Сборка сервиса

mkdir build

cd build

cmake ..

cmake --build .

База данных

В качестве базы данных используется Sqlite.

База данных содержит таблицу attendance.

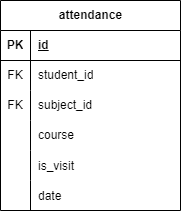


Рисунок 1. – Таблица “attendance”

Колонки таблицы attendance:

* id – идентификатор строки;
* student\_id – идентификатор студента;
* subject\_id – идентификатор учебной дисциплины;
* course – номер курса;
* is\_visit – отметка посещаемости (true / false);
* date – дата проведения занятия.

API

1. POST /api/attendance/one – создание одной записи посещаемости.

Пример тела запроса:

{

    "studetnt\_id": 1,

    "subjeцct\_id": 1,

    "course": 1,

    "is\_visit": **false**,

    "date": "2024-11-22"

}

1. POST /api/attendance/list – создание списка записей посещаемости.

Пример тела запроса:

[

    {

        "student\_id": 2,

        "subject\_id": 2,

        "course": 2,

        "is\_visit": **true**,

        "date": "2024-11-22"

    },

    {

        "student\_id": 3,

        "subject\_id": 2,

        "course": 2,

        "is\_visit": **true**,

        "date": "2024-11-22"

    },

    {

        "student\_id": 4,

        "subject\_id": 2,

        "course": 2,

        "is\_visit": **false**,

        "date": "2024-11-22"

    }

]

1. PUT /api/attendance/update – обновить посещаемость

Пример тела запроса:

{

    "id": 1,

    "studetnt\_id": 1,

    "subjeцct\_id": 1,

    "course": 1,

    "is\_visit": **false**,

    "date": "2024-11-22"

}

1. DELETE /api/attendance/delete – удалить посещаемость

Пример тела запроса:

{

    "id": 1,

}

1. GET /api/attendance/all – получение списка посещаемости

Пример тела запроса:

       {

        "student\_id": 3,

        "subject\_id": 3,

        "course": 3,

        "is\_visit": **true**,

        "date": "2024-11-22"

    }

Поля теля запроса являются необязательными и используются для фильтрации.

Например, для посещаемости одного студента следует использовать фильтр

{

        "student\_id": 10,

    }

Ответ содержит список студентов. Пример:

[

    {

        "id": 1,

        "student\_id": 2,

        "subject\_id": 2,

        "course": 2,

        "is\_visit": **true**,

        "date": "2024-11-22"

    },

    {

        "id": 2,

        "student\_id": 3,

        "subject\_id": 2,

        "course": 2,

        "is\_visit": **true**,

        "date": "2024-11-22"

    },

    {

        "id": 3,

        "student\_id": 4,

        "subject\_id": 2,

        "course": 2,

        "is\_visit": **false**,

        "date": "2024-11-22"

    }

]

1. GET /api/attendance/count/visit – получение количества посещенных занятий

Пример тела запроса:

{

    "studetnt\_id": 1,

    "subjeцct\_id": 1,

    "course": 1

}

1. GET /api/attendance/count/absence – получение количества отсутствий на занятиях

Пример тела запроса:

{

    "studetnt\_id": 1,

    "subjeцct\_id": 1,

    "course": 1

}

1. DELETE /api/attendance/delete/subject – удалить всю посещаемость для заданного предмета

Пример тела запроса:

{

    "subjeцct\_id": 1,

}

1. GET /api/attendance/delete/student – удалить всю посещаемость для заданного студента

Пример тела запроса:

{

    "studetnt\_id": 1,

}

**Структуры**

Тип данных для идентификатора посещаемости:

typedef long EOP\_Attendance\_id;

Тип данных для идентификатора студента:  
typedef long EOP\_Attendance\_student\_id;

Тип данных для идентификатора предмета:  
typedef long EOP\_Attendance\_subject\_id;

Тип данных для номера курса:  
typedef int EOP\_Attendance\_course;

Тип данных для отметки посещения:  
typedef **bool** EOP\_Attendance\_is\_visit;

