МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ

КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ордена Трудового Красного Знамени федеральное государственное

бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Информатика»

Лабораторная работа №6

**«Технологии доступа к данным»**

по дисциплине

**Информационные технологии и программирование**

|  |  |
| --- | --- |
|  | Выполнил: студент гр. БЭИ2202  Бояркин Е. В.  Вариант №3  Проверил: зав. кафедры «Информатика» Халабия Р. Ф. |

Москва, 2024 г.

Содержание

[1 Постановка задачи 4](#_Toc184167824)

[2 Диаграмма классов 5](#_Toc184167825)

[3 Алгоритмы 6](#_Toc184167826)

[3.1 Функция get\_valid\_int 6](#_Toc184167827)

[3.2 Функция get\_valid\_string 6](#_Toc184167828)

[3.3 Функция get\_valid\_date 7](#_Toc184167829)

[Структура Group 7](#_Toc184167830)

[3.4 Метод to\_json 7](#_Toc184167831)

[3.5 Метод from\_json 8](#_Toc184167832)

[Структура Dormitory 8](#_Toc184167833)

[3.6 Метод to\_json 8](#_Toc184167834)

[3.7 Метод from\_json 8](#_Toc184167835)

[Структура Room 9](#_Toc184167836)

[3.8 Метод to\_json 9](#_Toc184167837)

[3.9 Метод from\_json 9](#_Toc184167838)

[Структура Student 9](#_Toc184167839)

[3.10 Метод to\_json 9](#_Toc184167840)

[3.11 Метод from\_json 9](#_Toc184167841)

[Структура AccommodationInfo 10](#_Toc184167842)

[3.12 Метод to\_json 10](#_Toc184167843)

[3.13 Метод from\_json 10](#_Toc184167844)

[Класс Database 10](#_Toc184167845)

[3.14 Конструктор Database(string fname) 10](#_Toc184167846)

[3.15 Метод save() 11](#_Toc184167847)

[3.16 Метод is\_group\_code\_unique(const int code) 11](#_Toc184167848)

[3.17 Метод group\_exists(const int code) 11](#_Toc184167849)

[3.18 Метод add\_group(const Group& group) 12](#_Toc184167850)

[3.19 Метод delete\_group(int code) 12](#_Toc184167851)

[3.20 Метод edit\_group(const int code, const Group& new\_group) 12](#_Toc184167852)

[3.21 Аналогичные методы для Dormitory, Room, Student и AccommodationInfo 13](#_Toc184167853)

[3.22 Метод display\_groups() 13](#_Toc184167854)

[3.23 Методы display\_dormitories(), display\_rooms(), display\_students(), display\_accommodations() 14](#_Toc184167855)

[3.24 Функция menu\_groups(Database& db) 14](#_Toc184167856)

[3.25 Функция menu\_dormitories(Database& db) 15](#_Toc184167857)

[3.26 Функция menu\_rooms(Database& db) 15](#_Toc184167858)

[3.27 Функция menu\_students(Database& db) 16](#_Toc184167859)

[3.28 Функция menu\_accommodations(Database& db) 16](#_Toc184167860)

[3.29 Функция main\_menu(Database& db) 16](#_Toc184167861)

[3.30 Основная функция main() 17](#_Toc184167862)

[4 Листинг программы 21](#_Toc184167863)

[5 Результаты работы программы 42](#_Toc184167864)

[6 Вывод 45](#_Toc184167865)

# 1 Постановка задачи

Изучить и освоить механизмы доступа и передачи данных (XML и JSON).

Задание:

Написать программу согласно варианту, которая добавляет, удаляет и изменяет записи данных, хранящиеся определенном формате в файле заданного типа. Если файла нет, то необходимо его создавать, если все записи в файле удалены, то файл должен оставаться пустым. Также мы должны выводить информацию из файла на экран в табличном виде. Для редактирования данных необходимо небольшое простое меню. Таблицы организовать внутри программы как классы. И главное таблицы связанны между собой.

| Таблица 1 — Вариант задания | | |
| --- | --- | --- |
| Вариант | Структура данных | Формат файла хранения |
| 3 | «Учёт студентов, проживающих в общежитиях» Информация должна хранится в следующей cтруктуре:   * группа (код группы, название группы, название факультета); * общежитие (код общежития, название, адрес); * комната (код комнаты, № комнаты, код общежития); * студент (код студента, ФИО студента, дата рождения, пол, код группы, серия паспорта, номер паспорта, кем и когда выдан); * информация о заселении (код комнаты, код студента, дата заселения, дата выселения). | JSON |

# 2 Диаграмма классов

Диаграмма классов представлена на рисунке 1.

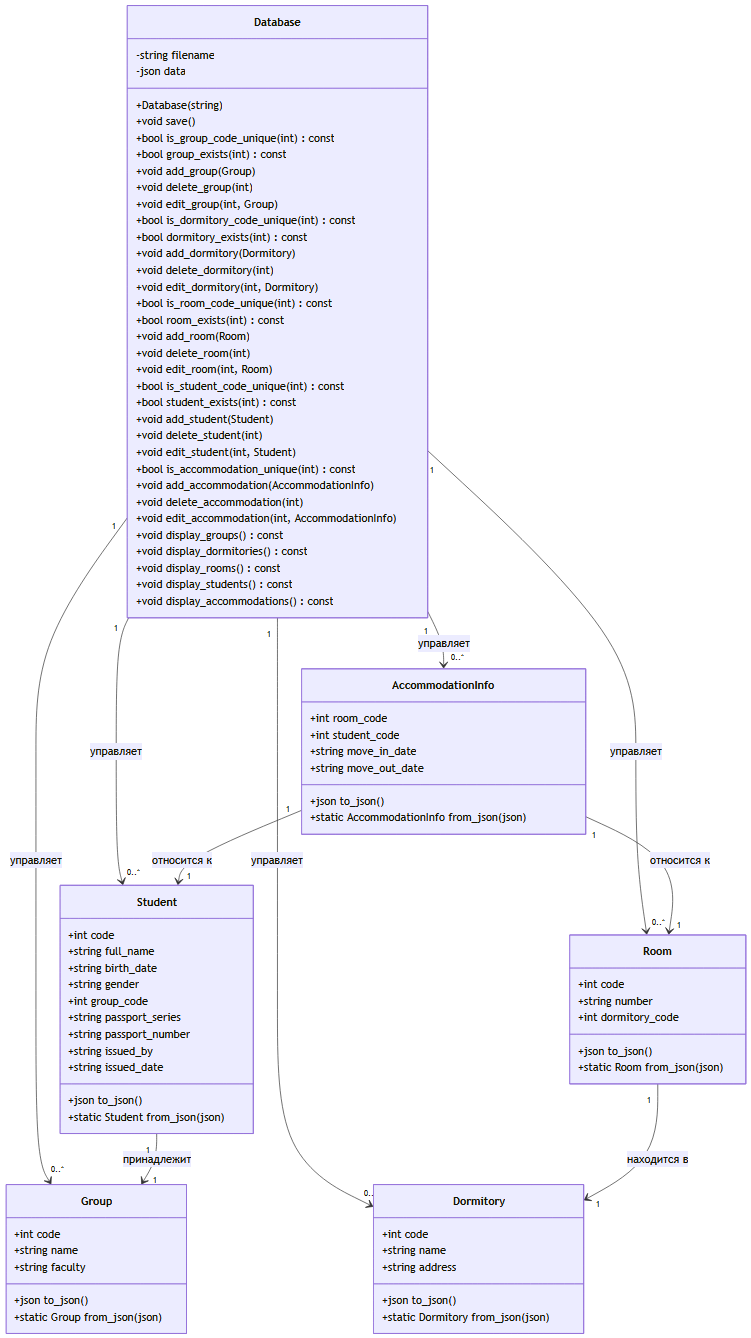


Рисунок 1 — Диаграмма классов

# 3 Алгоритмы

## 3.1 Функция get\_valid\_int

Шаг 1. Запрос ввода числа

Шаг 1.1. Пользователю выводится сообщение с просьбой ввести целое число.

