МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения

вычислительной техники и автоматизированных

систем

Курсовая работа

по дисциплине: ООП

тема: «**Создание системы рейтинга и оценок преподавателей и курсов университета**»

Выполнил: студент группы
ПВ-233
Мороз Роман Алексеевич

Проверили:
Мороз Данила Александрович

Белгород 2025 г.

Цель: Создание системы оценки преподавателей и учебных курсов, обеспечивающей сбор, анализ и хранение данных на основе мнений студентов. Система должна предоставлять инструменты для:

- Интерактивного взаимодействия пользователей через интерфейс.
- Автоматического расчета рейтингов по заданным критериям.
- Интеграции с существующими университетскими системами (LMS, базы данных).
- Генерации аналитических отчетов для администраторов и преподавателей.

Постановка задачи

1. **Архитектура приложения**:
    - Проектирование модульной системы на основе принципов ООП с использованием методологии Гради Бутча.
    - Реализация слоистой архитектуры (презентационный, бизнес-логики, данных).
    - Обеспечение масштабируемости (поддержка новых типов оценок, критериев, факультетов).
2. **Пользовательский интерфейс**:
    - Интерфейс для студентов (оценка курсов/преподавателей, просмотр рейтингов).
    - Административная панель (управление критериями, генерация отчетов, модерация отзывов).
3. **Логика оценки и рейтингов**:
    - Валидация оценок (ограничение диапазона, проверка уникальности для избежания спама).
    - Расчет рейтингов по взвешенным критериям (например: качество лекций, доступность материалов, коммуникация).
    -

Функциональные требования

1. **Пользовательский интерфейс**:
    - Формы оценки с критериями (1–5 баллов) и текстовыми отзывами.
    - Интерактивные графики рейтингов (столбчатые диаграммы, heatmap по факультетам).

- Поиск преподавателей/курсов с автодополнением.

2. **Логика оценки**:
   - Проверка прав доступа (только студенты, завершившие курс, могут оставить оценку).
   - Учет анонимности отзывов (настройка через административную панель).
   - Расчет средневзвешенных рейтингов с учетом весов критериев.
3. **Аналитика и отчетность**:
   - Генерация сводных отчетов за семестр/год.
   - Сравнение рейтингов преподавателей внутри кафедры.
   - Выявление трендов (рост/падение оценок за несколько лет).

Диаграмма объектов:

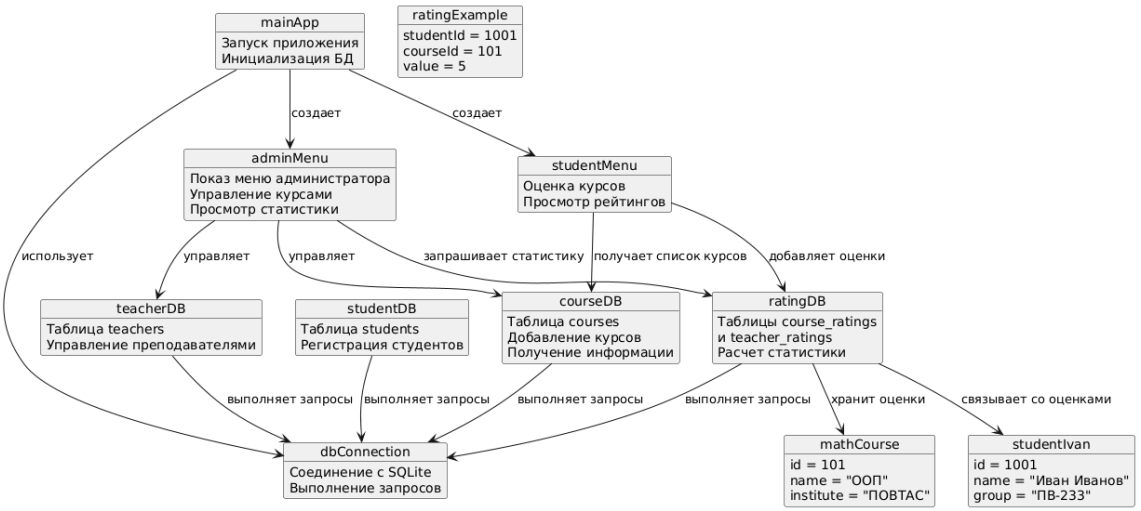
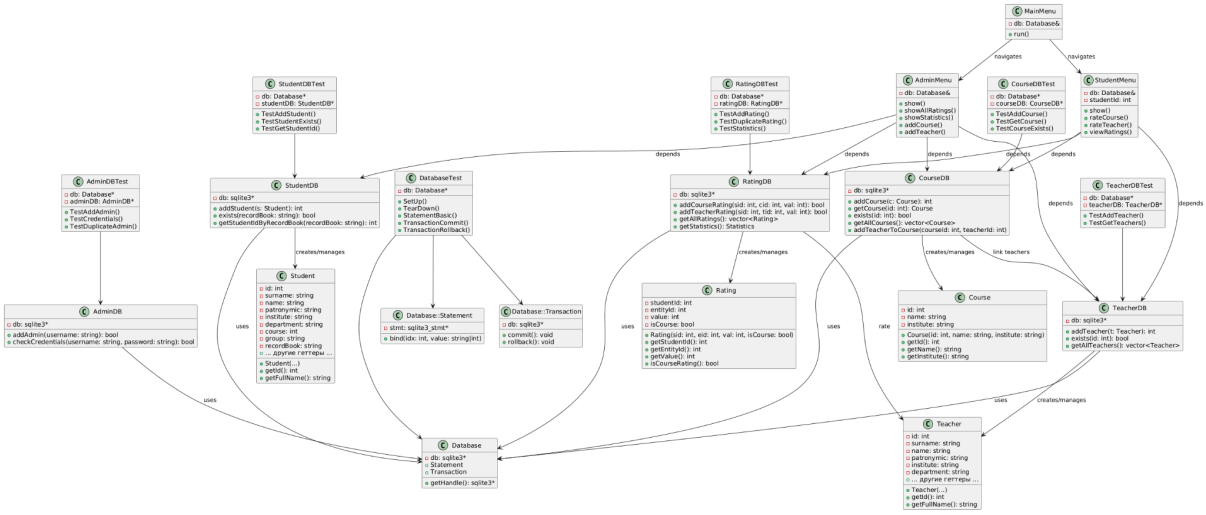


Диаграмма классов:



Паттерны проектирования:

1. Singleton (Одиночка)

Тип: Порождающий паттерн
Класс: Database
Описание:
Класс Database реализован как Singleton, что гарантирует единственный экземпляр подключения к SQLite-базе данных на всё приложение. Это обеспечивает:

- Консистентность данных (все компоненты работают с одной БД).
- Безопасное управление ресурсами (открытие/закрытие соединения).
- Упрощение доступа к БД через статический метод getInstance().

2. Repository (Репозиторий)

Тип: Структурный паттерн
Классы: CourseDB, StudentDB, RatingDB
Описание:
Каждый репозиторий инкапсулирует логику работы с данными для конкретной сущности (курсы, студенты, оценки). Это позволяет:

- Отделить бизнес-логику от механизмов хранения данных.
- Упростить тестирование (можно подменить реализацию на mock-объекты).
- Централизовать CRUD-операции.

3. MVC (Model-View-Controller)

Тип: Архитектурный паттерн
Компоненты:

- Model: Course, Student, Rating (сущности) + *DB (репозитории).
- View: CLI-меню (AdminMenu, StudentMenu).
- Controller: Методы в меню (rateCourse(), addTeacher()).

Описание:
Чёткое разделение ответственности:

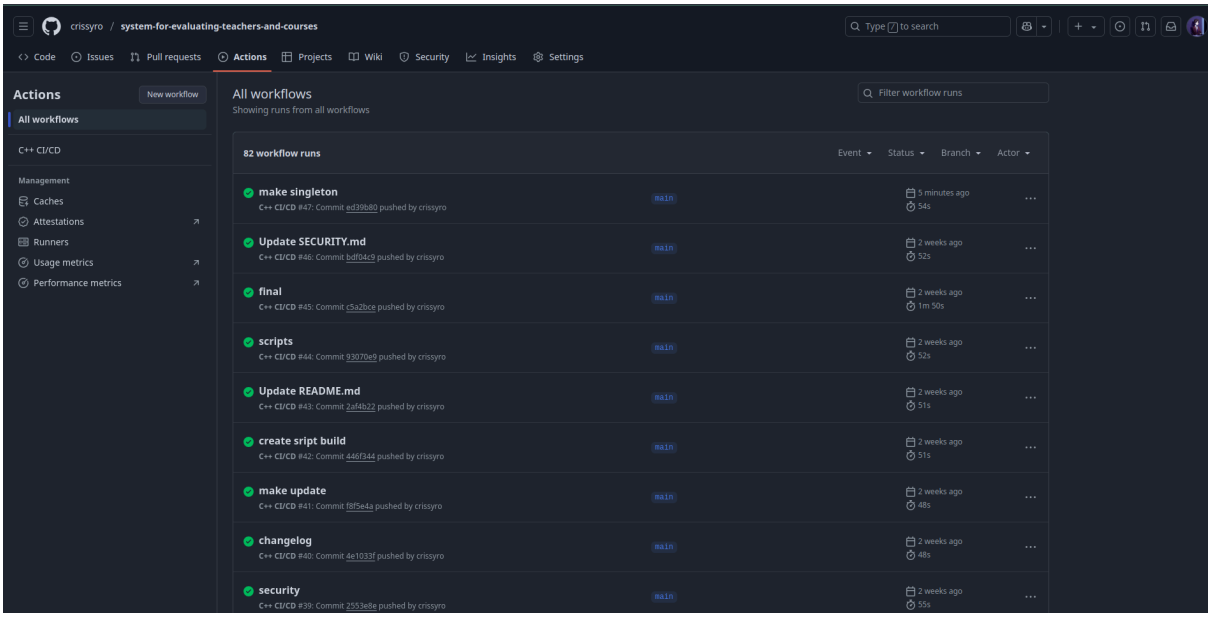
- Model хранит данные и логику работы с БД.
- View отображает интерфейс (цветной текст, таблицы).
- Controller обрабатывает ввод и координирует Model и View.

Преимущества:

- Возможность замены View (например, на графический интерфейс) без изменения Model.

