Q_OBJECT

signals:
    void firstWindow();
    void getGroup(QString comboGroup);

public:
    explicit CheckAttendance(QWidget *parent = nullptr);
    ~CheckAttendance();
    QString comboGroup;

public slots:
    void readGroup();
    void readStud();
    void numGroup();

private slots:
    void on_back_clicked();

private:
    Ui::CheckAttendance *ui;
    Download *download;
    QStandardItemModel *tableCheck;
};
```

```
#endif // CHECKATTENDANCE_H
```

4.2 Реализация класса (checkattendance.cpp)

```
#include "checkattendance.h"
#include "ui_checkattendance.h"
#include "download.h"

#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>

CheckAttendance::CheckAttendance(QWidget *parent) :
   QWidget(parent),
   ui(new Ui::CheckAttendance)
{
   ui->setupUi(this);

   download = new Download (); // Инициируем новый объект загрузки
   connect(this, SIGNAL(getGroup(QString)), download,
SLOT(getDataStud(QString)));
   connect(ui->select, SIGNAL(clicked()), this, SLOT(numGroup()));  // получаем
данные по нажатию на кнопку
   connect(download, &Download::onReady, this,
&CheckAttendance::readStud);    // вызываем функцию чтения файла

   tableCheck = new QStandardItemModel (ui->tableCheck);   // Инициализируем
таблицу
}

CheckAttendance::~CheckAttendance()
{
   delete ui;
}

void CheckAttendance::readGroup()
{
   ui->comboBox->clear();
   tableCheck->clear();
   ui->comboBox->addItem("Все группы");

   QFile questGroup(QCoreApplication::applicationDirPath() + "/group.json");
   if (!questGroup.open(QIODevice::ReadOnly))
      return;
```

```
    QByteArray dataG = questGroup.readAll();   // Парсинг файла с номерами групп
    QJsonDocument documentG = documentG.fromJson(dataG);
    QJsonObject root = documentG.object();
    root.value("data");
    QJsonValue groups = root.value("data");
    if(groups.isArray())
    {
        QJsonArray group = groups.toArray();
        for (int i = 0; i < group.count(); i++) {   // Работа с выпадающим списком
            ui->comboBox->addItem(group[i].toString());
        }
    }
}

void CheckAttendance::readStud()
{
    tableCheck->clear();
    QFile questStud(QCoreApplication::applicationDirPath() + "/student.json");
    if (!questStud.open(QIODevice::ReadOnly))
        return;
    QByteArray dataS = questStud.readAll();   // Парсинг файла со студентами
    QJsonDocument documentS = documentS.fromJson(dataS);
    QJsonObject root = documentS.object();
    root.value("data");
    QJsonValue students = root.value("data");

    QFile questPres(QCoreApplication::applicationDirPath() + "/timerecord.json");
    if (!questPres.open(QIODevice::ReadOnly))
        return;
    QByteArray dataP = questPres.readAll();   // Парсинг файла с посещаемостью
    QJsonDocument documentP = documentP.fromJson(dataP);
    QJsonObject root1 = documentP.object();
    root1.value("data");
    QJsonValue timerecords = root1.value("data");

    QString pres;
    if(timerecords.isArray())
    {
        QJsonArray timerecord = timerecords.toArray();
        if(students.isArray())
        {
            QJsonArray student = students.toArray();
```