Шаг 2. Обработка ввода

Шаг 2.1. Программа ожидает ввода пользователя.

Шаг 2.2. Введённое значение пытается преобразоваться в целое число.

Шаг 3. Проверка корректности

Шаг 3.1. Если преобразование не удалось, выводится сообщение об ошибке.

Шаг 3.1.1. Ввод очищается от некорректных данных.

Шаг 3.1.2. Пользователю предлагается повторить ввод.

Шаг 3.2. Если преобразование успешно, ввод завершается.

Шаг 4. Возврат результата

Шаг 4.1. Функция возвращает валидное целое число, введённое пользователем.

## 3.2 Функция get\_valid\_string

Шаг 1. Запрос ввода строки

Шаг 1.1. Пользователю выводится сообщение с просьбой ввести строку.

Шаг 2. Обработка ввода

Шаг 2.1. Программа ожидает ввода пользователя и считывает всю введённую строку.

Шаг 3. Проверка корректности

Шаг 3.1. Если введённая строка пуста, выводится сообщение об ошибке.

Шаг 3.2. Пользователю предлагается повторить ввод.

Шаг 4. Возврат результата

Шаг 4.1. Если строка не пуста, функция возвращает введённую строку.

## 3.3 Функция get\_valid\_date

Шаг 1. Запрос ввода даты

Шаг 1.1. Пользователю выводится сообщение с просьбой ввести дату в формате ДД.ММ.ГГГГ.

Шаг 2. Обработка ввода

Шаг 2.1. Программа ожидает ввода пользователя и считывает введённую строку.

Шаг 3. Проверка формата даты

Шаг 3.1. Проверяется соответствие введённой строки заданному регулярному выражению для даты.

Шаг 3.1.1. Если формат некорректен, выводится сообщение об ошибке.

Шаг 3.1.2. Пользователю предлагается повторить ввод.

Шаг 4. Проверка валидности даты

Шаг 4.1. Из введённой строки извлекаются день и месяц.

Шаг 4.1.1. Проверяется, что день находится в диапазоне от 1 до 31.

Шаг 4.1.2. Проверяется, что месяц находится в диапазоне от 1 до 12.

Шаг 4.2. Если день или месяц некорректны, выводится сообщение об ошибке.

Шаг 4.2.1. Пользователю предлагается повторить ввод.

Шаг 5. Возврат результата

Шаг 5.1. Если все проверки пройдены успешно, функция возвращает введённую дату.

## Структура Group

## 3.4 Метод to\_json

Шаг 1. Создание JSON-объекта

Шаг 1.1. Формируется JSON-объект, содержащий код, название и факультет группы.

Шаг 2. Возврат JSON-объекта

Шаг 2.1. Метод возвращает сформированный JSON-объект.

## 3.5 Метод from\_json

Шаг 1. Создание объекта группы

Шаг 1.1. Создаётся новый объект группы.

Шаг 2. Заполнение данных из JSON

Шаг 2.1. Из JSON-объекта извлекаются значения кода, названия и факультета.

Шаг 2.2. Эти значения присваиваются соответствующим полям объекта группы.

Шаг 3. Возврат объекта группы

Шаг 3.1. Метод возвращает заполненный объект группы.

## Структура Dormitory

## 3.6 Метод to\_json

Шаг 1. Создание JSON-объекта

Шаг 1.1. Формируется JSON-объект, содержащий код, название и адрес общежития.

Шаг 2. Возврат JSON-объекта

Шаг 2.1. Метод возвращает сформированный JSON-объект.

## 3.7 Метод from\_json

Шаг 1. Создание объекта общежития

Шаг 1.1. Создаётся новый объект общежития.

Шаг 2. Заполнение данных из JSON

Шаг 2.1. Из JSON-объекта извлекаются значения кода, названия и адреса.

Шаг 2.2. Эти значения присваиваются соответствующим полям объекта общежития.

Шаг 3. Возврат объекта общежития

Шаг 3.1. Метод возвращает заполненный объект общежития.

## Структура Room

## 3.8 Метод to\_json

Шаг 1. Создание JSON-объекта

Шаг 1.1. Формируется JSON-объект, содержащий код, номер комнаты и код общежития.

Шаг 2. Возврат JSON-объекта

Шаг 2.1. Метод возвращает сформированный JSON-объект.

## 3.9 Метод from\_json

Шаг 1. Создание объекта комнаты

Шаг 1.1. Создаётся новый объект комнаты.

Шаг 2. Заполнение данных из JSON

Шаг 2.1. Из JSON-объекта извлекаются значения кода, номера и кода общежития.

Шаг 2.2. Эти значения присваиваются соответствующим полям объекта комнаты.

Шаг 3. Возврат объекта комнаты

Шаг 3.1. Метод возвращает заполненный объект комнаты.

## Структура Student

## 3.10 Метод to\_json

Шаг 1. Создание JSON-объекта

Шаг 1.1. Формируется JSON-объект, содержащий код, полное имя, дату рождения, пол, код группы, паспортные данные и информацию о выдаче паспорта.

Шаг 2. Возврат JSON-объекта

Шаг 2.1. Метод возвращает сформированный JSON-объект.

## 3.11 Метод from\_json

Шаг 1. Создание объекта студента

Шаг 1.1. Создаётся новый объект студента.

Шаг 2. Заполнение данных из JSON

Шаг 2.1. Из JSON-объекта извлекаются значения всех полей студента.

Шаг 2.2. Эти значения присваиваются соответствующим полям объекта студента.

Шаг 3. Возврат объекта студента

Шаг 3.1. Метод возвращает заполненный объект студента.

## Структура AccommodationInfo

## 3.12 Метод to\_json

Шаг 1. Создание JSON-объекта

Шаг 1.1. Формируется JSON-объект, содержащий код комнаты, код студента, дату заселения и дату выселения.

Шаг 2. Возврат JSON-объекта

Шаг 2.1. Метод возвращает сформированный JSON-объект.

## 3.13 Метод from\_json

Шаг 1. Создание объекта информации о заселении

Шаг 1.1. Создаётся новый объект информации о заселении.

Шаг 2. Заполнение данных из JSON

Шаг 2.1. Из JSON-объекта извлекаются значения кодов комнаты и студента, а также даты заселения и выселения.

Шаг 2.2. Эти значения присваиваются соответствующим полям объекта информации о заселении.

Шаг 3. Возврат объекта информации о заселении

Шаг 3.1. Метод возвращает заполненный объект информации о заселении.

## Класс Database

## 3.14 Конструктор Database(string fname)

Шаг 1. Инициализация

Шаг 1.1. Сохраняется имя файла для хранения данных.

Шаг 2. Загрузка данных из файла

Шаг 2.1. Открывается файл с указанным именем.

Шаг 2.2. Проверяется, существует ли файл и доступен ли для чтения.

Шаг 2.2.1. Если файл существует, данные загружаются из него в структуру данных.

Шаг 2.2.2. Если файл не существует, создаётся новая структура данных с пустыми списками для всех категорий (группы, общежития, комнаты, студенты, информация о заселении).

Шаг 3. Сохранение данных

Шаг 3.1. Если файл не существовал, сохранённая структура данных записывается в новый файл.

## 3.15 Метод save()

Шаг 1. Открытие файла для записи

Шаг 1.1. Файл с указанным именем открывается для записи.