Ссылка на репозиторий:
https://github.com/crissyro/system-for-evaluating-teachers-and-courses

Мечта всех разработчиков:



Код программы:

course.hpp

```cpp
/**
* @file course.hpp
* @brief Заголовочный файл класса Course, представляющего учебный курс
*/

#pragma once

#include <string>

namespace course {

/**
* @class Course
* @brief Класс, описывающий учебный курс университета
*
* Хранит базовую информацию о курсе:
* - Уникальный идентификатор
* - Название курса
* - Институт, к которому относится курс
*/
class Course {
private:
 int id;  ///< Уникальный идентификатор курса
 std::string name;        ///< Название курса
 std::string institute;  ///< Институт, предлагающий курс

public:
 /**
  * @brief Конструктор курса
  * @param id Числовой идентификатор курса (должен быть > 0)
  * @param name Название курса (непустая строка)
  * @param institute Название института (непустая строка)
  */
 Course(int id, const std::string& name, const std::string& institute);

 /// @brief Виртуальный деструктор по умолчанию
 ~Course() = default;

 /**
  * @brief Получить идентификатор курса
  * @return Целочисленный уникальный идентификатор
  */
 int getId() const;
```

```
 /**
  * @brief Получить название курса
  * @return Константная ссылка на строку с названием
  */
 const std::string& getName() const;

 /**
  * @brief Получить название института
  * @return Константная ссылка на строку с институтом
  */
 const std::string& getInstitute() const;

};  // class Course

}  // namespace course
```

course.cpp

```
#include "../../include/classes/course.hpp"

namespace course {

Course::Course(int id, const std::string& name, const std::string&
institute)
    : id(id), name(name), institute(institute) {}

int Course::getId() const {
    return id;
}

const std::string& Course::getName() const {
    return name;
}

const std::string& Course::getInstitute() const {
    return institute;
}

}  // namespace course
```

rating.hpp

```
/**
* @file rating.hpp
* @brief Заголовочный файл класса Rating, представляющего оценку
*/

#pragma once

namespace rating {

/**
* @class Rating
* @brief Класс, хранящий информацию об оценке
*
* Может представлять оценку как для курса, так и для преподавателя.
* Содержит:
* - ID студента
* - ID оцениваемой сущности (курса или преподавателя)
* - Значение оценки
* - Флаг типа оценки
*/
class Rating {
private:
 int studentId;  ///< ID студента, поставившего оценку
 int entityId;  ///< ID сущности (курса или преподавателя)
 int value;      ///< Значение оценки (1-5)
 bool isCourse;  ///< Флаг типа оценки (true - курс, false -
преподаватель)

public:
 /**
  * @brief Конструктор оценки
  * @param sid ID студента
  * @param eid ID сущности
  * @param val Значение оценки
  * @param isCourse Флаг типа оценки
  */
 Rating(int sid, int eid, int val, bool isCourse);

 /// @brief Деструктор по умолчанию
 ~Rating() = default;

 /**
```

```
     * @brief Получить ID студента
     * @return Целочисленный ID студента
     */
    int getStudentId() const;

    /**
     * @brief Получить ID сущности
     * @return Целочисленный ID курса/преподавателя
     */
    int getEntityId() const;

    /**
     * @brief Получить значение оценки
     * @return Оценка в диапазоне 1-5
     */
    int getValue() const;

    /**
     * @brief Проверка типа оценки
     * @return true - оценка за курс, false - за преподавателя
     */
    bool isCourseRating() const;

};  // class Rating

}  // namespace rating
```

rating.cpp

```
#include "../../include/classes/rating.hpp"

namespace rating {

Rating::Rating(int sid, int eid, int val, bool isCourse)
    : studentId(sid), entityId(eid), value(val), isCourse(isCourse) {}

int Rating::getStudentId() const {
    return studentId;
}

int Rating::getEntityId() const {
    return entityId;
}
```

```cpp
int Rating::getValue() const {
    return value;
}

bool Rating::isCourseRating() const {
    return isCourse;
}

}  // namespace rating
```

student.hpp

```cpp
/**
* @file student.hpp
* @brief Заголовочный файл класса Student, представляющего студента
*/

#pragma once

#include <string>

namespace student {

/**
* @class Student
* @brief Класс, описывающий студента университета
*
* Содержит полную информацию о студенте:
* - Персональные данные
* - Учебные характеристики
* - Уникальные идентификаторы
*/
class Student {
private:
 int id;                    ///< Уникальный идентификатор
 std::string surname;       ///< Фамилия студента
 std::string name;          ///< Имя студента
 std::string patronymic;    ///< Отчество студента
 std::string institute;     ///< Институт/факультет
 std::string department;    ///< Кафедра
 int course;                ///< Курс обучения
 std::string group;         ///< Учебная группа
```

```cpp
 std::string recordBook;  ///< Номер зачетной книжки

public:
 /**
  * @brief Конструктор для нового студента (без ID)
  * @param surname Фамилия
  * @param name Имя
  * @param patronymic Отчество
  * @param institute Институт
  * @param department Кафедра
  * @param course Курс
  * @param group Группа
  * @param recordBook Номер зачетки
  */
 Student(const std::string& surname, const std::string& name,
         const std::string& patronymic, const std::string& institute,
         const std::string& department, int course, const std::string&
group,
         const std::string& recordBook);

 /**
  * @brief Конструктор для существующего студента (с ID)
  * @param id Уникальный идентификатор
  * @param surname Фамилия
  * @param name Имя
  * @param patronymic Отчество
  * @param institute Институт
  * @param department Кафедра
  * @param course Курс
  * @param group Группа
  * @param recordBook Номер зачетки
  */
 Student(int id, const std::string& surname, const std::string& name,
         const std::string& patronymic, const std::string& institute,
         const std::string& department, int course, const std::string&
group,
         const std::string& recordBook);

 /// @brief Деструктор по умолчанию
 ~Student() = default;

 /// @name Геттеры
 /// @{
```

```
 int getId() const;
 std::string getSurname() const;
 std::string getName() const;
 std::string getPatronymic() const;
 std::string getInstitute() const;
 std::string getDepartment() const;
 int getCourse() const;
 std::string getGroup() const;
 std::string getRecordBook() const;
 /// @}

 /**
  * @brief Получить полное имя
  * @return Строка формата "Фамилия Имя Отчество"
  */
 std::string getFullName() const;

};  // class Student

}  // namespace student
```

student.cpp

```
#include "../../include/classes/student.hpp"

namespace student {

Student::Student(const std::string& surname, const std::string& name,
const std::string& patronymic,
                 const std::string& institute, const std::string&
department, int course, const std::string& group,
                 const std::string& recordBook)
    : id(0),
      surname(surname),
      name(name),
      patronymic(patronymic),
      institute(institute),
      department(department),
      course(course),
      group(group),
      recordBook(recordBook) {}
```

```cpp
Student::Student(int id, const std::string& surname, const std::string&
name, const std::string& patronymic,
                 const std::string& institute, const std::string&
department, int course, const std::string& group,
                 const std::string& recordBook)
    : id(id),
      surname(surname),
      name(name),
      patronymic(patronymic),
      institute(institute),
      department(department),
      course(course),
      group(group),
      recordBook(recordBook) {}

int Student::getId() const {
    return id;
}

std::string Student::getSurname() const {
    return surname;
}

std::string Student::getName() const {
    return name;
}

std::string Student::getPatronymic() const {
    return patronymic;
}

std::string Student::getInstitute() const {
    return institute;
}

std::string Student::getDepartment() const {
    return department;
}

int Student::getCourse() const {
    return course;
}
```

```cpp
std::string Student::getGroup() const {
    return group;
}

std::string Student::getRecordBook() const {
    return recordBook;
}

std::string Student::getFullName() const {
    return surname + " " + name + " " + patronymic;
}

}  // namespace student
```

teacher.hpp

```cpp
/**
* @file teacher.hpp
* @brief Заголовочный файл класса Teacher, представляющего
преподавателя
*/

#pragma once

#include <string>

namespace teacher {

/**
* @class Teacher
* @brief Класс, описывающий преподавателя университета
*
* Содержит информацию:
* - Персональные данные
* - Место работы
* - Уникальный идентификатор
*/
class Teacher {
private:
 int id;                    ///< Уникальный идентификатор
 std::string surname;       ///< Фамилия
```

```
    std::string name;         ///< Имя
    std::string patronymic;   ///< Отчество
    std::string institute;    ///< Институт/факультет
    std::string department;   ///< Кафедра

public:
    /**
     * @brief Конструктор преподавателя
     * @param id Уникальный идентификатор
     * @param surname Фамилия
     * @param name Имя
     * @param patronymic Отчество
     * @param institute Институт
     * @param department Кафедра
     */
    Teacher(int id, const std::string& surname, const std::string& name,
            const std::string& patronymic, const std::string& institute,
            const std::string& department);

    /// @brief Деструктор по умолчанию
    ~Teacher() = default;

    /// @name Геттеры
    /// @{
    int getId() const;
    std::string getSurname() const;
    std::string getName() const;
    std::string getPatronymic() const;
    std::string getInstitute() const;
    std::string getDepartment() const;
    /// @}

    /**
     * @brief Получить полное имя
     * @return Строка формата "Фамилия Имя Отчество"
     */
    std::string getFullName() const;

};  // class Teacher

}  // namespace teacher
```

teacher.cpp

```cpp
#include "../../include/classes/teacher.hpp"

namespace teacher {

Teacher::Teacher(int id, const std::string& surname, const std::string&
name, const std::string& patronymic,
                 const std::string& institute, const std::string&
department)
    : id(id), surname(surname), name(name), patronymic(patronymic),
institute(institute), department(department) {}

int Teacher::getId() const {
    return id;
}

std::string Teacher::getSurname() const {
    return surname;
}

std::string Teacher::getName() const {
    return name;
}

std::string Teacher::getPatronymic() const {
    return patronymic;
}

std::string Teacher::getInstitute() const {
    return institute;
}

std::string Teacher::getDepartment() const {
    return department;
}

std::string Teacher::getFullName() const {
    return surname + " " + name + " " + patronymic;
}

}  // namespace teacher
```

admin_db.hpp

```cpp
/**
* @file admin_db.hpp
* @brief Работа с администраторами в базе данных
*/

#pragma once

#include <string>

#include "database.hpp"

namespace database {

/**
* @class AdminDB
* @brief Управление учетными записями администраторов
*
* Обеспечивает:
* - Добавление новых администраторов
* - Проверку существования пользователя
* - Аутентификацию
*/
class AdminDB {
private:
 sqlite3* db;  ///< Указатель на соединение с БД

public:
 /**
  * @brief Конструктор, инициализирующий соединение
  * @param database Экземпляр базы данных
  */
 explicit AdminDB(const Database& database);

 /// @brief Деструктор по умолчанию
 ~AdminDB() = default;

 /**
  * @brief Добавить администратора
```

```
     * @param username Логин администратора
     * @return true если операция успешна
     */
    bool addAdmin(const std::string& username);

    /**
     * @brief Проверить существование администратора
     * @param username Логин для проверки
     * @return true если пользователь существует
     */
    bool adminExists(const std::string& username);

    /**
     * @brief Проверка учетных данных
     * @param username Логин
     * @param password Пароль
     * @return true если аутентификация успешна
     */
    bool checkCredentials(const std::string& username,
                          const std::string& password);

};  // class AdminDB

}  // namespace database
```