```
        tableCheck->setHorizontalHeaderLabels(QStringList() << "Фамилия" <<
"Имя" << "Отчество" << "Номер группы" << "Дата занятия" << "Время занятия"
<< "Был/Не был");
        for (int i = 0; i < student.count(); i++) {
            QJsonObject personal = student.at(i).toObject();
            for (int j = 0; j < timerecord.count(); j++) {
                QJsonObject present = timerecord.at(j).toObject();
                if (personal.value("id").toInt() == present.value("id").toInt())    //
Сравниваем идентификатор студента в списке студентов и посещаемости, чтобы
выводить посещаемость напротив конкретного студента
                {
                    if(present.value("present").toBool() == true)
                        pres = "Присутствовал";
                    else
                        pres = "Отсутствовал";
                    QList<QStandardItem *> items;   // Создаем массив столбцов
таблицы и заполняем его данными
                    items.append(new
QStandardItem(personal.value("lName").toString()));    // Первая колонка
                    items.append(new
QStandardItem(personal.value("fName").toString()));    // Вторая колонка
                    items.append(new
QStandardItem(personal.value("mName").toString()));    // Третья колонка
                    items.append(new
QStandardItem(personal.value("group").toString()));    // Четвертая колонка
                    items.append(new QStandardItem(present.value("date").toString()));
// Пятая колонка
                    items.append(new QStandardItem(present.value("time").toString()));
// Шестая колонка
                    items.append(new QStandardItem(pres));    // Третья колонка
                    tableCheck->appendRow(items);
                }
            }
        }
    }
    ui->tableCheck->resizeColumnsToContents();
    ui->tableCheck->setModel(tableCheck);   // Устанавливаем модель для
таблицы
  }
}

void CheckAttendance::numGroup()
```

```
{
   comboGroup = ui->comboBox->currentText();   // Получаем строку, выбранную
пользователем из выпадающего списка
   emit getGroup(comboGroup);
}

void CheckAttendance::on_back_clicked()
{
   this->close();
   emit firstWindow();
}
```

5. Окно изменения данных о посещаемости

5.1 Заголовочный файл (markattendance.h)

```
#ifndef MARKATTENDANCE_H
#define MARKATTENDANCE_H

#include <QWidget>
#include <QStandardItem>
#include <QStandardItemModel>
#include <QCheckBox>
#include <QStyledItemDelegate>

#include "download.h"
#include "checkboxdelegate.h"

namespace Ui {
class MarkAttendance;
}

class MarkAttendance : public QWidget
{
   Q_OBJECT

signals:
   void firstWindow();
   void getGroup(QString comboGroup);

public:
   explicit MarkAttendance(QWidget *parent = nullptr);
   ~MarkAttendance();
   QString comboGroup;
   QString date;
   QString time;
```

```
    QStringList presents;

public slots:
    void readGroup();
    void readStud();
    void postData();
    void handleEndOfRequest(QNetworkReply* reply);
    void numGroup();

private slots:
    void on_back_clicked();

private:
    Ui::MarkAttendance *ui;
    Download *download;
    QStandardItemModel *tableMark;
    QCheckBox *markPres;
    CheckBoxDelegate *cbd;
};

#endif // MARKATTENDANCE_H
```

5.2 Реализация класса (markattendance.cpp)

```
#include "markattendance.h"
#include "ui_markattendance.h"
#include "download.h"
#include "checkboxdelegate.h"

#include <QtNetwork/QNetworkAccessManager>
#include <QtNetwork/QNetworkReply>
#include <QtNetwork/QNetworkRequest>

#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>

#include <QCheckBox>
#include <QAbstractItemDelegate>

MarkAttendance::MarkAttendance(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::MarkAttendance)
```

```
{
   ui->setupUi(this);

   download = new Download ();
   connect(this, SIGNAL(getGroup(QString)), download,
SLOT(getDataStud(QString)));
   connect(ui->select, SIGNAL(clicked()), this, SLOT(numGroup()));  // получаем
данные по нажатию на кнопку
   connect(download, &Download::onReady, this, &MarkAttendance::readStud);    //
вызываем функцию чтения файла
   connect(ui->save, SIGNAL(clicked()), this, SLOT(postData()));

   tableMark = new QStandardItemModel (ui->tableMark);   // Инициализируем
таблицу
}

MarkAttendance::~MarkAttendance()
{
   delete ui;
}

void MarkAttendance::readGroup()
{
   tableMark->clear();
   ui->comboBox->clear();
   ui->comboBox->addItem("Все группы");

   QFile questGroup(QCoreApplication::applicationDirPath() + "/group.json");
   if (!questGroup.open(QIODevice::ReadOnly))
      return;