Шаг 2. Запись данных

Шаг 2.1. Текущие данные форматируются с отступами для удобочитаемости.

Шаг 2.2. Форматированные данные записываются в файл.

Шаг 3. Закрытие файла

Шаг 3.1. Файл закрывается, завершая процесс сохранения.

## 3.16 Метод is\_group\_code\_unique(const int code)

Шаг 1. Проверка уникальности кода группы

Шаг 1.1. Проходит проверка, что ни одна из существующих групп не имеет указанного кода.

Шаг 2. Возврат результата

Шаг 2.1. Если код уникален, возвращается true.

Шаг 2.2. В противном случае, возвращается false.

## 3.17 Метод group\_exists(const int code)

Шаг 1. Проверка существования группы

Шаг 1.1. Проходит проверка, что хотя бы одна группа имеет указанный код.

Шаг 2. Возврат результата

Шаг 2.1. Если группа существует, возвращается true.

Шаг 2.2. В противном случае, возвращается false.

## 3.18 Метод add\_group(const Group& group)

Шаг 1. Проверка уникальности кода

Шаг 1.1. Проверяется, что код новой группы уникален.

Шаг 2. Добавление группы

Шаг 2.1. Если код уникален, группа добавляется в структуру данных.

Шаг 2.2. Данные сохраняются в файл.

Шаг 3. Уведомление пользователя

Шаг 3.1. Пользователю выводится сообщение об успешном добавлении или об ошибке, если код не уникален.

## 3.19 Метод delete\_group(int code)

Шаг 1. Проверка существования группы

Шаг 1.1. Проверяется, существует ли группа с указанным кодом.

Шаг 2. Проверка связанных студентов

Шаг 2.1. Проверяется, связаны ли с этой группой какие-либо студенты.

Шаг 3. Удаление группы

Шаг 3.1. Если группа существует и не связана со студентами, она удаляется из структуры данных.

Шаг 3.2. Данные сохраняются в файл.

Шаг 4. Уведомление пользователя

Шаг 4.1. Пользователю выводится сообщение об успешном удалении или о невозможности удаления.

## 3.20 Метод edit\_group(const int code, const Group& new\_group)

Шаг 1. Поиск группы для изменения

Шаг 1.1. Проходит поиск группы с указанным кодом.

Шаг 2. Проверка уникальности нового кода

Шаг 2.1. Если новый код отличается от текущего, проверяется его уникальность.

Шаг 3. Обновление данных группы

Шаг 3.1. Если проверки пройдены, данные группы обновляются в структуре данных.

Шаг 3.2. Данные сохраняются в файл.

Шаг 4. Уведомление пользователя

Шаг 4.1. Пользователю выводится сообщение об успешном изменении или об ошибке.

## 3.21 Аналогичные методы для Dormitory, Room, Student и AccommodationInfo

Методы is\_dormitory\_code\_unique, dormitory\_exists, add\_dormitory, delete\_dormitory, edit\_dormitory, is\_room\_code\_unique, room\_exists, add\_room, delete\_room, edit\_room, is\_student\_code\_unique, student\_exists, add\_student, delete\_student, edit\_student, is\_accommodation\_unique, add\_accommodation, delete\_accommodation, edit\_accommodation работают аналогично методам для группы, выполняя соответствующие проверки, добавления, удаления и изменения данных для общежитий, комнат, студентов и информации о заселении.

## 3.22 Метод display\_groups()

Шаг 1. Формирование заголовка таблицы

Шаг 1.1. Создаётся строка заголовка с названиями столбцов.

Шаг 2. Вывод заголовка и разделительной линии

Шаг 2.1. Заголовок выводится на экран.

Шаг 2.2. Под заголовком выводится разделительная линия.

Шаг 3. Вывод информации о группах

Шаг 3.1. Для каждой группы в структуре данных формируется строка с её информацией.

Шаг 3.2. Строка выводится на экран.

## 3.23 Методы display\_dormitories(), display\_rooms(), display\_students(), display\_accommodations()

Шаги 1-3 аналогичны методу display\_groups(), но с соответствующими заголовками и данными для общежитий, комнат, студентов и информации о заселении.

## 3.24 Функция menu\_groups(Database& db)

Шаг 1. Отображение подменю управления группами

Шаг 1.1. Пользователю выводятся опции:

Добавить группу

Удалить группу

Изменить группу

Показать все группы

Назад

Шаг 2. Обработка выбора пользователя

Шаг 2.1. Пользователь выбирает действие из предложенных опций.

Шаг 3. Выполнение выбранного действия

Добавить группу:

Шаг 3.1.1. Запрашивается ввод кода группы.

Шаг 3.1.2. Проверяется уникальность кода.

Шаг 3.1.3. Если код уникален, запрашиваются название группы и факультет.

Шаг 3.1.4. Группа добавляется в базу данных.

Шаг 3.1.5. Пользователю сообщается об успешном добавлении.

Удалить группу:

Шаг 3.2.1. Запрашивается ввод кода группы для удаления.

Шаг 3.2.2. Проверяется существование группы.

Шаг 3.2.3. Проверяется наличие студентов в группе.

Шаг 3.2.4. Если проверка успешна, группа удаляется.

Шаг 3.2.5. Пользователю сообщается об успешном удалении или о невозможности удаления.

Изменить группу:

Шаг 3.3.1. Запрашивается ввод кода группы для изменения.

Шаг 3.3.2. Проверяется существование группы.

Шаг 3.3.3. Запрашивается ввод нового кода группы.

Шаг 3.3.4. Проверяется уникальность нового кода, если он изменился.

Шаг 3.3.5. Запрашиваются новое название и факультет.

Шаг 3.3.6. Данные группы обновляются в базе данных.

Шаг 3.3.7. Пользователю сообщается об успешном изменении.

Показать все группы:

Шаг 3.4.1. Вызывается метод для отображения всех групп.

Назад:

Шаг 3.5.1. Возврат в главное меню.

Шаг 4. Повторение меню

Шаг 4.1. Меню отображается снова до тех пор, пока пользователь не выберет "Назад".

## 3.25 Функция menu\_dormitories(Database& db)

Шаги аналогичны функции menu\_groups, но применяются к общежитиям:

Добавление, удаление, изменение общежитий.

Проверка уникальности кода общежития.

Проверка наличия связанных комнат при удалении.

Отображение всех общежитий.

## 3.26 Функция menu\_rooms(Database& db)

Шаги аналогичны функциям menu\_groups и menu\_dormitories, но применяются к комнатам:

Добавление, удаление, изменение комнат.

Проверка уникальности кода комнаты.

Проверка существования общежития при добавлении или изменении комнаты.

Проверка наличия записей о заселении при удалении комнаты.

Отображение всех комнат.

## 3.27 Функция menu\_students(Database& db)

Шаги аналогичны предыдущим меню, но применяются к студентам:

Добавление, удаление, изменение студентов.

Проверка уникальности кода студента.

Проверка существования группы при добавлении или изменении студента.

Проверка наличия информации о заселении при удалении студента.

Валидация формата дат рождения и выдачи паспорта.

Отображение всех студентов.

## 3.28 Функция menu\_accommodations(Database& db)

Шаги аналогичны предыдущим меню, но применяются к информации о заселении:

Добавление, удаление, изменение информации о заселении.

Проверка существования комнаты и студента при добавлении или изменении информации.

Проверка уникальности информации о заселении для студента.

Валидация формата дат заселения и выселения.

Отображение всей информации о заселении.