Тип данных для даты посещения:  
typedef const char \*EOP\_Attendance\_date;

Структура посещаемости:  
typedef struct {  
 EOP\_Attendance\_id id;  
 EOP\_Attendance\_student\_id student\_id;  
 EOP\_Attendance\_subject\_id subject\_id;  
 EOP\_Attendance\_course course;  
 EOP\_Attendance\_is\_visit is\_visit;  
 EOP\_Attendance\_date date;  
} EOP\_Attendance;

Структура запроса получения количества посещений / отсутствий:  
typedef struct {  
 EOP\_Attendance\_student\_id student\_id;  
 EOP\_Attendance\_subject\_id subject\_id;  
 EOP\_Attendance\_course course;  
} EOP\_Attendance\_Count\_request;

Структура запроса удаления посещаемости:  
typedef struct {  
 EOP\_Attendance\_id id;  
} EOP\_Attendance\_Delete\_attendance\_request;

Структура запроса удаления всей посещаемости для студента:  
typedef struct {  
 EOP\_Attendance\_id student\_id;  
} EOP\_Attendance\_Delete\_student\_attendance\_request;

Структура запроса удаления всей посещаемости предмета:  
typedef struct {  
 EOP\_Attendance\_subject\_id subject\_id;  
} EOP\_Attendance\_Delete\_subject\_attendance\_request;