   QByteArray dataG = questGroup.readAll();   // Парсинг файла с номерами групп
   QJsonDocument documentG = documentG.fromJson(dataG);
   QJsonObject root = documentG.object();
   root.value("data");
   QJsonValue groups = root.value("data");
   if(groups.isArray())
   {
      QJsonArray group = groups.toArray();
      for (int i = 0; i < group.count(); i++) {   // Работа с выпадающим списком
         ui->comboBox->addItem(group[i].toString());
      }
   }
```

```
}

void MarkAttendance::readStud()
{
   tableMark->clear();
   CheckBoxDelegate *delegate = new CheckBoxDelegate();

   date = ui->dateEdit->date().toString("yyyy/MM/dd");
   time = ui->timeEdit->time().toString();

   QFile questStud(QCoreApplication::applicationDirPath() + "/student.json");
   if (!questStud.open(QIODevice::ReadOnly))
      return;
   QByteArray dataS = questStud.readAll();   // Парсинг файла со студентами
   QJsonDocument documentS = documentS.fromJson(dataS);
   QJsonObject root = documentS.object();
   root.value("data");
   QJsonValue students = root.value("data");
   if(students.isArray())
   {
      QJsonArray student = students.toArray();
      markPres = new QCheckBox ();
      QJsonObject main;
      QJsonArray timerec;
      QJsonObject data;

      main.insert("actions", QJsonValue::fromVariant("add"));

      tableMark->setHorizontalHeaderLabels(QStringList() << "id" << "Фамилия"
<< "Имя" << "Отчество" << "Номер группы" << "Был/Не был");
      for (int i = 0; i < student.count(); i++) {
         QJsonObject personal = student.at(i).toObject();

         QList<QStandardItem *> items;
         items.append(new
QStandardItem(QString::number(personal.value("id").toInt())));    // Скрытая
колонка
         items.append(new QStandardItem(personal.value("lName").toString()));    //
Первая колонка
         items.append(new QStandardItem(personal.value("fName").toString()));    //
Вторая колонка
         items.append(new QStandardItem(personal.value("mName").toString()));    //
Третья колонка
```

```
            items.append(new QStandardItem(personal.value("group").toString()));    // Четвертая колонка
            tableMark->appendRow(items);
            int id = personal.value("id").toInt();
            data.insert("date", QJsonValue::fromVariant(date));
            data.insert("id", QJsonValue::fromVariant(id));
            data.insert("time", QJsonValue::fromVariant(time));
            timerec.push_back(data);
            main.insert("data", timerec);
        }
        ui->tableMark->setItemDelegateForColumn(5, delegate);   // Добавляем солбец с чекбоксами

        QJsonDocument file(main);

        QString jsonString = file.toJson(QJsonDocument::Indented);
        //Записываем данные в файл
        QFile json;
        json.setFileName(QCoreApplication::applicationDirPath() + "/time.json");
        json.open(QIODevice::WriteOnly | QIODevice::Text);
        QTextStream stream( &json );
        stream << jsonString;
        json.close();
    }
    ui->tableMark->setModel(tableMark);   // Устанавливаем модель для таблицы
    ui->tableMark->resizeColumnsToContents();
    ui->tableMark->setColumnHidden(0, true);
//    ui->tableMark->horizontalHeader()->setStretchLastSection(true);
    questStud.close();

    QFile presentAbsent(QCoreApplication::applicationDirPath() + "/pres.txt");
    if (!presentAbsent.open(QIODevice::WriteOnly | QIODevice::Text))
        return;
    presentAbsent.close();
}

void MarkAttendance::handleEndOfRequest(QNetworkReply* reply)
{
    if (reply->error() == QNetworkReply::NoError)   // Проверяем наличие ошибок при выполнении запроса
    {
        qDebug() << "File post";
        reply->deleteLater();
```