## 3.29 Функция main\_menu(Database& db)

Шаг 1. Отображение главного меню

Шаг 1.1. Пользователю выводятся опции:

Управление группами

Управление общежитиями

Управление комнатами

Управление студентами

Управление заселением

Выход из программы

Шаг 2. Обработка выбора пользователя

Шаг 2.1. Пользователь выбирает действие из предложенных опций.

Шаг 3. Выполнение выбранного действия

Шаг 3.1. В зависимости от выбора вызывается соответствующая функция управления.

Управление группами → menu\_groups

Управление общежитиями → menu\_dormitories

Управление комнатами → menu\_rooms

Управление студентами → menu\_students

Управление заселением → menu\_accommodations

Выход → завершение программы

Шаг 4. Обработка некорректного выбора

Шаг 4.1. Если выбор некорректен, выводится сообщение об ошибке и меню отображается снова.

Шаг 5. Повторение меню

Шаг 5.1. Меню отображается снова до тех пор, пока пользователь не выберет "Выход".

## 3.30 Основная функция main()

Шаг 1. Инициализация базы данных

Шаг 1.1. Создаётся объект базы данных с использованием указанного файла.

Шаг 2. Запуск главного меню

Шаг 2.1. Вызывается функция main\_menu для начала взаимодействия с пользователем.

Шаг 3. Обработка исключений

Шаг 3.1. Программа отслеживает возможные ошибки форматирования.

Шаг 3.1.1. При возникновении ошибки выводится соответствующее сообщение.

Шаг 3.2. Программа отслеживает ошибки при работе с JSON.

Шаг 3.2.1. При возникновении ошибки выводится соответствующее сообщение.

Шаг 3.3. Программа отслеживает общие исключения.

Шаг 3.3.1. При возникновении общей ошибки выводится соответствующее сообщение.

Шаг 4. Завершение программы

Шаг 4.1. После завершения работы меню программа возвращает код завершения.

Дополнительные шаги в программе

Шаг 5. Обработка ошибок

Шаг 5.1. Отслеживание ошибок форматирования

Шаг 5.1.1. Программа отслеживает ошибки, связанные с неправильным форматированием данных.

Шаг 5.2. Вывод сообщений об ошибках

Шаг 5.2.1. При возникновении ошибки выводится соответствующее сообщение пользователю.

Шаг 5.3. Отслеживание ошибок работы с JSON

Шаг 5.3.1. Программа отслеживает ошибки, возникающие при работе с JSON-данными.

Шаг 5.4. Вывод сообщений об ошибках JSON

Шаг 5.4.1. При возникновении ошибки выводится соответствующее сообщение пользователю.

Шаг 5.5. Отлавливание общих исключений

Шаг 5.5.1. Программа отслеживает любые другие общие ошибки.

Шаг 5.6. Вывод сообщений об общих ошибках

Шаг 5.6.1. При возникновении общей ошибки выводится сообщение о произошедшей ошибке.

Шаг 6. Сохранение данных

Шаг 6.1. Автоматическое сохранение после изменений

Шаг 6.1.1. После каждого добавления, удаления или изменения данных выполняется сохранение текущего состояния базы данных в файл.

Шаг 6.2. Форматирование данных

Шаг 6.2.1. Данные форматируются с отступами для удобного чтения и хранения.

Шаг 6.3. Учет структуры данных

Шаг 6.3.1. При сохранении учитывается структура, содержащая списки групп, общежитий, комнат, студентов и информации о заселении.

Шаг 7. Взаимодействие с пользователем

Шаг 7.1. Постоянные запросы ввода

Шаг 7.1.1. Программа постоянно запрашивает ввод данных у пользователя для выполнения различных операций.

Шаг 7.2. Проверка корректности ввода

Шаг 7.2.1. Все вводимые данные проходят проверку на корректность и соответствие требуемому формату.

Шаг 7.3. Обработка некорректного ввода

Шаг 7.3.1. В случае некорректного ввода пользователю предоставляются сообщения об ошибках и предлагается повторить ввод.

Шаг 7.4. Уведомление о результате операций

Шаг 7.4.1. После успешного выполнения каждой операции пользователю отображается соответствующее уведомление.

Шаг 8. Форматирование и отображение данных

Шаг 8.1. Использование форматирования для вывода

Шаг 8.1.1. При отображении информации используется форматирование для выравнивания столбцов и улучшения читаемости.

Шаг 8.2. Создание заголовков

Шаг 8.2.1. Для каждой категории данных (группы, общежития, комнаты, студенты, заселение) создаются заголовки с соответствующими полями.

Шаг 8.3. Отображение разделительных линий

Шаг 8.3.1. Под заголовками отображаются разделительные линии для визуального отделения заголовков от данных.

Шаг 8.4. Вывод каждой записи

Шаг 8.4.1. Каждая запись данных выводится в отдельной строке с выровненными по столбцам значениями.

Шаг 9. Конец работы программы

Шаг 9.1. Завершение работы по выбору пользователя

Шаг 9.1.1. После выбора пользователем опции выхода, программа завершает свою работу.

Шаг 9.2. Освобождение ресурсов

Шаг 9.2.1. Все открытые файлы закрываются, и ресурсы освобождаются.