admin_db.cpp

```
#include "../../include/database/admin_db.hpp"

#include <stdexcept>

namespace database {

AdminDB::AdminDB(const Database& database) : db(database.getHandle()) {
    const char* sql = R"(
        CREATE TABLE IF NOT EXISTS admins (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL
        );)";
```

```
    Database::Statement stmt(db, sql);
    sqlite3_step(stmt);

    if (!adminExists("admin")) {
        const char* insert_sql = "INSERT INTO admins (username,
password) VALUES ('admin', 'admin123');";

        Database::Statement insert_stmt(db, insert_sql);
        sqlite3_step(insert_stmt);
    }
}

bool AdminDB::adminExists(const std::string& username) {
    const char* sql = "SELECT 1 FROM admins WHERE username = ?;";

    Database::Statement stmt(db, sql);
    stmt.bind(1, username);

    return sqlite3_step(stmt) == SQLITE_ROW;
}

bool AdminDB::addAdmin(const std::string& username) {
    const char* sql = "INSERT INTO admins (username, password) VALUES
(?, ?);";
    Database::Statement stmt(db, sql);
    stmt.bind(1, username);
    stmt.bind(2, "admin123");
    return sqlite3_step(stmt) == SQLITE_DONE;
}

bool AdminDB::checkCredentials(const std::string& username, const
std::string& password) {
    const char* sql = "SELECT password FROM admins WHERE username = ?;";

    Database::Statement stmt(db, sql);
    stmt.bind(1, username);

    if (sqlite3_step(stmt) == SQLITE_ROW) {
        std::string stored_password = reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 0));
        return password == "admin123";
    }
```

```
    return false;
}

}  // namespace database
```

course_db.hpp

```
/**
* @file course_db.hpp
* @brief Работа с курсами в базе данных
*/

#pragma once

#include <vector>

#include "../classes/course.hpp"
#include "database.hpp"

namespace database {

/**
* @class CourseDB
* @brief Управление курсами в БД
*
* Реализует CRUD-операции для учебных курсов
*/
class CourseDB {
private:
 sqlite3* db;  ///< Указатель на соединение с БД

public:
 /**
  * @brief Конструктор, инициализирующий соединение
  * @param database Экземпляр базы данных
  */
 explicit CourseDB(const Database& database);

 /// @brief Деструктор по умолчанию
 ~CourseDB() = default;
```

```
    /**
     * @brief Добавить новый курс
     * @param course Объект курса для добавления
     * @return ID созданной записи
     */
    int addCourse(const course::Course& course);

    /**
     * @brief Получить курс по ID
     * @param id Идентификатор курса
     * @return Объект курса
     * @throws std::runtime_error если курс не найден
     */
    course::Course getCourse(int id);

    /**
     * @brief Проверить существование курса
     * @param courseId Идентификатор курса
     * @return true если курс существует
     */
    bool exists(int courseId);

    /**
     * @brief Получить все курсы
     * @return Вектор объектов курсов
     */
    std::vector<course::Course> getAllCourses();

};  // class CourseDB

}  // namespace database
```

course_db.cpp

```
#include "../../include/database/course_db.hpp"

namespace database {

CourseDB::CourseDB(const Database& database) : db(database.getHandle())
{
    const char* sql = R"(
```

```
        CREATE TABLE IF NOT EXISTS courses (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            institute TEXT NOT NULL
        );)";

    Database::Statement stmt(db, sql);
    sqlite3_step(stmt);
}

int CourseDB::addCourse(const course::Course& course) {
    const char* sql = "INSERT INTO courses (name, institute) VALUES (?,
?);";

    Database::Statement stmt(db, sql);
    stmt.bind(1, course.getName());
    stmt.bind(2, course.getInstitute());

    if (sqlite3_step(stmt) != SQLITE_DONE) {
        throw std::runtime_error("Failed to add course");
    }

    return sqlite3_last_insert_rowid(db);
}

course::Course CourseDB::getCourse(int id) {
    const char* sql = "SELECT * FROM courses WHERE id = ?;";

    Database::Statement stmt(db, sql);
    stmt.bind(1, id);

    if (sqlite3_step(stmt) == SQLITE_ROW) {
        return course::Course(sqlite3_column_int(stmt, 0),
reinterpret_cast<const char*>(sqlite3_column_text(stmt, 1)),
                              reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 2)));
    }

    throw std::runtime_error("Course not found");
}

bool CourseDB::exists(int courseId) {
    const char* sql = "SELECT 1 FROM courses WHERE id = ?;";
```

```
    Database::Statement stmt(db, sql);
    stmt.bind(1, courseId);

    return sqlite3_step(stmt) == SQLITE_ROW;
}

std::vector<course::Course> CourseDB::getAllCourses() {
    std::vector<course::Course> courses;
    const char* sql = "SELECT * FROM courses;";

    Database::Statement stmt(db, sql);

    while (sqlite3_step(stmt) == SQLITE_ROW) {
        courses.emplace_back(sqlite3_column_int(stmt, 0),
reinterpret_cast<const char*>(sqlite3_column_text(stmt, 1)),
                             reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 2)));
    }

    return courses;
}

}  // namespace database
```

database.hpp

```
/**
* @file database.hpp
* @brief Базовый класс работы с SQLite
*/

#pragma once

#include <sqlite3.h>

#include <memory>
#include <stdexcept>
#include <string>

namespace database {
```

```
/**
* @class Database
* @brief Обертка для работы с SQLite
*
* Обеспечивает:
* - Управление соединением
* - Выполнение запросов
* - Транзакции
*/
class Database {
private:
 sqlite3* db;                  ///< Указатель на БД
 static const char* DB_NAME;   ///< Имя файла БД

public:
 /// @brief Конструктор, открывающий соединение
 Database();

 /// @brief Деструктор, закрывающий соединение
 ~Database();

 Database(const Database&) = delete;
 Database& operator=(const Database&) = delete;
 Database(Database&&) = delete;
 Database& operator=(Database&&) = delete;

 /**
  * @class Statement
  * @brief Подготовленный SQL-запрос
  */
 class Statement {
   sqlite3_stmt* stmt;  ///< Указатель на подготовленный запрос

  public:
   /**
    * @brief Конструктор с подготовкой запроса
    * @param db Указатель на БД
    * @param sql SQL-запрос
    */
   Statement(sqlite3* db, const char* sql);

   /// @brief Деструктор с финализацией запроса
```

```
  ~Statement();

  /// @brief Неявное преобразование к sqlite3_stmt*
  operator sqlite3_stmt*() const { return stmt; }

  /**
   * @brief Привязать строковое значение
   * @param idx Номер параметра (1-based)
   * @param value Значение для привязки
   */
  void bind(int idx, const std::string& value);

  /**
   * @brief Привязать целочисленное значение
   * @param idx Номер параметра (1-based)
   * @param value Значение для привязки
   */
  void bind(int idx, int value);

};  // class Statement

/**
 * @class Transaction
 * @brief Управление транзакциями
 */
class Transaction {
  sqlite3* db;  ///< Указатель на БД

 public:
  /// @brief Начать транзакцию
  explicit Transaction(sqlite3* db);

  /// @brief Откат при разрушении
  ~Transaction();

  /// @brief Подтвердить транзакцию
  void commit();

  /// @brief Откатить транзакцию
  void rollback();

};  // class Transaction
```

```
 static Database& getInstance();

 /**
  * @brief Получить низкоуровневый дескриптор БД
  * @return Указатель на sqlite3
  */
 sqlite3* getHandle() const { return db; }

 /**
  * @brief Начать новую транзакцию
  * @return unique_ptr управляемого объекта транзакции
  */
 std::unique_ptr<Transaction> beginTransaction();

};  // class Database

}  // namespace database
```

database.cpp

```
#include "../../include/database/database.hpp"

namespace database {

const char* Database::DB_NAME = "university.db";

Database::Database() {
    if (sqlite3_open(DB_NAME, &db) != SQLITE_OK) {
        throw std::runtime_error(sqlite3_errmsg(db));
    }

    const char* pragma = "PRAGMA foreign_keys = ON;";
    sqlite3_exec(db, pragma, nullptr, nullptr, nullptr);
}

Database::~Database() {
    sqlite3_close(db);
}

Database& Database::getInstance() {
    static Database instance;
```

```
    return instance;
}

Database::Statement::Statement(sqlite3* db, const char* sql) {
    if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) {
        throw std::runtime_error(sqlite3_errmsg(db));
    }
}

Database::Statement::~Statement() {
    sqlite3_finalize(stmt);
}

void Database::Statement::bind(int idx, const std::string& value) {
    sqlite3_bind_text(stmt, idx, value.c_str(), -1, SQLITE_TRANSIENT);
}

void Database::Statement::bind(int idx, int value) {
    sqlite3_bind_int(stmt, idx, value);
}

Database::Transaction::Transaction(sqlite3* db) : db(db) {
    sqlite3_exec(db, "BEGIN TRANSACTION;", nullptr, nullptr, nullptr);
}

Database::Transaction::~Transaction() {
    if (db) rollback();
}

void Database::Transaction::commit() {
    sqlite3_exec(db, "COMMIT;", nullptr, nullptr, nullptr);
    db = nullptr;
}

void Database::Transaction::rollback() {
    sqlite3_exec(db, "ROLLBACK;", nullptr, nullptr, nullptr);
    db = nullptr;
}

std::unique_ptr<Database::Transaction> Database::beginTransaction() {
    return std::make_unique<Transaction>(db);
}
```

```
}  // namespace database
```