```
    }
    else
    {
        qDebug() << reply->errorString();
    }
}

void MarkAttendance::postData()
{
    QFile timeStud(QCoreApplication::applicationDirPath() + "/time.json");
    if (!timeStud.open(QIODevice::ReadOnly))
        return;
    QByteArray dataTimeStud = timeStud.readAll();
    QJsonDocument docTimeStud = docTimeStud.fromJson(dataTimeStud);
    QJsonObject root = docTimeStud.object();
    root.value("data");
    QJsonValue timeStudValue = root.value("data");

    QFile presentAbsent(QCoreApplication::applicationDirPath() + "/pres.txt");
    if (!presentAbsent.open(QIODevice::ReadOnly | QIODevice::Text))
        return;
    while(!presentAbsent.atEnd())
    {
        //читаем строку
        QString str = presentAbsent.readLine();
        if(str == "true\n")
            presents.push_back("true");
        else
            presents.push_back("false");
    }
    presentAbsent.close();

    if(timeStudValue.isArray())
    {
        // Добавляем значения посещаемости в файл
        QJsonArray timeStudElement = timeStudValue.toArray();
        QJsonObject main;
        QJsonArray timerec;
        QJsonObject data;
        bool pres;

        main.insert("action", QJsonValue::fromVariant("add"));
        for (int i = 0; i < timeStudElement.count(); i++) {
```

```
            QJsonObject present = timeStudElement.at(i).toObject();
            QString date = present.value("date").toString();
            int id = present.value("id").toInt();
            QString time = present.value("time").toString();
            if(presents.at(i) == "true")
               pres = true;
            else
               pres = false;

            data.insert("date", QJsonValue::fromVariant(date));
            data.insert("id", QJsonValue::fromVariant(id));
            data.insert("present", QJsonValue::fromVariant(pres));
            data.insert("time", QJsonValue::fromVariant(time));
            timerec.push_back(data);
            main.insert("data", timerec);
         }
         QJsonDocument file(main);

         QString jsonString = file.toJson(QJsonDocument::Indented);
         QFile json;
         json.setFileName(QCoreApplication::applicationDirPath() + "/time.json");
         json.open(QIODevice::WriteOnly | QIODevice::Text);
         QTextStream stream( &json );
         stream << jsonString;
         json.close();
      }
      timeStud.close();

      // Отправляем файл
      QFile *askStud = new QFile (QCoreApplication::applicationDirPath() +
"/time.json");
      if(askStud->open(QFile::ReadOnly))
      {
         QByteArray data = askStud->readAll();
         QJsonDocument file = file.fromJson(data);

         QUrl resource("http://192.168.0.10/timerecords");
         QNetworkAccessManager* postData = new QNetworkAccessManager(this);
         connect(postData, SIGNAL(finished(QNetworkReply*)), this,
SLOT(handleEndOfRequest(QNetworkReply*)));

         QNetworkRequest request(resource);
         request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");
```

```
        postData->post(request, data);
    }
}

void MarkAttendance::numGroup()
{
    comboGroup = ui->comboBox->currentText();   // Получаем строку, выбранную
пользователем из выпадающего списка
    emit getGroup(comboGroup);
}

void MarkAttendance::on_back_clicked()  // Открываем главное окно
{
    this->close();
    emit firstWindow();
}
```

6. Код окна с данными студентов

6.1 Заголовочный файл (studentslist.h)

```
#ifndef STUDENTSLIST_H
#define STUDENTSLIST_H

#include <QWidget>
#include <QStandardItem>
#include <QStandardItemModel>

#include "download.h"
#include "addstudent.h"
#include "deletestudent.h"
#include "alertstudent.h"

namespace Ui {
class StudentsList;
}

class StudentsList : public QWidget
{
    Q_OBJECT

signals:
    void firstWindow();
    void getGroup(QString comboGroup);
    void postDelStud(QStringList deleteStud);
```

```
    void postAlertStud(QStringList alertStud);

public:
    explicit StudentsList(QWidget *parent = nullptr);
    ~StudentsList();
    QString comboGroup;
    QStringList deleteStud;
    QStringList alertStud;

public slots:
    void readGroup();
    void readStud();
    void numGroup();

private slots:
    void on_back_clicked();
    void on_add_clicked();
    void on_del_clicked();

    void on_alert_clicked();

private:
    Ui::StudentsList *ui;
    Download *download;
    AddStudent *add;
    DeleteStudent *confirm;
    QStandardItemModel *tableStud;
    AlertStudent *alert;
};