**Диаграмма классов**

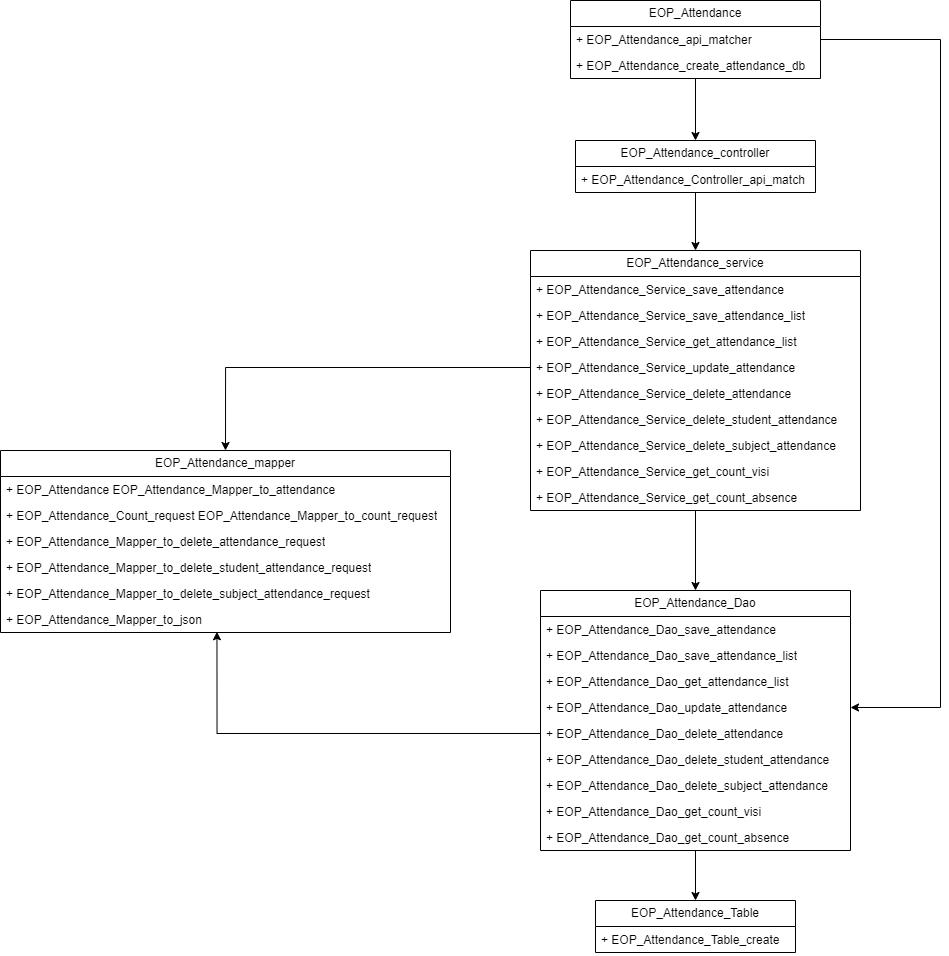


Рисунок 1. – Диаграмма классов

**Доступность**

1. Необходимо обеспечить мониторинг работоспособности и доступности системы в течение 23ч в сутки: настройка оповещений команды разработки о возникающих ошибках работы системы.
2. В случае отключения системы должен быть предусмотрен автоматический перезапуск системы
3. Система должна быть доступна как минимум 98% времени.
4. Плановое обслуживание по резервному копированию данных не должно влиять на доступность системы.
5. В случае проведения плановых работ предусмотрено уведомление пользователей.
6. Максимальное время ожидания ответа равно 30 секундам, по истечению времени пользователю должен быть возвращен ответ с кодом 500 Timeout.

**Целостность**

1. Должна быть реализована поддержка транзакционных операций по записи, обновлению и удалению данных в системе.
2. База данных должна содержать ограничения уникальности.
3. Каждый день с 03:00-04:00 автоматически должно производиться резервное копирование базы данных с дальнейшим сохранением копии в двух хранилищах, расположенных в разных датацентрах.

**Совместимость**

1. Взаимодействие клиента с системой должно производиться с использованием http/https протокола.
2. Прием и отправка ответа от сервера производится в json формате.
3. Версия базы данных SQLite – 3.45.3.
4. Хранение данных в БД происходит в кодировке UTF-8.

**Производительность**

1. Среднее время получения информации о выборке посещаемости для одного студента за все время - 500 ms. Максимальное время ожидания - 5 секунд.
2. Среднее время получения информации о выборке посещаемости всех студентов для одного предмета - 400 ms. Максимальное время ожидания - 5 секунд.

**Надежность**

1. Критическая неисправность системы должна быть исправлена в течение 12 часов с момента обнаружения неисправности: сервис не реагирует на запросы, некорректное сохранение посещаемости.
2. Неисправность среднего уровня должна быть исправлена в течение 5 рабочих дней: некорректное получение посещаемости.
3. Неисправность низкого уровня должна быть исправлена в течение 20 рабочих дней: остальные факторы, не влияющие на сохранение и получение посещаемости.
4. Среднее время восстановления не должно превышать (сумма времени простоя за период, деленная на количество инцидентов) 1 час в день.
5. Средняя наработка на отказ (чем больше времени между отказами, тем надежнее система. Общее время работы, деленное на количество сбоев) не должна быть меньше 23 часов - 1 сбой в день длительностью 1 час.

**Устойчивость**

1. Система должна оповещать пользователей о невозможности работы с системой при ошибках соединения с БД или сторонними системами.

**Удобство использования**

1. Система должна предоставлять описание каждого метода API с телом запроса и ответа.
2. Содержать комментарии к столбцам таблиц и таблицам в базе данных.

**Безопасность**

1. В системе должна быть предусмотрена защита от SQL-инъекций.

**Защита**

1. Должна быть реализована защита от DDoS атак: пользователь может делать не более 3 запросов в секунду на получение сведений из сервиса от одного ip адреса.

**Эффективность**

В системе должно быть предусмотрено кеширование обратной связи из бд на срок до 5 минут или при устаревании данных (произошло обновление обратной связи).

**Возможность модификации**

1. Архитектура системы должна быть представлена **трехуровневой архитектурой**. На программном уровне должны быть выделены слои: взаимодействие с БД, слой бизнес-логики, слой представления данных.
2. Внесение изменений в базы данных должно проводиться с использованием миграций для возможности приведения хранилища в предыдущее состояние.

**Переносимость**

1. В новой системе должны быть установлены следующие программы: cmake, git
2. Система должна предоставлять возможность сборки docker-образа для его дальнейшего запуска на сервере.

**Масштабируемость**

1. Система должна предусматривать возможность балансировки нагрузки при обращении к сервисам, которые предоставляют информацию о посещаемости.