rating_db.hpp

```
/**
* @file rating_db.hpp
* @brief Работа с рейтингами в базе данных
*/

#pragma once

#include <utility>
#include <vector>

#include "../classes/rating.hpp"
#include "database.hpp"

namespace database {

/**
* @class RatingDB
* @brief Управление рейтингами курсов и преподавателей
*/
class RatingDB {
private:
 sqlite3* db;  ///< Указатель на соединение с БД

public:
 /// @brief Статистика оценок
 struct Statistics {
   std::vector<std::pair<int, double>> courseStats;  ///< Средние
оценки курсов
   std::vector<std::pair<int, double>>
       teacherStats;  ///< Средние оценки преподавателей
 };

 /**
  * @brief Конструктор, инициализирующий соединение
  * @param database Экземпляр базы данных
  */
 explicit RatingDB(const Database& database);
```

```
/// @brief Деструктор по умолчанию
~RatingDB() = default;

/**
 * @brief Добавить оценку курсу
 * @param studentId ID студента
 * @param courseId ID курса
 * @param rating Значение оценки (1-5)
 * @return true при успешном добавлении
 */
bool addCourseRating(int studentId, int courseId, int rating);

/**
 * @brief Добавить оценку преподавателю
 * @param studentId ID студента
 * @param teacherId ID преподавателя
 * @param rating Значение оценки (1-5)
 * @return true при успешном добавлении
 */
bool addTeacherRating(int studentId, int teacherId, int rating);

/**
 * @brief Получить все оценки
 * @return Вектор объектов Rating
 */
std::vector<rating::Rating> getAllRatings();

/**
 * @brief Получить статистику оценок
 * @return Структура Statistics с усредненными значениями
 */
Statistics getStatistics();

/**
 * @brief Проверить существование оценки
 * @param studentId ID студента
 * @param entityId ID сущности (курса/преподавателя)
 * @param isCourse Тип сущности
 * @return true если оценка уже существует
 */
bool hasExistingRating(int studentId, int entityId, bool isCourse);
```

```
    /// @name Методы получения оценок
    /// @{
    std::vector<std::pair<int, int>> getCourseRatings(int studentId);
    std::vector<std::pair<int, int>> getTeacherRatings(int studentId);
    std::pair<std::vector<std::pair<int, int>>, std::vector<std::pair<int,
int>>>
    getStudentRatings(int studentId);
    /// @}

};  // class RatingDB

}  // namespace database
```

rating_db.cpp

```
#include "../../include/database/rating_db.hpp"

namespace database {

RatingDB::RatingDB(const Database& database) : db(database.getHandle())
{
    const char* sql = R"(
        CREATE TABLE IF NOT EXISTS course_ratings (
            student_id INTEGER NOT NULL,
            course_id INTEGER NOT NULL,
            rating INTEGER CHECK(rating BETWEEN 1 AND 5),
            UNIQUE(student_id, course_id),
            FOREIGN KEY(student_id) REFERENCES students(id),
            FOREIGN KEY(course_id) REFERENCES courses(id)
        );)";

    Database::Statement stmt(db, sql);
    sqlite3_step(stmt);

    sql = R"(
        CREATE TABLE IF NOT EXISTS teacher_ratings (
            student_id INTEGER NOT NULL,
            teacher_id INTEGER NOT NULL,
            rating INTEGER CHECK(rating BETWEEN 1 AND 5),
            UNIQUE(student_id, teacher_id),
            FOREIGN KEY(student_id) REFERENCES students(id),
```

```
            FOREIGN KEY(teacher_id) REFERENCES teachers(id)
        );)";

    Database::Statement stmt2(db, sql);
    sqlite3_step(stmt2);
}

bool RatingDB::addTeacherRating(int studentId, int teacherId, int
rating) {
    const char* sql = "INSERT INTO teacher_ratings VALUES (?, ?, ?);";

    Database::Statement stmt(db, sql);
    stmt.bind(1, studentId);
    stmt.bind(2, teacherId);
    stmt.bind(3, rating);

    return sqlite3_step(stmt) == SQLITE_DONE;
}

bool RatingDB::hasExistingRating(int studentId, int entityId, bool
isCourse) {
    const char* sql;
    if (isCourse) {
        sql = "SELECT 1 FROM course_ratings WHERE student_id = ? AND
course_id = ?;";
    } else {
        sql =
            "SELECT 1 FROM teacher_ratings WHERE student_id = ? AND
teacher_id = "
            "?;";
    }

    Database::Statement stmt(db, sql);
    stmt.bind(1, studentId);
    stmt.bind(2, entityId);

    return sqlite3_step(stmt) == SQLITE_ROW;
}

RatingDB::Statistics RatingDB::getStatistics() {
    Statistics stats;

    const char* course_sql = R"(
```

```
        SELECT course_id, AVG(rating)
        FROM course_ratings
        GROUP BY course_id;)";

    Database::Statement course_stmt(db, course_sql);
    while (sqlite3_step(course_stmt) == SQLITE_ROW) {
        stats.courseStats.emplace_back(sqlite3_column_int(course_stmt,
0), sqlite3_column_double(course_stmt, 1));
    }

    const char* teacher_sql = R"(
        SELECT teacher_id, AVG(rating)
        FROM teacher_ratings
        GROUP BY teacher_id;)";

    Database::Statement teacher_stmt(db, teacher_sql);
    while (sqlite3_step(teacher_stmt) == SQLITE_ROW) {
        stats.teacherStats.emplace_back(sqlite3_column_int(teacher_stmt,
0), sqlite3_column_double(teacher_stmt, 1));
    }

    return stats;
}

std::vector<rating::Rating> RatingDB::getAllRatings() {
    std::vector<rating::Rating> ratings;

    const char* sql = R"(
        SELECT 'course' as type, student_id, course_id, rating FROM
course_ratings
        UNION ALL
        SELECT 'teacher', student_id, teacher_id, rating FROM
teacher_ratings
    )";

    Database::Statement stmt(db, sql);

    while (sqlite3_step(stmt) == SQLITE_ROW) {
        ratings.emplace_back(sqlite3_column_int(stmt, 1),
sqlite3_column_int(stmt, 2), sqlite3_column_int(stmt, 3),

                             std::string(reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 0))) == "course");
```

```
    }

    return ratings;
}

std::vector<std::pair<int, int>> RatingDB::getCourseRatings(int
studentId) {
    std::vector<std::pair<int, int>> ratings;
    const char* sql = "SELECT course_id, rating FROM course_ratings
WHERE student_id = ?;";

    Database::Statement stmt(db, sql);
    stmt.bind(1, studentId);

    while (sqlite3_step(stmt) == SQLITE_ROW) {
        ratings.emplace_back(sqlite3_column_int(stmt, 0),
sqlite3_column_int(stmt, 1));
    }

    return ratings;
}

std::vector<std::pair<int, int>> RatingDB::getTeacherRatings(int
studentId) {
    std::vector<std::pair<int, int>> ratings;
    const char* sql = "SELECT teacher_id, rating FROM teacher_ratings
WHERE student_id = ?;";

    Database::Statement stmt(db, sql);
    stmt.bind(1, studentId);

    while (sqlite3_step(stmt) == SQLITE_ROW) {
        ratings.emplace_back(sqlite3_column_int(stmt, 0),
sqlite3_column_int(stmt, 1));
    }

    return ratings;
}

std::pair<std::vector<std::pair<int, int>>, std::vector<std::pair<int,
int>>> RatingDB::getStudentRatings(
    int studentId) {
    return {getCourseRatings(studentId), getTeacherRatings(studentId)};
```

```
}

bool RatingDB::addCourseRating(int studentId, int courseId, int rating)
{
    const char* sql = R"(
        INSERT OR REPLACE INTO course_ratings
        (student_id, course_id, rating)
        VALUES (?, ?, ?);)";

    Database::Statement stmt(db, sql);
    stmt.bind(1, studentId);
    stmt.bind(2, courseId);
    stmt.bind(3, rating);

    return sqlite3_step(stmt) == SQLITE_DONE;
}

}  // namespace database
```

student_db.hpp

```
/**
* @file student_db.hpp
* @brief Работа со студентами в базе данных
*/

#pragma once

#include "../classes/student.hpp"
#include "database.hpp"

namespace database {

/**
* @class StudentDB
* @brief Управление студентами в БД
*/
class StudentDB {
private:
 sqlite3* db;  ///< Указатель на соединение с БД
```

```
public:
 /**
  * @brief Конструктор, инициализирующий соединение
  * @param database Экземпляр базы данных
  */
 explicit StudentDB(const Database& database);

 /// @brief Деструктор по умолчанию
 ~StudentDB() = default;

 /**
  * @brief Добавить студента
  * @param student Объект студента
  * @return ID созданной записи
  */
 int addStudent(const student::Student& student);

 /**
  * @brief Проверить существование по номеру зачетки
  * @param recordBook Номер зачетной книжки
  * @return true если студент существует
  */
 bool exists(const std::string& recordBook);

 /**
  * @brief Получить ID студента по зачетке
  * @param recordBook Номер зачетной книжки
  * @return ID студента или -1 если не найден
  */
 int getStudentIdByRecordBook(const std::string& recordBook);

};  // class StudentDB

}  // namespace database
```

student_db.cpp

```
#include "../../include/database/student_db.hpp"

namespace database {
```

```cpp
StudentDB::StudentDB(const Database& database) :
db(database.getHandle()) {
    const char* sql = R"(
        CREATE TABLE IF NOT EXISTS students (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            surname TEXT NOT NULL,
            name TEXT NOT NULL,
            patronymic TEXT,
            institute TEXT NOT NULL,
            department TEXT NOT NULL,
            course INTEGER NOT NULL,
            group_name TEXT NOT NULL,
            record_book INTEGER UNIQUE NOT NULL
        );)";

    Database::Statement stmt(db, sql);
    sqlite3_step(stmt);
}

int StudentDB::addStudent(const student::Student& student) {
    const char* sql = R"(
        INSERT INTO students
        (surname, name, patronymic, institute,
         department, course, group_name, record_book)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?);)";

    Database::Statement stmt(db, sql);
    stmt.bind(1, student.getSurname());
    stmt.bind(2, student.getName());
    stmt.bind(3, student.getPatronymic());
    stmt.bind(4, student.getInstitute());
    stmt.bind(5, student.getDepartment());
    stmt.bind(6, student.getCourse());
    stmt.bind(7, student.getGroup());
    stmt.bind(8, student.getRecordBook());

    if (sqlite3_step(stmt) != SQLITE_DONE) {
        throw std::runtime_error("Ошибка регистрации студента");
    }

    return sqlite3_last_insert_rowid(db);
}
```

```cpp
bool StudentDB::exists(const std::string& recordBook) {
    const char* sql = "SELECT 1 FROM students WHERE record_book = ?;";

    Database::Statement stmt(db, sql);
    stmt.bind(1, recordBook);

    return sqlite3_step(stmt) == SQLITE_ROW;
}

int StudentDB::getStudentIdByRecordBook(const std::string& recordBook)
{
    const char* sql = "SELECT id FROM students WHERE record_book = ?;";

    Database::Statement stmt(db, sql);
    stmt.bind(1, recordBook);

    if (sqlite3_step(stmt) == SQLITE_ROW) {
        return sqlite3_column_int(stmt, 0);
    }

    return -1;
}

}  // namespace database
```