#endif // STUDENTSLIST_H
```

6.2 Реализация класса (studentslist.cpp)

```
#include "studentslist.h"
#include "ui_studentslist.h"
#include "download.h"
#include "addstudent.h"
#include "deletestudent.h"

#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>
```

```
StudentsList::StudentsList(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::StudentsList)
{
    ui->setupUi(this);

    add = new AddStudent ();
    connect(add, &AddStudent::addString, this, &AddStudent::show);

    alert = new AlertStudent ();
    connect(this, SIGNAL(postAlertStud(QStringList)), alert,
SLOT(studentAlert(QStringList)));
    connect(alert, &AlertStudent::alertString, this, &AlertStudent::show);

    confirm = new DeleteStudent ();
    connect(this, SIGNAL(postDelStud(QStringList)), confirm,
SLOT(studentDelete(QStringList)));
    connect(confirm, &DeleteStudent::deleteString, this, &DeleteStudent::show);

    download = new Download (); // Инициируем новый объект загрузки
    connect(this, SIGNAL(getGroup(QString)), download,
SLOT(getDataStud(QString)));
    connect(ui->select, SIGNAL(clicked()), this, SLOT(numGroup()));  // получаем
данные по нажатию на кнопку
    connect(download, &Download::onReady, this, &StudentsList::readStud);    //
вызываем функцию чтения файла

    tableStud = new QStandardItemModel (ui->tableStud);   // Инициализируем
таблицу
}

StudentsList::~StudentsList()
{
    delete ui;
}

void StudentsList::readGroup()
{
    tableStud->clear();
    ui->comboBox->clear();
    ui->comboBox->addItem("Все группы");

    QFile questGroup(QCoreApplication::applicationDirPath() + "/group.json");
```

```
    if (!questGroup.open(QIODevice::ReadOnly))
        return;

    QByteArray dataG = questGroup.readAll();   // Парсинг файла с номерами групп
    QJsonDocument documentG = documentG.fromJson(dataG);
    QJsonObject root = documentG.object();
    root.value("data");
    QJsonValue groups = root.value("data");
    if(groups.isArray())
    {
        QJsonArray group = groups.toArray();
        for (int i = 0; i < group.count(); i++) {   // Работа с выпадающим списком
            ui->comboBox->addItem(group[i].toString());
        }
    }
}

void StudentsList::readStud()
{
    tableStud->clear();
    QFile questStud(QCoreApplication::applicationDirPath() + "/student.json");
    if (!questStud.open(QIODevice::ReadOnly))
        return;
    QByteArray dataS = questStud.readAll();   // Парсинг файла со студентами
    QJsonDocument documentS = documentS.fromJson(dataS);
    QJsonObject root = documentS.object();
    root.value("data");
    QJsonValue students = root.value("data");
    if(students.isArray())
    {
        QJsonArray student = students.toArray();

        tableStud->setHorizontalHeaderLabels(QStringList() << "id" << "Фамилия" <<
"Имя" << "Отчество" << "Номер группы");
        for (int i = 0; i < student.count(); i++) {
            QJsonObject personal = student.at(i).toObject();

            QList<QStandardItem *> items;
            items.append(new
QStandardItem(QString::number(personal.value("id").toInt())));    // Скрытая
колонка
            items.append(new QStandardItem(personal.value("lName").toString()));    //
Вторая колонка
```