# 4 Листинг программы

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <optional>  #include <format>  #include <regex>  #include <limits>  #include "json.hpp"  using json = nlohmann::json;  using namespace std;  namespace  {  int get\_valid\_int(const string& prompt)  {  int value;  while (true)  {  cout << prompt;  cin >> value;  if (cin.fail())  {  cout << "Неверный ввод. Пожалуйста, введите целое число.\n";  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  }  else  {  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  return value;  }  }  }  string get\_valid\_string(const string& prompt)  {  string value;  while (true)  {  cout << prompt;  getline(cin, value);  if (value.empty())  {  cout << "Ввод не может быть пустым. Пожалуйста, попробуйте снова.\n";  }  else  {  return value;  }  }  }  string get\_valid\_date(const string& prompt)  {  string date;  const regex date\_pattern(R"(^\d{2}\.\d{2}\.\d{4}$)");  while (true)  {  cout << prompt;  getline(cin, date);  if (!regex\_match(date, date\_pattern))  {  cout << "Неверный формат даты. Пожалуйста, используйте формат ДД.ММ.ГГГГ.\n";  }  else  {  const int day = stoi(date.substr(0, 2));  const int month = stoi(date.substr(3, 2));  if (day < 1 || day > 31 || month < 1 || month > 12)  {  cout << "Неверная дата. Пожалуйста, введите корректную дату.\n";  }  else  {  return date;  }  }  }  }  }  struct Group  {  int code;  string name;  string faculty;  [[nodiscard]] json to\_json() const  {  return json{{"code", code}, {"name", name}, {"faculty", faculty}};  }  static Group from\_json(const json& j)  {  Group g;  g.code = j.at("code").get<int>();  g.name = j.at("name").get<string>();  g.faculty = j.at("faculty").get<string>();  return g;  }  };  struct Dormitory  {  int code;  string name;  string address;  [[nodiscard]] json to\_json() const  {  return json{{"code", code}, {"name", name}, {"address", address}};  }  static Dormitory from\_json(const json& j)  {  Dormitory d;  d.code = j.at("code").get<int>();  d.name = j.at("name").get<string>();  d.address = j.at("address").get<string>();  return d;  }  };  struct Room  {  int code;  string number;  int dormitory\_code;  [[nodiscard]] json to\_json() const  {  return json{{"code", code}, {"number", number}, {"dormitory\_code", dormitory\_code}};  }  static Room from\_json(const json& j)  {  Room r;  r.code = j.at("code").get<int>();  r.number = j.at("number").get<string>();  r.dormitory\_code = j.at("dormitory\_code").get<int>();  return r;  }  };  struct Student  {  int code;  string full\_name;  string birth\_date;  string gender;  int group\_code;  string passport\_series;  string passport\_number;  string issued\_by;  string issued\_date;  [[nodiscard]] json to\_json() const  {  return json{  {"code", code},  {"full\_name", full\_name},  {"birth\_date", birth\_date},  {"gender", gender},  {"group\_code", group\_code},  {"passport\_series", passport\_series},  {"passport\_number", passport\_number},  {"issued\_by", issued\_by},  {"issued\_date", issued\_date}  };  }  static Student from\_json(const json& j)  {  Student s;  s.code = j.at("code").get<int>();  s.full\_name = j.at("full\_name").get<string>();  s.birth\_date = j.at("birth\_date").get<string>();  s.gender = j.at("gender").get<string>();  s.group\_code = j.at("group\_code").get<int>();  s.passport\_series = j.at("passport\_series").get<string>();  s.passport\_number = j.at("passport\_number").get<string>();  s.issued\_by = j.at("issued\_by").get<string>();  s.issued\_date = j.at("issued\_date").get<string>();  return s;  }  };  struct AccommodationInfo  {  int room\_code;  int student\_code;  string move\_in\_date;  string move\_out\_date;  [[nodiscard]] json to\_json() const  {  return json{  {"room\_code", room\_code},  {"student\_code", student\_code},  {"move\_in\_date", move\_in\_date},  {"move\_out\_date", move\_out\_date}  };  }  static AccommodationInfo from\_json(const json& j)  {  AccommodationInfo a;  a.room\_code = j.at("room\_code").get<int>();  a.student\_code = j.at("student\_code").get<int>();  a.move\_in\_date = j.at("move\_in\_date").get<string>();  a.move\_out\_date = j.at("move\_out\_date").get<string>();  return a;  }  };  class Database  {  string filename;  json data;  public:  Database(string fname) : filename(std::move(fname))  {  ifstream file(filename);  if (file.good())  {  file >> data;  }  else  {  data = json{  {"groups", json::array()},  {"dormitories", json::array()},  {"rooms", json::array()},  {"students", json::array()},  {"accommodations", json::array()}  };  save();  }  }  void save() const  {  ofstream file(filename);  file << std::format("{0:4}\n", data.dump(4));  }  [[nodiscard]] bool is\_group\_code\_unique(const int code) const  {  return std::ranges::all\_of(data["groups"], [code](auto j)  {  return j["code"].template get<int>() != code;  });  }  [[nodiscard]] bool group\_exists(const int code) const  {  return std::ranges::any\_of(data["groups"], [code](auto j)  {  return j["code"].template get<int>() == code;  });  }  void add\_group(const Group& group)  {  if (!is\_group\_code\_unique(group.code))  {  cout << "Группа с таким кодом уже существует. Добавление отменено.\n";  return;  }  data["groups"].push\_back(group.to\_json());  save();  }  void delete\_group(int code)  {  if (!group\_exists(code))  {  std::cout << "Группа с кодом " << code << " не найдена.\n";  return;  }  bool has\_students = std::any\_of(data["students"].begin(), data["students"].end(),  [code](const json& j) { return j["group\_code"] == code; });  if (has\_students)  {  std::cout << "Невозможно удалить группу, так как к ней привязаны студенты.\n";  return;  }  auto& groups = data["groups"];  groups.erase(  std::remove\_if(groups.begin(), groups.end(),  [code](const json& j) { return j["code"] == code; }),  groups.end());  save();  std::cout << "Группа удалена.\n";  }  void edit\_group(const int code, const Group& new\_group)  {  auto& groups = data["groups"];  for (auto& j : groups)  {  if (j["code"] == code)  {  if (new\_group.code != code && !is\_group\_code\_unique(new\_group.code))  {  cout << "Группа с новым кодом уже существует. Изменение отменено.\n";  return;  }  j = new\_group.to\_json();  save();  return;  }  }  cout << "Группа с кодом " << code << " не найдена.\n";  }  [[nodiscard]] bool is\_dormitory\_code\_unique(const int code) const  {  return std::ranges::all\_of(data["dormitories"], [code](auto j)  {  return j["code"].template get<int>() != code;  });  }  [[nodiscard]] bool dormitory\_exists(const int code) const  {  return std::ranges::any\_of(data["dormitories"], [code](auto j)  {  return j["code"].template get<int>() == code;  });  }  void add\_dormitory(const Dormitory& dorm)  {  if (!is\_dormitory\_code\_unique(dorm.code))  {  cout << "Общежитие с таким кодом уже существует. Добавление отменено.\n";  return;  }  data["dormitories"].push\_back(dorm.to\_json());  save();  }  void delete\_dormitory(int code)  {  if (!dormitory\_exists(code))  {  std::cout << "Общежитие с кодом " << code << " не найдено.\n";  return;  }  bool has\_rooms = std::any\_of(data["rooms"].begin(), data["rooms"].end(),  [code](const json& j) { return j["dormitory\_code"] == code; });  if (has\_rooms)  {  std::cout << "Невозможно удалить общежитие, так как к нему привязаны комнаты.\n";  return;  }  auto& dorms = data["dormitories"];  dorms.erase(  std::remove\_if(dorms.begin(), dorms.end(),  [code](const json& j) { return j["code"] == code; }),  dorms.end());  save();  std::cout << "Общежитие удалено.\n";  }  void edit\_dormitory(const int code, const Dormitory& new\_dorm)  {  auto& dorms = data["dormitories"];  for (auto& j : dorms)  {  if (j["code"] == code)  {  if (new\_dorm.code != code && !is\_dormitory\_code\_unique(new\_dorm.code))  {  cout << "Общежитие с новым кодом уже существует. Изменение отменено.\n";  return;  }  j = new\_dorm.to\_json();  save();  return;  }  }  cout << "Общежитие с кодом " << code << " не найдено.\n";  }  [[nodiscard]] bool is\_room\_code\_unique(const int code) const  {  return std::ranges::all\_of(data["rooms"], [code](auto j)  {  return j["code"].template get<int>() != code;  });  }  [[nodiscard]] bool room\_exists(const int code) const  {  return std::ranges::any\_of(data["rooms"], [code](auto j)  {  return j["code"].template get<int>() == code;  });  }  void add\_room(const Room& room)  {  if (!is\_room\_code\_unique(room.code))  {  cout << "Комната с таким кодом уже существует. Добавление отменено.\n";  return;  }  if (!dormitory\_exists(room.dormitory\_code))  {  cout << "Общежитие с кодом " << room.dormitory\_code << " не существует. Добавление комнаты отменено.\n";  return;  }  data["rooms"].push\_back(room.to\_json());  save();  }  void delete\_room(int code)  {  if (!room\_exists(code))  {  std::cout << "Комната с кодом " << code << " не найдена.\n";  return;  }  bool has\_accommodations = std::any\_of(data["accommodations"].begin(), data["accommodations"].end(),  [code](const json& j) { return j["room\_code"] == code; });  if (has\_accommodations)  {  std::cout << "Невозможно удалить комнату, так как к ней привязаны записи заселения.