teacher_db.hpp

```cpp
/**
* @file teacher_db.hpp
* @brief Работа с преподавателями в базе данных
*/

#pragma once

#include <vector>

#include "../classes/teacher.hpp"
#include "database.hpp"

namespace database {
```

```cpp
/**
* @class TeacherDB
* @brief Управление преподавателями в БД
*/
class TeacherDB {
private:
 sqlite3* db;  ///< Указатель на соединение с БД

public:
 /**
  * @brief Конструктор, инициализирующий соединение
  * @param database Экземпляр базы данных
  */
 explicit TeacherDB(const Database& database);

 /// @brief Деструктор по умолчанию
 ~TeacherDB() = default;

 /**
  * @brief Добавить преподавателя
  * @param teacher Объект преподавателя
  * @return ID созданной записи
  */
 int addTeacher(const teacher::Teacher& teacher);

 /**
  * @brief Проверить существование преподавателя
  * @param teacherId ID преподавателя
  * @return true если преподаватель существует
  */
 bool exists(int teacherId);

 /**
  * @brief Получить всех преподавателей
  * @return Вектор объектов Teacher
  */
 std::vector<teacher::Teacher> getAllTeachers();

 /**
  * @brief Получить преподавателя по ID
  * @param id Идентификатор преподавателя
  * @return Объект преподавателя
  * @throws std::runtime_error если не найден
```

```
  */
 teacher::Teacher getTeacher(int id);

};  // class TeacherDB

}  // namespace database
```

teacher_db.cpp

```
#include "../../include/database/teacher_db.hpp"

namespace database {

TeacherDB::TeacherDB(const Database& database) :
db(database.getHandle()) {
    const char* sql = R"(
        CREATE TABLE IF NOT EXISTS teachers (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            surname TEXT NOT NULL,
            name TEXT NOT NULL,
            patronymic TEXT,
            institute TEXT NOT NULL,
            department TEXT NOT NULL
        );)";

    Database::Statement stmt(db, sql);
    sqlite3_step(stmt);
}

int TeacherDB::addTeacher(const teacher::Teacher& teacher) {
    const char* sql = R"(
        INSERT INTO teachers
        (surname, name, patronymic, institute, department)
        VALUES (?, ?, ?, ?, ?);)";

    Database::Statement stmt(db, sql);
    stmt.bind(1, teacher.getSurname());
    stmt.bind(2, teacher.getName());
    stmt.bind(3, teacher.getPatronymic());
    stmt.bind(4, teacher.getInstitute());
    stmt.bind(5, teacher.getDepartment());
```

```
    if (sqlite3_step(stmt) != SQLITE_DONE) {
        throw std::runtime_error("Ошибка добавления преподавателя");
    }

    return sqlite3_last_insert_rowid(db);
}

bool TeacherDB::exists(int teacherId) {
    const char* sql = "SELECT 1 FROM teachers WHERE id = ?;";

    Database::Statement stmt(db, sql);
    stmt.bind(1, teacherId);

    return sqlite3_step(stmt) == SQLITE_ROW;
}

teacher::Teacher TeacherDB::getTeacher(int id) {
    const char* sql = "SELECT * FROM teachers WHERE id = ?;";

    Database::Statement stmt(db, sql);
    stmt.bind(1, id);

    if (sqlite3_step(stmt) == SQLITE_ROW) {
        return teacher::Teacher(sqlite3_column_int(stmt, 0),
                                reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 1)),
                                reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 2)),
                                reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 3)),
                                reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 4)),
                                reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 5)));
    }

    throw std::runtime_error("Преподаватель не найден");
}

std::vector<teacher::Teacher> TeacherDB::getAllTeachers() {
    std::vector<teacher::Teacher> teachers;
    const char* sql = "SELECT * FROM teachers;";
```

```
    Database::Statement stmt(db, sql);

    while (sqlite3_step(stmt) == SQLITE_ROW) {
        const char* patronymicPtr = reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 3));
        std::string patronymic = patronymicPtr ? patronymicPtr : "";

        teachers.emplace_back(sqlite3_column_int(stmt, 0),
reinterpret_cast<const char*>(sqlite3_column_text(stmt, 1)),
                              reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 2)), patronymic,
                              reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 4)),
                              reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 5)));
    }

    return teachers;
}

}  // namespace database
```

admin_menu.hpp

```
/**
* @file admin_menu.hpp
* @brief Меню администратора системы рейтингов
*/

#pragma once

#include "../database/database.hpp"
#include "colors.hpp"
namespace menu {
/**
 * @class AdminMenu
 * @brief Предоставляет функционал для администратора
 *
 * Включает:
 * - Просмотр всех оценок
```

```
 * - Статистику
 * - Управление курсами и преподавателями
 */
class AdminMenu {
 private:
  database::Database& db; ///< Ссылка на базу данных
 public:
  /**
   * @brief Конструктор, инициализирующий подключение к БД
   * @param database Ссылка на объект базы данных
   */
  explicit AdminMenu(database::Database& database);
  /// @brief Деструктор по умолчанию
  ~AdminMenu() = default;
  /**
   * @brief Главный цикл меню администратора
   */
  void show();
  /**
   * @brief Отобразить все оценки системы
   */
  void showAllRatings();
  /**
   * @brief Показать статистику оценок
   */
  void showStatistics();
  /**
   * @brief Добавить новый курс в систему
   */
  void addCourse();
  /**
   * @brief Добавить нового преподавателя
   */
  void addTeacher();
};  // class AdminMenu
}  // namespace menu
```

admin_menu.cpp

```
#include "../../include/ui/admin_menu.hpp"
```

```cpp
#include <iostream>

#include "../../include/database/course_db.hpp"
#include "../../include/database/rating_db.hpp"
#include "../../include/database/teacher_db.hpp"

namespace menu {

AdminMenu::AdminMenu(database::Database& database) : db(database) {}

void AdminMenu::showAllRatings() {
    database::RatingDB ratingDB(db);
    auto ratings = ratingDB.getAllRatings();

    std::cout << BOLD << YELLOW << "\n=== Все оценки ===\n" << RESET;
    for (const auto& r : ratings) {
        std::cout << "Студент ID: " << r.getStudentId() << " | "
                  << (r.isCourseRating() ? "Курс ID: " : "Преподаватель
ID: ") << r.getEntityId()
                  << " | Оценка: " << r.getValue() << "/5\n";
    }
}

void AdminMenu::showStatistics() {
    database::RatingDB ratingDB(db);
    database::CourseDB courseDB(db);
    database::TeacherDB teacherDB(db);

    auto stats = ratingDB.getStatistics();

    std::cout << BOLD << CYAN << "\n=== Статистика по курсам ===\n" <<
RESET;
    for (const auto& [courseId, avg] : stats.courseStats) {
        try {
            auto course = courseDB.getCourse(courseId);
            std::cout << course.getName() << " (ID: " << courseId << "):
" << avg << "/5\n";
        } catch (...) {
            std::cout << "Курс ID:" << courseId << " (удален): " << avg
<< "/5\n";
        }
    }
```

```
    std::cout << BOLD << CYAN << "\n=== Статистика по преподавателям
===\n" << RESET;
    for (const auto& [teacherId, avg] : stats.teacherStats) {
        try {
            auto teacher = teacherDB.getTeacher(teacherId);
            std::cout << teacher.getFullName() << " (ID: " << teacherId
<< "): " << avg << "/5\n";
        } catch (...) {
            std::cout << "Преподаватель ID:" << teacherId << " (удален):
" << avg << "/5\n";
        }
    }
}

void AdminMenu::show() {
    int choice;
    while (true) {
        std::cout << BOLD << MAGENTA << "\nАдминистраторское меню:\n"
                  << RESET << "1. Показать все оценки\n"
                  << "2. Статистика оценок\n"
                  << "3. Добавить курс\n"
                  << "4. Добавить преподавателя\n"
                  << "0. Выход\n"
                  << "Выбор: ";
        std::cin >> choice;
        std::cin.ignore();

        try {
            switch (choice) {
                case 1:
                    showAllRatings();
                    break;
                case 2:
                    showStatistics();
                    break;
                case 3:
                    addCourse();
                    break;
                case 4:
                    addTeacher();
                    break;
                case 0:
```

```
                    return;
                default:
                    std::cout << RED << "Неверный выбор!\n" << RESET;
            }
        } catch (const std::exception& e) {
            std::cout << RED << "Ошибка: " << e.what() << RESET << "\n";
        }
    }
}

void AdminMenu::addCourse() {
    std::cout << BOLD << CYAN << "\nДобавление нового курса\n" << RESET;

    std::string name, institute;
    std::cout << "Название курса: ";
    std::getline(std::cin, name);

    std::cout << "Институт: ";
    std::getline(std::cin, institute);

    try {
        database::CourseDB courseDB(db);
        int id = courseDB.addCourse(course::Course(0, name, institute));
        std::cout << GREEN << "Курс добавлен! ID: " << id << RESET <<
"\n";
    } catch (const std::exception& e) {
        std::cout << RED << "Ошибка: " << e.what() << RESET << "\n";
    }
}

void AdminMenu::addTeacher() {
    std::cout << BOLD << CYAN << "\nДобавление преподавателя\n" <<
RESET;

    std::string surname, name, patronymic, institute, department;
    std::cout << "Фамилия: ";
    std::getline(std::cin, surname);

    std::cout << "Имя: ";
    std::getline(std::cin, name);

    std::cout << "Отчество: ";
    std::getline(std::cin, patronymic);
```

```
    std::cout << "Институт: ";
    std::getline(std::cin, institute);

    std::cout << "Кафедра: ";
    std::getline(std::cin, department);

    try {
        database::TeacherDB teacherDB(db);
        int id = teacherDB.addTeacher(teacher::Teacher(0, surname, name,
patronymic, institute, department));
        std::cout << GREEN << "Преподаватель добавлен! ID: " << id <<
RESET << "\n";
    } catch (const std::exception& e) {
        std::cout << RED << "Ошибка: " << e.what() << RESET << "\n";
    }
}

}  // namespace menu
```