```
            items.append(new QStandardItem(personal.value("fName").toString()));    //
Первая колонка
            items.append(new QStandardItem(personal.value("mName").toString()));    //
Третья колонка
            items.append(new QStandardItem(personal.value("group").toString()));    //
Четвертая колонка
            tableStud->appendRow(items);
        }
    }
    ui->tableStud->resizeColumnsToContents();
    ui->tableStud->setModel(tableStud);   // Устанавливаем модель для таблицы
    ui->tableStud->setColumnHidden(0, true);
}

void StudentsList::numGroup()
{
    comboGroup = ui->comboBox->currentText();
    emit getGroup(comboGroup);
}

void StudentsList::on_back_clicked()
{
    this->close();
    emit firstWindow();
}

void StudentsList::on_add_clicked()
{
    add->show();
}

void StudentsList::on_del_clicked()
{
    // Получаем данные из выделенной строки, которые будут удалены
    QString delStud;
    for (int i = 0; i < 5; i++) {
        delStud = tableStud->data(tableStud->index(ui->tableStud-
>currentIndex().row(), i)).toString();
        deleteStud.push_back(delStud);
    }
    emit postDelStud(deleteStud);
    confirm->show();
}
```

```
void StudentsList::on_alert_clicked()
{
   // Ролучаем измененные данные из текущей строки
   QString alStud;
   for (int i = 0; i < 5; i++) {
      alStud = tableStud->data(tableStud->index(ui->tableStud->currentIndex().row(),
i)).toString();
      alertStud.push_front(alStud);
   }
   emit postAlertStud(alertStud);
   alert->show();
}
```

7. Код добавления нового студента

7.1 Заголовочный файл (addstudent.h)

```
#ifndef ADDSTUDENT_H
#define ADDSTUDENT_H

#include <QWidget>

#include "download.h"

namespace Ui {
class AddStudent;
}

class AddStudent : public QWidget
{
   Q_OBJECT

signals:
   void addString();

public:
   explicit AddStudent(QWidget *parent = nullptr);
   ~AddStudent();
   QString lName;
   QString fName;
   QString mName;
   QString group;

public slots:
```

```
    void readyRequest(QNetworkReply*);

private slots:
    void on_cancel_clicked();
    void on_save_clicked();

private:
    Ui::AddStudent *ui;
};

#endif // ADDSTUDENT_H
```

Реализация класса (addstudent.cpp)

```
#include "addstudent.h"
#include "ui_addstudent.h"

#include <QtNetwork/QNetworkAccessManager>
#include <QtNetwork/QNetworkReply>
#include <QtNetwork/QNetworkRequest>

#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>

AddStudent::AddStudent(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::AddStudent)
{
    ui->setupUi(this);
}

AddStudent::~AddStudent()
{
    delete ui;
}

void AddStudent::on_cancel_clicked()
{
    this->close();
    emit addString();
}

void AddStudent::readyRequest(QNetworkReply* reply)
```

```
{
    if (reply->error() == QNetworkReply::NoError)
    {
        qDebug() << "File post";
        reply->deleteLater();
    }
    else
    {
        qDebug() << reply->errorString();
    }
}

void AddStudent::on_save_clicked()
{
    ui->lName->clear();
    ui->fName->clear();
    ui->mName->clear();
    ui->numGroup->clear();
    lName = ui->lName->text();  // Получаем данные из формы
    fName = ui->fName->text();
    mName = ui->mName->text();
    group = ui->numGroup->text();

    QJsonObject main;
    QJsonArray student;
    QJsonObject data;

    main.insert("action", QJsonValue::fromVariant("add"));

    data.insert("lName", QJsonValue::fromVariant(lName));
    data.insert("fName", QJsonValue::fromVariant(fName));
    data.insert("mName", QJsonValue::fromVariant(mName));
    data.insert("group", QJsonValue::fromVariant(group));
    student.push_back(data);
    main.insert("data", student);
    QJsonDocument file(main);
    QString jsonString = file.toJson(QJsonDocument::Indented);
    //Записываем данные в файл
    QFile json;
    json.setFileName(QCoreApplication::applicationDirPath() + "/addStudent.json");
    json.open(QIODevice::WriteOnly | QIODevice::Text);
    QTextStream stream( &json );
    stream << jsonString;
```