\n";  return;  }  auto& rooms = data["rooms"];  rooms.erase(  std::remove\_if(rooms.begin(), rooms.end(),  [code](const json& j) { return j["code"] == code; }),  rooms.end());  save();  std::cout << "Комната удалена.\n";  }  void edit\_room(const int code, const Room& new\_room)  {  auto& rooms = data["rooms"];  for (auto& j : rooms)  {  if (j["code"] == code)  {  if (new\_room.code != code && !is\_room\_code\_unique(new\_room.code))  {  cout << "Комната с новым кодом уже существует. Изменение отменено.\n";  return;  }  if (!dormitory\_exists(new\_room.dormitory\_code))  {  cout << "Общежитие с кодом " << new\_room.dormitory\_code << " не существует. Изменение комнаты отменено.\n";  return;  }  j = new\_room.to\_json();  save();  return;  }  }  cout << "Комната с кодом " << code << " не найдена.\n";  }  [[nodiscard]] bool is\_student\_code\_unique(const int code) const  {  return std::ranges::all\_of(data["students"], [code](auto j)  {  return j["code"].template get<int>() != code;  });  }  [[nodiscard]] bool student\_exists(const int code) const  {  return std::ranges::any\_of(data["students"], [code](auto j)  {  return j["code"].template get<int>() == code;  });  }  void add\_student(const Student& student)  {  if (!is\_student\_code\_unique(student.code))  {  cout << "Студент с таким кодом уже существует. Добавление отменено.\n";  return;  }  if (!group\_exists(student.group\_code))  {  cout << "Группа с кодом " << student.group\_code << " не существует. Добавление студента отменено.\n";  return;  }  data["students"].push\_back(student.to\_json());  save();  }  void delete\_student(int code)  {  if (!student\_exists(code))  {  std::cout << "Студент с кодом " << code << " не найден.\n";  return;  }  bool has\_accommodation = std::any\_of(data["accommodations"].begin(), data["accommodations"].end(),  [code](const json& j) { return j["student\_code"] == code; });  if (has\_accommodation)  {  std::cout << "Невозможно удалить студента, так как к нему привязана информация о заселении.\n";  return;  }  auto& students = data["students"];  students.erase(  std::remove\_if(students.begin(), students.end(),  [code](const json& j) { return j["code"] == code; }),  students.end());  save();  std::cout << "Студент удален.\n";  }  void edit\_student(const int code, const Student& new\_student)  {  auto& students = data["students"];  for (auto& j : students)  {  if (j["code"] == code)  {  if (new\_student.code != code && !is\_student\_code\_unique(new\_student.code))  {  cout << "Студент с новым кодом уже существует. Изменение отменено.\n";  return;  }  if (!group\_exists(new\_student.group\_code))  {  cout << "Группа с кодом " << new\_student.group\_code << " не существует. Изменение студента отменено.\n";  return;  }  j = new\_student.to\_json();  save();  return;  }  }  cout << "Студент с кодом " << code << " не найден.\n";  }  [[nodiscard]] bool is\_accommodation\_unique(const int student\_code) const  {  return std::ranges::all\_of(data["accommodations"], [student\_code](auto j)  {  return j["student\_code"].template get<int>() != student\_code;  });  }  void add\_accommodation(const AccommodationInfo& acc)  {  if (!room\_exists(acc.room\_code))  {  cout << "Комната с кодом " << acc.room\_code << " не существует. Добавление информации о заселении отменено.\n";  return;  }  if (!student\_exists(acc.student\_code))  {  cout << "Студент с кодом " << acc.student\_code << " не существует. Добавление информации о заселении отменено.\n";  return;  }  if (!is\_accommodation\_unique(acc.student\_code))  {  cout << "Информация о заселении для этого студента уже существует. Добавление отменено.\n";  return;  }  data["accommodations"].push\_back(acc.to\_json());  save();  }  void delete\_accommodation(int student\_code)  {  auto& accs = data["accommodations"];  auto it = std::find\_if(accs.begin(), accs.end(),  [student\_code](const json& j) { return j["student\_code"] == student\_code; });  if (it == accs.end())  {  std::cout << "Информация о заселении для студента с кодом " << student\_code << " не найдена.\n";  return;  }  accs.erase(it);  save();  std::cout << "Информация о заселении удалена.\n";  }  void edit\_accommodation(const int student\_code, const AccommodationInfo& new\_acc)  {  auto& accs = data["accommodations"];  for (auto& j : accs)  {  if (j["student\_code"] == student\_code)  {  if (!room\_exists(new\_acc.room\_code))  {  cout << "Комната с кодом " << new\_acc.room\_code << " не существует. Изменение информации о заселении отменено.\n";  return;  }  if (!student\_exists(new\_acc.student\_code))  {  cout << "Студент с кодом " << new\_acc.student\_code << " не существует. Изменение информации о заселении отменено.\n";  return;  }  j = new\_acc.to\_json();  save();  return;  }  }  cout << "Информация о заселении для студента с кодом " << student\_code << " не найдена.\n";  }  void display\_groups() const  {  const string header = std::format("{:<10} {:<30} {:<30}\n", "Код", "Название группы", "Факультет");  cout << header;  cout << string(header.size() - 1, '-') << '\n';  for (const auto& j : data["groups"])  {  string row = std::format("{:<10} {:<30} {:<30}\n",  j["code"].get<int>(),  j["name"].get<string>(),  j["faculty"].get<string>());  cout << row;  }  }  void display\_dormitories() const  {  const string header = std::format("{:<10} {:<30} {:<50}\n", "Код", "Название", "Адрес");  cout << header;  cout << string(header.size() - 1, '-') << '\n';  for (const auto& j : data["dormitories"])  {  string row = std::format("{:<10} {:<30} {:<50}\n",  j["code"].get<int>(),  j["name"].get<string>(),  j["address"].get<string>());  cout << row;  }  }  void display\_rooms() const  {  const string header = std::format("{:<10} {:<20} {:<20}\n", "Код", "Номер комнаты", "Код общежития");  cout << header;  cout << string(header.size() - 1, '-') << '\n';  for (const auto& j : data["rooms"])  {  string row = std::format("{:<10} {:<20} {:<20}\n",  j["code"].get<int>(),  j["number"].get<string>(),  j["dormitory\_code"].get<int>());  cout << row;  }  }  void display\_students() const  {  const string header = std::format("{:<10} {:<30} {:<15} {:<10} {:<10} {:<10} {:<15} {:<30} {:<15}\n",  "Код", "ФИО", "Дата рождения", "Пол", "Код группы",  "Серия", "Номер паспорта", "Кем выдан", "Дата выдачи");  cout << header;  cout << string(header.size() - 1, '-') << '\n';  for (const auto& j : data["students"])  {  string row = std::format("{:<10} {:<30} {:<15} {:<10} {:<10} {:<10} {:<15} {:<30} {:<15}\n",  j["code"].get<int>(),  j["full\_name"].get<string>(),  j["birth\_date"].get<string>(),  j["gender"].get<string>(),  j["group\_code"].get<int>(),  j["passport\_series"].get<string>(),  j["passport\_number"].get<string>(),  j["issued\_by"].get<string>(),  j["issued\_date"].get<string>());  cout << row;  }  }  void display\_accommodations() const  {  const string header = std::format("{:<15} {:<15} {:<15} {:<15}\n", "Код комнаты", "Код студента", "Дата заселения",  "Дата выселения");  cout << header;  cout << string(header.size() - 1, '-') << '\n';  for (const auto& j : data["accommodations"])  {  string row = std::format("{:<15} {:<15} {:<15} {:<15}\n",  j["room\_code"].get<int>(),  j["student\_code"].get<int>(),  j["move\_in\_date"].get<string>(),  j["move\_out\_date"].get<string>());  cout << row;  }  }  };  namespace  {  void menu\_groups(Database& db)  {  int choice;  do  {  cout << "\n--- Управление группами ---\n";  cout << "1. Добавить группу\n";  cout << "2. Удалить группу\n";  cout << "3. Изменить группу\n";  cout << "4. Показать все группы\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  if (choice == 1)  {  Group g;  while (true)  {  g.code = get\_valid\_int("Введите код группы: ");  if (db.is\_group\_code\_unique(g.code))  break;  cout << "Группа с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  g.name = get\_valid\_string("Введите название группы: ");  g.faculty = get\_valid\_string("Введите название факультета: ");  db.add\_group(g);  cout << "Группа добавлена.\n";  }  else if (choice == 2)  {  const int code = get\_valid\_int("Введите код группы для удаления: ");  db.delete\_group(code);  }  else if (choice == 3)  {  const int code = get\_valid\_int("Введите код группы для изменения: ");  if (!db.group\_exists(code))  {  cout << "Группа с таким кодом не существует.\n";  continue;  }  Group g;  while (true)  {  g.code = get\_valid\_int("Введите новый код группы: ");  if (g.code == code || db.is\_group\_code\_unique(g.code))  break;  cout << "Группа с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  g.name = get\_valid\_string("Введите новое название группы: ");  g.faculty = get\_valid\_string("Введите новое название факультета: ");  db.edit\_group(code, g);  cout << "Группа изменена.