colors.hpp

```
/**
* @file colors.hpp
* @brief ANSI-коды для цветного вывода в консоль
*
* @def RESET Сброс цвета
* @def RED Красный цвет
* @def GREEN Зеленый цвет
* @def YELLOW Желтый цвет
* @def BLUE Синий цвет
* @def MAGENTA Пурпурный цвет
* @def CYAN Голубой цвет
* @def BOLD Жирный шрифт
*/

#pragma once

#define RESET "\033[0m"     ///< Сброс стилей
#define RED "\033[31m"      ///< Текст красного цвета
#define GREEN "\033[32m"    ///< Текст зеленого цвета
#define YELLOW "\033[33m"   ///< Текст желтого цвета
#define BLUE "\033[34m"     ///< Текст синего цвета
```

```
#define MAGENTA "\033[35m" ///< Текст пурпурного цвета
#define CYAN "\033[36m"    ///< Текст голубого цвета
#define BOLD "\033[1m"     ///< Жирное начертание
```

login_menu.hpp

```
/**
* @file login_menu.hpp
* @brief Меню авторизации пользователей
*/

#pragma once

#include "../database/admin_db.hpp"
#include "../database/database.hpp"
#include "../database/student_db.hpp"
#include "colors.hpp"
namespace menu {
/**
 * @class LoginMenu
 * @brief Обрабатывает вход студентов и администраторов
 */
class LoginMenu {
 private:
  database::Database& db; ///< Ссылка на базу данных
 public:
  /**
   * @brief Конструктор, инициализирующий подключение к БД
   * @param database Ссылка на объект базы данных
   */
  LoginMenu(database::Database& database);
  /// @brief Деструктор по умолчанию
  ~LoginMenu() = default;
  /**
   * @brief Авторизация студента
   * @return ID студента или -1 при ошибке
   */
  int loginStudent();
  /**
   * @brief Авторизация администратора
   * @return true при успешной аутентификации
```

```
    */
    bool loginAdmin();
};
}  // namespace menu
```

login_menu.cpp

```
#include "../../include/ui/login_menu.hpp"

#include <iostream>

#include "../../include/database/admin_db.hpp"
#include "../../include/database/student_db.hpp"

namespace menu {

LoginMenu::LoginMenu(database::Database& database) : db(database) {}

int LoginMenu::loginStudent() {
    std::string recordBook;
    std::cout << CYAN << "Введите номер зачетки: " << RESET;
    std::getline(std::cin, recordBook);

    database::StudentDB studentDB(db);
    if (studentDB.exists(recordBook)) {
        return studentDB.getStudentIdByRecordBook(recordBook);
    }
    std::cout << RED << "Студент не найден!\n" << RESET;
    return -1;
}

bool LoginMenu::loginAdmin() {
    std::string username;
    std::cout << CYAN << "Логин: " << RESET;

    std::getline(std::cin, username);

    std::string password;
    std::cout << CYAN << "Пароль: " << RESET;

    std::getline(std::cin, password);
```

```
    database::AdminDB adminDB(db);

    return adminDB.checkCredentials(username, password);
}

}  // namespace menu
```

main_menu.hpp

```
/**
* @file main_menu.hpp
* @brief Главное меню приложения
*/

#pragma once

#include "../database/database.hpp"
#include "colors.hpp"
namespace menu {
/**
 * @class MainMenu
 * @brief Центральный узел навигации приложения
 *
 * Обеспечивает переходы:
 * - Вход/регистрация
 * - Меню студента
 * - Меню администратора
 */
class MainMenu {
 private:
  database::Database& db; ///< Ссылка на базу данных
 public:
  /**
   * @brief Конструктор, инициализирующий подключение к БД
   * @param database Ссылка на объект базы данных
   */
  explicit MainMenu(database::Database& database);
  /// @brief Деструктор по умолчанию
  ~MainMenu() = default;
  /**
```

```
    * @brief Главный цикл приложения
    */
   void run();
};  // class MainMenu
}  // namespace menu
```

main_menu.cpp

```
#include "../../include/ui/main_menu.hpp"

#include <iostream>

#include "../../include/ui/admin_menu.hpp"
#include "../../include/ui/login_menu.hpp"
#include "../../include/ui/registration_menu.hpp"
#include "../../include/ui/student_menu.hpp"

namespace menu {

MainMenu::MainMenu(database::Database& database) : db(database) {}

void MainMenu::run() {
    int choice;
    while (true) {
        std::cout << BOLD << CYAN << "\nГлавное меню:\n"
                  << RESET << "1. Вход студента\n"
                  << "2. Вход администратора\n"
                  << "3. Регистрация студента\n"
                  << "0. Выход\n"
                  << "Выбор: ";

        std::cin >> choice;
        std::cin.ignore();

        try {
            switch (choice) {
                case 1: {
                    LoginMenu login(db);
                    int studentId = login.loginStudent();
                    if (studentId != -1) {
                        StudentMenu studentMenu(db, studentId);
```

```
                    studentMenu.show();
                }
                break;
            }

            case 2: {
                LoginMenu login(db);
                if (login.loginAdmin()) {
                    AdminMenu adminMenu(db);
                    adminMenu.show();
                } else {
                    std::cout << RED << "Неверные данные!\n" <<
RESET;
                }
                break;
            }

            case 3: {
                RegistrationMenu reg(db);
                int id = reg.registerStudent();
                if (id != -1) {
                    std::cout << GREEN << "Студент зарегистрирован!
ID: " << id << RESET << "\n";
                }
                break;
            }

            case 0:
                return;

            default:
                std::cout << RED << "Неверный выбор!\n" << RESET;
        }
    } catch (const std::exception& e) {
        std::cout << RED << "Ошибка: " << e.what() << RESET << "\n";
    }
  }
}

}  // namespace menu
```

registration_menu.hpp

```cpp
/**
* @file registration_menu.hpp
* @brief Меню регистрации новых студентов
*/

#pragma once

#include "../database/database.hpp"
#include "colors.hpp"
namespace menu {
/**
 * @class RegistrationMenu
 * @brief Регистрирует новых студентов в системе
 */
class RegistrationMenu {
 private:
  database::Database& db; ///< Ссылка на базу данных
 public:
  /**
   * @brief Конструктор, инициализирующий подключение к БД
   * @param database Ссылка на объект базы данных
   */
  explicit RegistrationMenu(database::Database& database);
  /// @brief Деструктор по умолчанию
  ~RegistrationMenu() = default;
  /**
   * @brief Процесс регистрации студента
   * @return ID нового студента или -1 при ошибке
   */
  int registerStudent();
};  // class RegistrationMenu
}  // namespace menu
```

registration_menu.cpp

```cpp
#include "../../include/ui/registration_menu.hpp"

#include <iostream>
```

```cpp
#include "../../include/classes/student.hpp"
#include "../../include/database/student_db.hpp"

namespace menu {

RegistrationMenu::RegistrationMenu(database::Database& database) :
db(database) {}

int RegistrationMenu::registerStudent() {
   std::string surname;
   std::string name;
   std::string patronymic;
   std::string institute;
   std::string department;
   std::string group;
   std::string recordBook;
   int course;

   std::cout << BOLD << CYAN << "\n=== Регистрация студента ===\n" <<
RESET;

   std::cout << "Фамилия: ";
   std::getline(std::cin, surname);

   std::cout << "Имя: ";
   std::getline(std::cin, name);

   std::cout << "Отчество: ";
   std::getline(std::cin, patronymic);

   std::cout << "Институт: ";
   std::getline(std::cin, institute);

   std::cout << "Кафедра: ";
   std::getline(std::cin, department);

   std::cout << "Курс: ";
   std::cin >> course;

   std::cin.ignore();

   std::cout << "Группа: ";
   std::getline(std::cin, group);
```

```
    std::cout << "Номер зачетки: ";
    std::getline(std::cin, recordBook);

    try {
        database::StudentDB studentDB(db);

        if (studentDB.exists(recordBook)) {
            std::cout << RED << "Ошибка: Зачетная книжка уже
зарегистрирована!\n" << RESET;
            return -1;
        }

        student::Student newStudent(0, surname, name, patronymic,
institute, department, course, group, recordBook);

        return studentDB.addStudent(newStudent);
    } catch (const std::exception& e) {
        std::cout << RED << "Ошибка: " << e.what() << RESET << "\n";
        return -1;
    }
}

}  // namespace menu
```

student_menu.hpp

```
/**
* @file student_menu.hpp
* @brief Меню студента
*/

#pragma once

#include "../database/database.hpp"
#include "colors.hpp"
namespace menu {
/**
 * @class StudentMenu
 * @brief Функционал доступный студенту
 *
```

```
 * Включает:
 * - Оценку курсов и преподавателей
 * - Просмотр своих оценок
 */
class StudentMenu {
 private:
  database::Database& db; ///< Ссылка на базу данных
  int studentId;          ///< ID текущего студента
  /**
   * @brief Печатает заголовок раздела
   * @param title Текст заголовка
   */
  void printHeader(const std::string& title);
 public:
  /**
   * @brief Конструктор, инициализирующий сессию студента
   * @param database Ссылка на БД
   * @param sid ID студента
   */
  StudentMenu(database::Database& database, int sid);
  /// @brief Деструктор по умолчанию
  ~StudentMenu() = default;
  /**
   * @brief Главный цикл меню студента
   */
  void show();
  /**
   * @brief Оценить курс
   */
  void rateCourse();
  /**
   * @brief Оценить преподавателя
   */
  void rateTeacher();
  /**
   * @brief Просмотр своих оценок
   */
  void viewRatings();
};  // class StudentMenu
}  // namespace menu
```