```
    json.close();
//    POST-запрос/отправка файла
    QFile *askNewStud = new QFile (QCoreApplication::applicationDirPath() +
"/addStudent.json");
    if(askNewStud->open(QFile::ReadOnly))
    {
        QByteArray data = askNewStud->readAll();
        QJsonDocument file = file.fromJson(data);

        QUrl resource("http://192.168.0.10/groups/all");
        QNetworkAccessManager* postNewStud = new
QNetworkAccessManager(this);
        connect(postNewStud, SIGNAL(finished(QNetworkReply*)), this,
SLOT(readyRequest(QNetworkReply*)));

        QNetworkRequest request(resource);
        request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");

        postNewStud->post(request, data);
    }

    this->close();
    emit addString();
}
```

8. Код изменения данных студента

8.1 Заголовочный файл (alertstudent.h)

```
#ifndef ALERTSTUDENT_H
#define ALERTSTUDENT_H

#include <QWidget>

#include "download.h"

namespace Ui {
class AlertStudent;
}

class AlertStudent : public QWidget
{
    Q_OBJECT

signals:
```

```
    void alertString();

public:
    explicit AlertStudent(QWidget *parent = nullptr);
    ~AlertStudent();
    QStringList alertStud;

public slots:
    void readyRequest(QNetworkReply*);
    void studentAlert(QStringList alertStud);

private slots:
    void on_cancel_clicked();
    void on_alert_clicked();

private:
    Ui::AlertStudent *ui;
};
#endif // ALERTSTUDENT_H
```

8.2 Реализация класса (alertstudent.cpp)

```
#include "alertstudent.h"
#include "ui_alertstudent.h"
#include <QtNetwork/QNetworkAccessManager>
#include <QtNetwork/QNetworkReply>
#include <QtNetwork/QNetworkRequest>

#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>

AlertStudent::AlertStudent(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::AlertStudent)
{
    ui->setupUi(this);
    setWindowTitle("Подтверждение изменений");
}

AlertStudent::~AlertStudent()
{
    delete ui;
}
```

```
void AlertStudent::on_cancel_clicked()
{
   this->close();
   emit alertString();
}

void AlertStudent::readyRequest(QNetworkReply* reply)
{
   if (reply->error() == QNetworkReply::NoError)   // Проверяем, что запрос
завершился без ошибок
   {
      qDebug() << "File post";
      reply->deleteLater();
   }
   else
   {
      qDebug() << reply->errorString();
   }
}

void AlertStudent::studentAlert(QStringList alertStud)
{
   QJsonObject main;
   QJsonArray student;
   QJsonObject data;
   // Добавляем в файл измененные данные о студенте
   main.insert("action", QJsonValue::fromVariant("alter"));

   int id = alertStud.at(4).toInt();
   data.insert("id", QJsonValue::fromVariant(id));
   data.insert("lName", QJsonValue::fromVariant(alertStud.at(3)));
   data.insert("fName", QJsonValue::fromVariant(alertStud.at(2)));
   data.insert("mName", QJsonValue::fromVariant(alertStud.at(1)));
   data.insert("group", QJsonValue::fromVariant(alertStud.at(0)));
   student.push_front(data);
   main.insert("data", student);

   QJsonDocument file(main);

   QString jsonString = file.toJson(QJsonDocument::Indented);
   //Записываем данные в файл
   QFile json;
```

```
    json.setFileName(QCoreApplication::applicationDirPath() + "/alertStudent.json");
    json.open(QIODevice::WriteOnly | QIODevice::Text);
    QTextStream stream( &json );
    stream << jsonString;
    json.close();
}

void AlertStudent::on_alert_clicked()
{
    // Отправляем данные об изменении информации о студенте
    QFile *askAlertStud = new QFile (QCoreApplication::applicationDirPath() +
"/alertStudent.json");
    if(askAlertStud->open(QFile::ReadOnly))
    {
        QByteArray data = askAlertStud->readAll();
        QJsonDocument file = file.fromJson(data);