\n";  }  else if (choice == 4)  {  db.display\_groups();  }  }  while (choice != 0);  }  void menu\_dormitories(Database& db)  {  int choice;  do  {  cout << "\n--- Управление общежитиями ---\n";  cout << "1. Добавить общежитие\n";  cout << "2. Удалить общежитие\n";  cout << "3. Изменить общежитие\n";  cout << "4. Показать все общежития\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  if (choice == 1)  {  Dormitory d;  while (true)  {  d.code = get\_valid\_int("Введите код общежития: ");  if (db.is\_dormitory\_code\_unique(d.code))  break;  cout << "Общежитие с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  d.name = get\_valid\_string("Введите название общежития: ");  d.address = get\_valid\_string("Введите адрес общежития: ");  db.add\_dormitory(d);  cout << "Общежитие добавлено.\n";  }  else if (choice == 2)  {  const int code = get\_valid\_int("Введите код общежития для удаления: ");  db.delete\_dormitory(code);  }  else if (choice == 3)  {  const int code = get\_valid\_int("Введите код общежития для изменения: ");  if (!db.dormitory\_exists(code))  {  cout << "Общежитие с таким кодом не существует.\n";  continue;  }  Dormitory d;  while (true)  {  d.code = get\_valid\_int("Введите новый код общежития: ");  if (d.code == code || db.is\_dormitory\_code\_unique(d.code))  break;  cout << "Общежитие с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  d.name = get\_valid\_string("Введите новое название общежития: ");  d.address = get\_valid\_string("Введите новый адрес общежития: ");  db.edit\_dormitory(code, d);  cout << "Общежитие изменено.\n";  }  else if (choice == 4)  {  db.display\_dormitories();  }  }  while (choice != 0);  }  void menu\_rooms(Database& db)  {  int choice;  do  {  cout << "\n--- Управление комнатами ---\n";  cout << "1. Добавить комнату\n";  cout << "2. Удалить комнату\n";  cout << "3. Изменить комнату\n";  cout << "4. Показать все комнаты\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  if (choice == 1)  {  Room r;  while (true)  {  r.code = get\_valid\_int("Введите код комнаты: ");  if (db.is\_room\_code\_unique(r.code))  break;  cout << "Комната с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  r.number = get\_valid\_string("Введите номер комнаты: ");  while (true)  {  r.dormitory\_code = get\_valid\_int("Введите код общежития: ");  if (db.dormitory\_exists(r.dormitory\_code))  break;  cout << "Общежитие с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  db.add\_room(r);  cout << "Комната добавлена.\n";  }  else if (choice == 2)  {  const int code = get\_valid\_int("Введите код комнаты для удаления: ");  db.delete\_room(code);  }  else if (choice == 3)  {  const int code = get\_valid\_int("Введите код комнаты для изменения: ");  if (!db.room\_exists(code))  {  cout << "Комната с таким кодом не существует.\n";  continue;  }  Room r;  while (true)  {  r.code = get\_valid\_int("Введите новый код комнаты: ");  if (r.code == code || db.is\_room\_code\_unique(r.code))  break;  cout << "Комната с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  r.number = get\_valid\_string("Введите новый номер комнаты: ");  while (true)  {  r.dormitory\_code = get\_valid\_int("Введите новый код общежития: ");  if (db.dormitory\_exists(r.dormitory\_code))  break;  cout << "Общежития с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  db.edit\_room(code, r);  cout << "Комната изменена.\n";  }  else if (choice == 4)  {  db.display\_rooms();  }  }  while (choice != 0);  }  void menu\_students(Database& db)  {  int choice;  do  {  cout << "\n--- Управление студентами ---\n";  cout << "1. Добавить студента\n";  cout << "2. Удалить студента\n";  cout << "3. Изменить студента\n";  cout << "4. Показать всех студентов\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  if (choice == 1)  {  Student s;  while (true)  {  s.code = get\_valid\_int("Введите код студента: ");  if (db.is\_student\_code\_unique(s.code))  break;  cout << "Студент с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  s.full\_name = get\_valid\_string("Введите ФИО студента: ");  s.birth\_date = get\_valid\_date("Введите дату рождения (ДД.ММ.ГГГГ): ");  s.gender = get\_valid\_string("Введите пол: ");  while (true)  {  s.group\_code = get\_valid\_int("Введите код группы: ");  if (db.group\_exists(s.group\_code))  break;  cout << "Группа с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  s.passport\_series = get\_valid\_string("Введите серию паспорта: ");  s.passport\_number = get\_valid\_string("Введите номер паспорта: ");  s.issued\_by = get\_valid\_string("Кем выдан паспорт: ");  s.issued\_date = get\_valid\_date("Дата выдачи паспорта (ДД.ММ.ГГГГ): ");  db.add\_student(s);  cout << "Студент добавлен.\n";  }  else if (choice == 2)  {  const int code = get\_valid\_int("Введите код студента для удаления: ");  db.delete\_student(code);  }  else if (choice == 3)  {  const int code = get\_valid\_int("Введите код студента для изменения: ");  if (!db.student\_exists(code))  {  cout << "Студент с таким кодом не существует.\n";  continue;  }  Student s;  while (true)  {  s.code = get\_valid\_int("Введите новый код студента: ");  if (s.code == code || db.is\_student\_code\_unique(s.code))  break;  cout << "Студент с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  s.full\_name = get\_valid\_string("Введите новое ФИО студента: ");  s.birth\_date = get\_valid\_date("Введите новую дату рождения (ДД.ММ.ГГГГ): ");  s.gender = get\_valid\_string("Введите новый пол: ");  while (true)  {  s.group\_code = get\_valid\_int("Введите новый код группы: ");  if (db.group\_exists(s.group\_code))  break;  cout << "Группа с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  s.passport\_series = get\_valid\_string("Введите новую серию паспорта: ");  s.passport\_number = get\_valid\_string("Введите новый номер паспорта: ");  s.issued\_by = get\_valid\_string("Кем выдан паспорт: ");  s.issued\_date = get\_valid\_date("Новая дата выдачи паспорта (ДД.ММ.ГГГГ): ");  db.edit\_student(code, s);  cout << "Студент изменен.\n";  }  else if (choice == 4)  {  db.display\_students();  }  }  while (choice != 0);  }  void menu\_accommodations(Database& db)  {  int choice;  do  {  cout << "\n--- Управление заселением ---\n";  cout << "1. Добавить информацию о заселении\n";  cout << "2. Удалить информацию о заселении\n";  cout << "3. Изменить информацию о заселении\n";  cout << "4. Показать всю информацию о заселении\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  if (choice == 1)  {  AccommodationInfo a;  while (true)  {  a.room\_code = get\_valid\_int("Введите код комнаты: ");  if (db.room\_exists(a.room\_code))  break;  cout << "Комната с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  while (true)  {  a.student\_code = get\_valid\_int("Введите код студента: ");  if (db.student\_exists(a.student\_code))  {  if (db.is\_accommodation\_unique(a.student\_code))  {  break;  }  cout << "Информация о заселении для этого студента уже существует.\n";  return;  }  cout << "Студент с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  a.move\_in\_date = get\_valid\_date("Введите дату заселения (ДД.ММ.ГГГГ): ");  a.move\_out\_date = get\_valid\_date("Введите дату выселения (ДД.ММ.ГГГГ): ");  db.add\_accommodation(a);  cout << "Информация о заселении добавлена.\n";  }  else if (choice == 2)  {  const int student\_code = get\_valid\_int("Введите код студента для удаления информации о заселении: ");  db.delete\_accommodation(student\_code);  }  else if (choice == 3)  {  const int student\_code = get\_valid\_int("Введите код студента для изменения информации о заселении: ");  if (!db.is\_accommodation\_unique(student\_code) && db.student\_exists(student\_code))  {  AccommodationInfo a;  while (true)  {  a.room\_code = get\_valid\_int("Введите новый код комнаты: ");  if (db.room\_exists(a.room\_code))  break;  cout << "Комната с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  a.student\_code = student\_code;  a.move\_in\_date = get\_valid\_date("Введите новую дату заселения (ДД.ММ.ГГГГ): ");  a.move\_out\_date = get\_valid\_date("Введите новую дату выселения (ДД.ММ.ГГГГ): ");  db.edit\_accommodation(student\_code, a);  cout << "Информация о заселении изменена.\n";  }  else  {  cout << "Информация о заселении для этого студента не существует.\n";  }  }  else if (choice == 4)  {  db.display\_accommodations();  }  }  while (choice != 0);  }  void main\_menu(Database& db)  {  int choice;  do  {  string menu =  std::format("\n=== Учёт студентов в общежитиях ===\n"  "1. Управление группами\n"  "2. Управление общежитиями\n"  "3. Управление комнатами\n"  "4. Управление студентами\n"  "5. Управление заселением\n"  "0. Выход\n"  "Выберите действие: ");  cout << menu;  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  switch (choice)  {  case 1:  menu\_groups(db);  break;  case 2:  menu\_dormitories(db);  break;  case 3:  menu\_rooms(db);  break;  case 4:  menu\_students(db);  break;  case 5:  menu\_accommodations(db);  break;  case 0:  cout << "Выход из программы.\n";  break;  default:  cout << "Неверный выбор. Попробуйте снова.\n";  }  }  while (choice != 0);  }  }  int main()  {  try  {  Database db("database.json");  main\_menu(db);  }  catch (const std::format\_error& e)  {  cerr << "Ошибка форматирования: " << e.what() << '\n';  }  catch (const json::exception& e)  {  cerr << "Ошибка работы с JSON: " << e.what() << '\n';  }  catch (const exception& e)  {  cerr << "Произошла ошибка: " << e.what() << '\n';  }  return 0;  } |