student_menu.cpp

```cpp
#include "../../include/ui/student_menu.hpp"

#include <iostream>

#include "../../include/database/course_db.hpp"
#include "../../include/database/rating_db.hpp"
#include "../../include/database/teacher_db.hpp"

namespace menu {

StudentMenu::StudentMenu(database::Database& database, int sid) :
db(database), studentId(sid) {}

void StudentMenu::printHeader(const std::string& title) {
    std::cout << BOLD << CYAN << "\n=== " << title << " ===\n" << RESET;
}

void StudentMenu::rateCourse() {
    printHeader("Оценка курса");
    database::CourseDB courseDB(db);
    auto courses = courseDB.getAllCourses();

    for (const auto& course : courses) {
        std::cout << course.getId() << ". " << course.getName() << "\n";
    }

    int courseId, rating;
    std::cout << "ID курса: ";
    std::cin >> courseId;

    database::RatingDB ratingDB(db);
    bool hasExistingRating = ratingDB.hasExistingRating(studentId,
courseId, true);

    std::cout << "Оценка (1-5): ";
    std::cin >> rating;

    if (rating < 1 || rating > 5) {
        std::cout << RED << "Некорректная оценка!\n" << RESET;
        return;
    }
```

```
    if (ratingDB.addCourseRating(studentId, courseId, rating)) {
        std::cout << GREEN << "Оценка " << (hasExistingRating ?
"обновлена" : "добавлена") << "!\n" << RESET;
    } else {
        std::cout << RED << "Ошибка!\n" << RESET;
    }
}

void StudentMenu::rateTeacher() {
    printHeader("Оценка преподавателя");
    database::TeacherDB teacherDB(db);
    auto teachers = teacherDB.getAllTeachers();

    for (const auto& teacher : teachers) {
        std::cout << teacher.getId() << ". " << teacher.getFullName() <<
"\n";
    }

    int teacherId, rating;
    std::cout << "ID преподавателя: ";
    std::cin >> teacherId;

    database::RatingDB ratingDB(db);
    bool hasExistingRating = ratingDB.hasExistingRating(studentId,
teacherId, false);

    std::cout << "Оценка (1-5): ";
    std::cin >> rating;

    if (rating < 1 || rating > 5) {
        std::cout << RED << "Некорректная оценка!\n" << RESET;
        return;
    }

    if (ratingDB.addTeacherRating(studentId, teacherId, rating)) {
        std::cout << GREEN << "Оценка " << (hasExistingRating ?
"обновлена" : "добавлена") << "!\n" << RESET;
    } else {
        std::cout << RED << "Ошибка!\n" << RESET;
    }
}
```

```
void StudentMenu::viewRatings() {
    database::RatingDB ratingDB(db);

    auto [courseRatings, teacherRatings] =
ratingDB.getStudentRatings(studentId);

    std::cout << YELLOW << "\nОценки за курсы:\n" << RESET;

    for (const auto& [courseId, rating] : courseRatings) {
        std::cout << "Курс ID: " << courseId << ": " << rating <<
"/5\n";
    }

    std::cout << YELLOW << "\nОценки преподавателей:\n" << RESET;

    for (const auto& [teacherId, rating] : teacherRatings) {
        std::cout << "Преподаватель ID: " << teacherId << ": " << rating
<< "/5\n";
    }
}

void StudentMenu::show() {
    int choice;
    while (true) {
        std::cout << BOLD << MAGENTA << "\nМеню студента:\n"
                  << RESET << "1. Оценить курс\n"
                  << "2. Просмотреть оценки\n"
                  << "3. Оценить преподавателя\n"
                  << "0. Выход\n"
                  << "Выбор: ";

        std::cin >> choice;
        std::cin.ignore();

        try {
            switch (choice) {
                case 1:
                    rateCourse();
                    break;
                case 2:
                    viewRatings();
                    break;
                case 3:
```

```
                    rateTeacher();
                    break;
                case 0:
                    return;
                default:
                    std::cout << RED << "Неверный выбор!\n" << RESET;
            }
        } catch (const std::exception& e) {
            std::cout << RED << "Ошибка: " << e.what() << RESET << "\n";
        }
    }
}

}  // namespace menu
```

main.cpp

```
#include <iostream>

#include "../include/database/database.hpp"
#include "../include/ui/main_menu.hpp"

int main() {
    try {
        database::Database db;

        menu::MainMenu mainMenu(db);
        mainMenu.run();

        std::cout << GREEN << "Программа завершена." << RESET <<
std::endl;

        return 0;
    } catch (const std::exception& e) {
        std::cerr << RED << "\nКритическая ошибка: " << e.what() <<
RESET << std::endl;
        return 1;
    }
}
```

test_admin_db.cpp

```cpp
#include "../include/database/admin_db.hpp"
#include "../include/database/database.hpp"
#include <gtest/gtest.h>
#include <sqlite3.h>

class AdminDBTest : public ::testing::Test {
protected:
 void SetUp() override {
   db = std::make_unique<database::Database>();
   adminDB = std::make_unique<database::AdminDB>(*db);
 }

 void TearDown() override {
   sqlite3_exec(db->getHandle(), "DELETE FROM admins;", nullptr,
nullptr,
                nullptr);
 }

 std::unique_ptr<database::Database> db;
 std::unique_ptr<database::AdminDB> adminDB;
};

TEST_F(AdminDBTest, InitialAdminCreated) {
 EXPECT_TRUE(adminDB->checkCredentials("admin", "admin123"));
}

TEST_F(AdminDBTest, AddNewAdmin) {
 EXPECT_TRUE(adminDB->addAdmin("newadmin"));
 EXPECT_TRUE(adminDB->checkCredentials("newadmin", "admin123"));
}

TEST_F(AdminDBTest, CheckInvalidCredentials) {
 EXPECT_FALSE(adminDB->checkCredentials("admin", "wrongpass"));
 EXPECT_FALSE(adminDB->checkCredentials("nonexistent", "admin123"));
}

TEST_F(AdminDBTest, PreventDuplicateAdmins) {
 EXPECT_TRUE(adminDB->addAdmin("testadmin"));
 EXPECT_FALSE(adminDB->addAdmin("testadmin"));
}
```

test_course_db.cpp

```
#include "../include/database/course_db.hpp"
#include "../include/database/database.hpp"
#include <gtest/gtest.h>

class CourseDBTest : public ::testing::Test {
protected:
 void SetUp() override {
   db = std::make_unique<database::Database>();
   courseDB = std::make_unique<database::CourseDB>(*db);
 }

 void TearDown() override {
   sqlite3_exec(db->getHandle(), "DELETE FROM courses;", nullptr,
nullptr,
                nullptr);
 }

 std::unique_ptr<database::Database> db;
 std::unique_ptr<database::CourseDB> courseDB;
};

TEST_F(CourseDBTest, AddAndRetrieveCourse) {
 int id = courseDB->addCourse({1, "Test Course", "Computer Science"});
 auto course = courseDB->getCourse(id);

 EXPECT_EQ(course.getName(), "Test Course");
 EXPECT_EQ(course.getInstitute(), "Computer Science");
}

TEST_F(CourseDBTest, CourseExistenceCheck) {
 int id = courseDB->addCourse({2, "Math", "Mathematics Dept"});
 EXPECT_TRUE(courseDB->exists(id));
 EXPECT_FALSE(courseDB->exists(9999));
}

TEST_F(CourseDBTest, GetAllCourses) {
 courseDB->addCourse({3, "Course1", "Dept1"});
 courseDB->addCourse({4, "Course2", "Dept2"});
```

```
 auto courses = courseDB->getAllCourses();
 ASSERT_EQ(courses.size(), 5);
}
```

test_rating_db.cpp

```
#include "../include/database/course_db.hpp"
#include "../include/database/database.hpp"
#include "../include/database/rating_db.hpp"
#include "../include/database/student_db.hpp"
#include "../include/database/teacher_db.hpp"
#include <gtest/gtest.h>

class RatingDBTest : public ::testing::Test {
protected:
 void SetUp() override {
   db = std::make_unique<database::Database>();

   studentDB = std::make_unique<database::StudentDB>(*db);
   courseDB = std::make_unique<database::CourseDB>(*db);
   teacherDB = std::make_unique<database::TeacherDB>(*db);
   ratingDB = std::make_unique<database::RatingDB>(*db);

   studentId =
       studentDB->addStudent({"Ivanov", "Ivan", "Ivanovich",
"Institute",
                              "Department", 3, "Group", "12345"});

   courseId = courseDB->addCourse({1, "Math", "Mathematics"});
   teacherId = teacherDB->addTeacher(
       {1, "Petrov", "Petr", "Petrovich", "Mathematics", "Math
Department"});
 }

 std::unique_ptr<database::Database> db;
 std::unique_ptr<database::StudentDB> studentDB;
 std::unique_ptr<database::CourseDB> courseDB;
 std::unique_ptr<database::TeacherDB> teacherDB;
 std::unique_ptr<database::RatingDB> ratingDB;
```

```
 int studentId;
 int courseId;
 int teacherId;
};

TEST_F(RatingDBTest, AddAndRetrieveRatings) {
 EXPECT_TRUE(ratingDB->addCourseRating(studentId, courseId, 5));
 EXPECT_TRUE(ratingDB->addTeacherRating(studentId, teacherId, 4));

 auto ratings = ratingDB->getAllRatings();
 ASSERT_EQ(ratings.size(), 2);
 EXPECT_EQ(ratings[0].getValue(), 5);
 EXPECT_EQ(ratings[1].getValue(), 4);
}
```

.clang-format

```
---
BasedOnStyle: Google
AllowShortFunctionsOnASingleLine: Empty
AlignAfterOpenBracket: Align
BreakBeforeBraces: Attach
ColumnLimit: 120
IndentWidth: 4
```