        QUrl resource("http://192.168.0.10/groups/all");
        QNetworkAccessManager* postAlertStud = new
QNetworkAccessManager(this);
        connect(postAlertStud, SIGNAL(finished(QNetworkReply*)), this,
SLOT(readyRequest(QNetworkReply*)));

        QNetworkRequest request(resource);
        request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");

        postAlertStud->post(request, data);
    }
    this->close();
    emit alertString();
}
```

9. Код добавления нового студента

9.1 Заголовочный файл (deletestudent.h)

```
#ifndef DELETESTUDENT_H
#define DELETESTUDENT_H

#include <QWidget>
#include "download.h"
namespace Ui {
class DeleteStudent;
}

class DeleteStudent : public QWidget
```

```
{
    Q_OBJECT

signals:
    void deleteString();

public:
    explicit DeleteStudent(QWidget *parent = nullptr);
    ~DeleteStudent();
    QStringList deleteStud;

public slots:
    void readyRequest(QNetworkReply*);
    void studentDelete(QStringList deleteStud);

private slots:
    void on_cancel_clicked();
    void on_del_clicked();

private:
    Ui::DeleteStudent *ui;
};

#endif // DELETESTUDENT_H
```

9.2 Реализация класса (deletestudent.cpp)

```
#include "deletestudent.h"
#include "ui_deletestudent.h"

#include <QtNetwork/QNetworkAccessManager>
#include <QtNetwork/QNetworkReply>
#include <QtNetwork/QNetworkRequest>

#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>

DeleteStudent::DeleteStudent(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::DeleteStudent)
{
    ui->setupUi(this);
    setWindowTitle("Подтверждение удаления");
```

```
}

DeleteStudent::~DeleteStudent()
{
   delete ui;
}

void DeleteStudent::on_cancel_clicked()
{
   this->close();
   emit deleteString();
}

void DeleteStudent::readyRequest(QNetworkReply* reply)
{
   if (reply->error() == QNetworkReply::NoError)   // Проверяем, что запрос
завершился без ошибок
   {
      qDebug() << "File post";
      reply->deleteLater();
   }
   else
   {
      qDebug() << reply->errorString();
   }
}

void DeleteStudent::studentDelete(QStringList deleteStud)
{
   QJsonObject main;
   QJsonArray student;
   QJsonObject data;
   // Записываем данные студента для операции удаления
   main.insert("action", QJsonValue::fromVariant("remove"));

   int id = deleteStud.at(0).toInt();
   data.insert("fName", QJsonValue::fromVariant(deleteStud.at(1)));
   data.insert("group", QJsonValue::fromVariant(deleteStud.at(4)));
   data.insert("id", QJsonValue::fromVariant(id));
   data.insert("lName", QJsonValue::fromVariant(deleteStud.at(2)));
   data.insert("mName", QJsonValue::fromVariant(deleteStud.at(3)));
   student.push_front(data);
   main.insert("data", student);
```

```
    QJsonDocument file(main);
    QString jsonString = file.toJson(QJsonDocument::Indented);

    //Записываем данные в файл
    QFile json;
    json.setFileName(QCoreApplication::applicationDirPath() +
"/deleteStudent.json");
    json.open(QIODevice::WriteOnly | QIODevice::Text);
    QTextStream stream( &json );
    stream << jsonString;
    json.close();
}
void DeleteStudent::on_del_clicked()
{
    // Отправка файла с данными студента, которого нужно удалить
    QFile *askDelStud = new QFile (QCoreApplication::applicationDirPath() +
"/deleteStudent.json");
    if(askDelStud->open(QFile::ReadOnly))
    {
        QByteArray data = askDelStud->readAll();
        QJsonDocument file = file.fromJson(data);
        QUrl resource("http://192.168.0.10/groups/all");
        QNetworkAccessManager* postDelStud = new QNetworkAccessManager(this);
        connect(postDelStud, SIGNAL(finished(QNetworkReply*)), this,
SLOT(readyRequest(QNetworkReply*)));
        QNetworkRequest request(resource);
        request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");
        postDelStud->post(request, data);
    }
    this->close();
    emit deleteString();
}
```