# 5 Результаты работы программы

Главное меню представлено на рисунке 2.

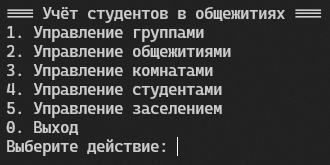


Рисунок 2 — Главное меню

Добавление группы представлено на рисунке 3.

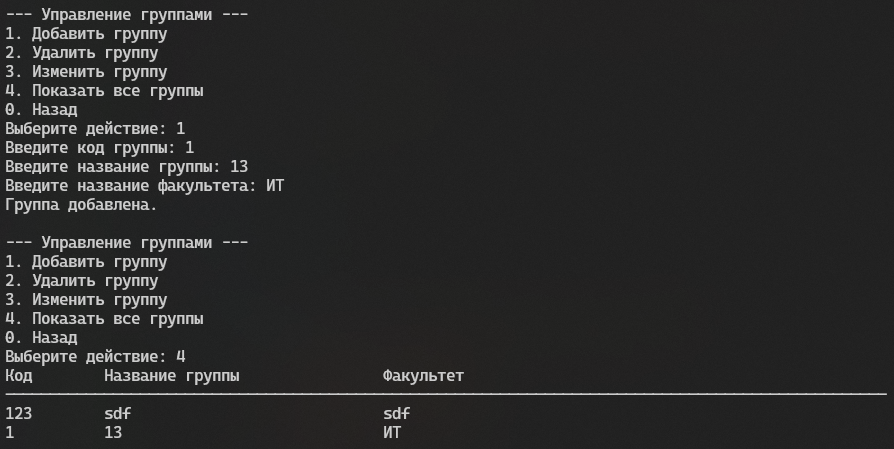


Рисунок 3 — Добавление группы

Изменение и удаление группы представлено на рисунке 4.

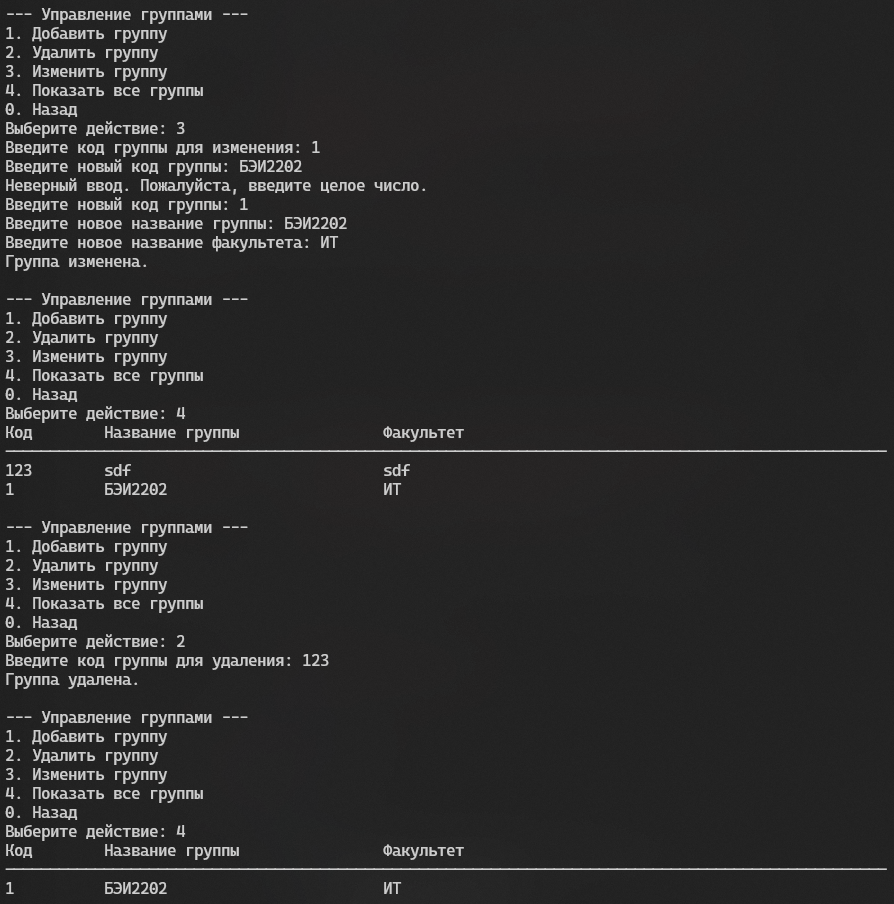


Рисунок 4 — Изменение и удаление группы

Все остальные объекты управляются аналогично.

Изменённый JSON-файл изображён на рисунке 5.



Рисунок — JSON-файл

# 6 Вывод

В процессе работы были изучены механизмы доступа и передачи данных с помощью JSON.