Makefile

```
## @file
# @brief Makefile системы рейтингов университета

################################################################################
########
#                            Настройки компиляции
#
################################################################################
########

CXX := g++
CXXFLAGS := -std=c++17 -Wall -Wextra -Wpedantic -Iinclude
```

```
LDFLAGS := -lsqlite3 -lpthread
DEBUG_FLAGS := -g -O0
RELEASE_FLAGS := -O3
COVERAGE_FLAGS := -fprofile-arcs -ftest-coverage

################################################################################
########
#                           Структура проекта
#
################################################################################
########

BUILD_DIR := build
SRC_DIR := src
TARGET := university_app
DOCS_DIR := docs
REPORT_DIR := report
DIST_DIR := dist
TEST_DIR := tests
DOXYFILE := Doxyfile
TEST_DB := $(TEST_DIR)/test.db

################################################################################
########
#                           Настройки установки
#
################################################################################
########

BIN_DIR := bin
DESTDIR :=

################################################################################
########
#                           Вспомогательные переменные
#
################################################################################
########

SOURCES := $(shell find $(SRC_DIR) -name '*.cpp')
OBJECTS := $(SOURCES:$(SRC_DIR)/%.cpp=$(BUILD_DIR)/%.o)
TEST_SOURCES := $(shell find $(TEST_DIR) -name '*.cpp')
TEST_OBJECTS := $(TEST_SOURCES:$(TEST_DIR)/%.cpp=$(BUILD_DIR)/%.o)
```

```
TEST_TARGET := $(BUILD_DIR)/university_tests

################################################################################
########
#                              Основные цели
#
################################################################################
########

.PHONY: all install uninstall clean distclean test gcov_report docs

all: build

build: CXXFLAGS += $(RELEASE_FLAGS)
build: $(BUILD_DIR)/$(TARGET)

debug: CXXFLAGS += $(DEBUG_FLAGS)
debug: build

################################################################################
########
#                              Установка и очистка
#
################################################################################
########

install: build
    @mkdir -p $(BIN_DIR)
    cp $(BUILD_DIR)/$(TARGET) $(BIN_DIR)/
    zip -j $(BIN_DIR)/$(TARGET).zip $(BIN_DIR)/$(TARGET)

uninstall:
    rm -rf $(BIN_DIR)

clean:
    rm -rf $(BUILD_DIR) *.gcov *.gcda *.gcno $(TEST_DB) $(REPORT_DIR)
    @if [ -f university.db ]; then \
        echo "Removing existing database..."; \
        rm university.db; \
    else \
        echo "Database file not found, skipping..."; \
    fi
```

```
distclean: clean
	rm -rf $(DIST_DIR) $(DOCS_DIR) coverage.info

################################################################################
#                            Тестирование и покрытие
#
################################################################################

test: CXXFLAGS += $(COVERAGE_FLAGS) -DTEST_DB_PATH=\"$(TEST_DB)\"
test: LDFLAGS += $(COVERAGE_FLAGS) -lgtest -lgtest_main
test: $(TEST_TARGET)
	@mkdir -p $(TEST_DIR)
	$(TEST_TARGET) --gtest_output=xml:$(BUILD_DIR)/test_results.xml

gcov_report: test
	lcov --capture --directory $(BUILD_DIR) --output-file coverage.info
--ignore-errors mismatch
	lcov --remove coverage.info '/usr/*' '$(TEST_DIR)/*' --output-file
coverage.info
	genhtml coverage.info --output-directory $(REPORT_DIR)/coverage

################################################################################
#                            Документирование
#
################################################################################

docs:
	doxygen $(DOXYFILE)

################################################################################
#                            Форматирование кода
#
################################################################################

format:
	find $(SRC_DIR) include $(TEST_DIR) -name '*.hpp' -o -name '*.cpp'
-exec clang-format -i -style=file {} \;
```

```
check-style:
	find $(SRC_DIR) include $(TEST_DIR) -name '*.hpp' -o -name '*.cpp'
-exec clang-format --dry-run --Werror -style=file {} \;

################################################################################
########
#                              Дистрибутив
#
################################################################################
########

dist: distclean
	@mkdir -p $(DIST_DIR)
	tar -czvf $(DIST_DIR)/$(TARGET)_$(shell date +%Y%m%d).tar.gz \
	--transform "s,^,$(TARGET)/," --exclude=".*" *

################################################################################
########
#                              Системные правила
#
################################################################################
########

-include $(OBJECTS:.o=.d)
-include $(TEST_OBJECTS:.o=.d)

$(BUILD_DIR)/$(TARGET): $(OBJECTS)
	@mkdir -p $(@D)
	$(CXX) $(CXXFLAGS) $^ -o $@ $(LDFLAGS)

$(TEST_TARGET): $(TEST_OBJECTS) $(filter-out $(BUILD_DIR)/main.o,
$(OBJECTS))
	@mkdir -p $(@D)
	$(CXX) $(CXXFLAGS) $^ -o $@ $(LDFLAGS)

$(BUILD_DIR)/%.o: $(SRC_DIR)/%.cpp
	@mkdir -p $(@D)
	$(CXX) $(CXXFLAGS) -MMD -MP -c $< -o $@

$(BUILD_DIR)/%.o: $(TEST_DIR)/%.cpp
	@mkdir -p $(@D)
	$(CXX) $(CXXFLAGS) -I$(TEST_DIR) -MMD -MP -c $< -o $@
```

cpp-ci.yml

```yaml
name: C++ CI/CD

on:
 push:
   branches: [ "main" ]
 pull_request:
   branches: [ "main" ]

jobs:
 build-and-check:
   runs-on: ubuntu-latest
   steps:
   - name: Checkout code
     uses: actions/checkout@v4

   - name: Install dependencies
     run: |
       sudo apt-get update
       sudo apt-get install -y g++ make sqlite3 libsqlite3-dev
clang-format libgtest-dev lcov

   - name: Build project
     run: make

   - name: Check code style
     run: make check-style

   - name: Run tests
     run: make test

   - name: Generate coverage report
     run: |
       make clean
       make gcov_report
       echo "Coverage report available in report/coverage/index.html"

 clang-format-check:
   runs-on: ubuntu-latest
```

```
    steps:
      - uses: actions/checkout@v4

      - name: Check formatting
        run: |
          find $(SRC_DIR) include tests -name '*.hpp' -o -name '*.cpp'
-exec clang-format --dry-run --Werror {} \;
```

build.sh

```
#!/usr/bin/env bash


RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m'

header() {
    echo -e "${YELLOW}"
    echo "╔═══════════════════════════════════════════════╗"
    echo "╠══ Building University Rating System ═══╣"
    echo "╚═══════════════════════════════════════════════╝"
    echo -e "${NC}"
}

check_dependencies() {
    cd ..
    local deps=("g++" "make" "sqlite3" "clang-format")
    for dep in "${deps[@]}"; do
        if ! command -v $dep &> /dev/null; then
            echo -e "${RED}Error: $dep not found!${NC}"
            exit 1
        fi
    done
}

main() {
    header
    check_dependencies
```

```
    echo -e "${GREEN}[1/3] Cleaning previous build...${NC}"
    make clean

    echo -e "${GREEN}[2/3] Compiling project...${NC}"
    if ! make build; then
        echo -e "${RED}Build failed!${NC}"
        exit 1
    fi

    echo -e "${GREEN}[3/3] Verifying binary...${NC}"
    if [ -f "build/university_app" ]; then
        echo -e "${GREEN}✅ Build successful!${NC}"
    else
        echo -e "${RED}❌ Binary not found!${NC}"
        exit 1
    fi
}

main
```

clean.sh

```
#!/usr/bin/env bash


RED='\033[0;31m'
NC='\033[0m'

prompt_clean() {
    echo -e "${RED}"
    read -p "⚠️  This will delete all build artifacts! Continue? [y/N] "
-n 1 -r
    echo -e "${NC}"

    if [[ $REPLY =~ ^[Yy]$ ]]; then
        cd ..
        echo "🧹 Cleaning project..."
        make clean
        echo "✅ Project cleaned"
    else
        echo "🛑 Clean cancelled"
```

```
    fi
}

main() {
    prompt_clean
}

main
```

docs.sh

```
#!/usr/bin/env bash


PURPLE='\033[0;35m'
NC='\033[0m'

generate_docs() {
    echo -e "${PURPLE}"
    echo "📚 Generating documentation..."
    echo -e "${NC}"

    doxygen Doxyfile
    echo -e "\nDocumentation saved to: docs/html/"
}

open_docs() {
    if [ -f "docs/html/index.html" ]; then
        xdg-open docs/html/index.html
    else
        echo "Docs not found! Generating first..."
        generate_docs
        xdg-open docs/html/index.html
    fi
}

main() {
    generate_docs
    open_docs
}
```

```
main
```

install.sh

```
#!/usr/bin/env bash


GREEN='\033[0;32m'
NC='\033[0m'

install_app() {
   cd ..
   echo -e "${GREEN}🔧 Installing University Rating System...${NC}"

   if [ ! -f "build/university_app" ]; then
       echo "Building project first..."
       ./build.sh
   fi

   echo -e "\n${GREEN}🎉 Installation complete!"
   echo "Run with: ${NC}urs"
}

main() {
   install_app
}

main
```

run_coverage.sh

```
#!/usr/bin/env bash


CYAN='\033[0;36m'
GREEN='\033[0;32m'
NC='\033[0m'

show_coverage() {
   echo -e "${CYAN}"
```

```
    echo "╔═══════════════════════════════════════╗"
    echo "╠══ Code Coverage Report ═══════════════╣"
    echo "╚═══════════════════════════════════════╝"
    echo -e "${NC}"

    if [ -d "docs/coverage" ]; then
        echo -e "${GREEN}Opening report...${NC}"
        xdg-open docs/coverage/index.html
    else
        cd ..
        echo "Generating report..."
        make gcov_report
        xdg-open docs/coverage/index.html
    fi
}

main() {
    make gcov_report
    show_coverage
}

main
```

run_tests.sh

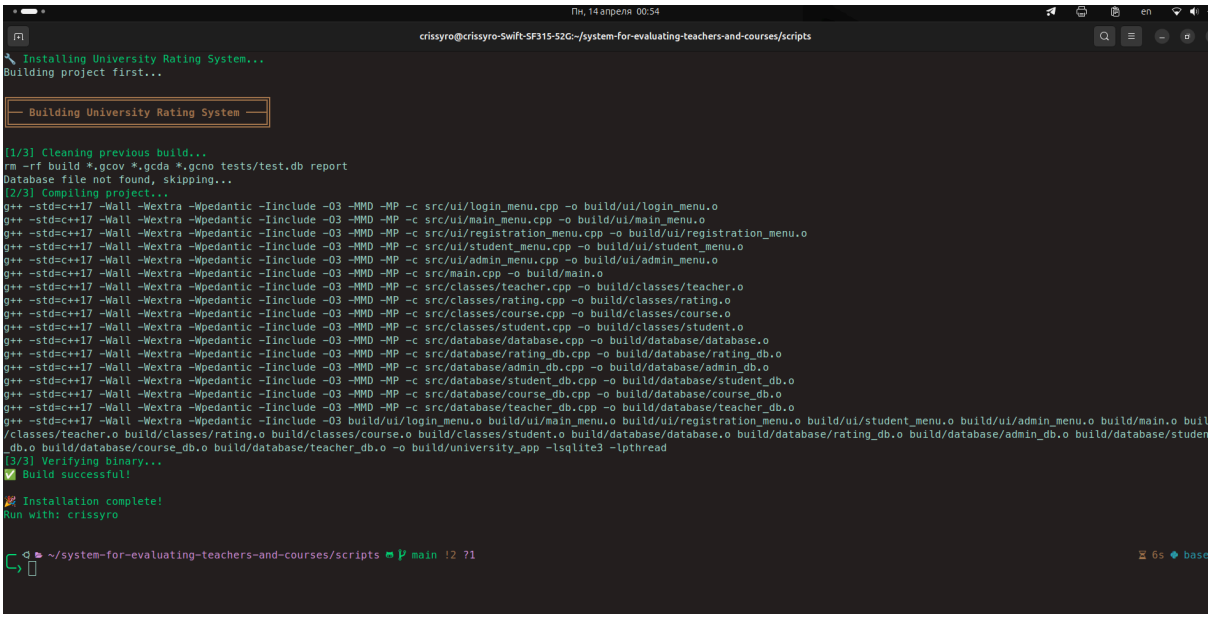
```
#!/usr/bin/env bash

RED='\033[0;31m'
GREEN='\033[0;32m'
BLUE='\033[0;34m'
NC='\033[0m'

run_tests() {
    cd ..
    echo -e "${BLUE}🚀 Running test suite...${NC}"
    if ! make test; then
        echo -e "${RED}❌ Tests failed!${NC}"
        exit 1
    fi

    echo -e "\n${GREEN}📊 Test results:${NC}"
```

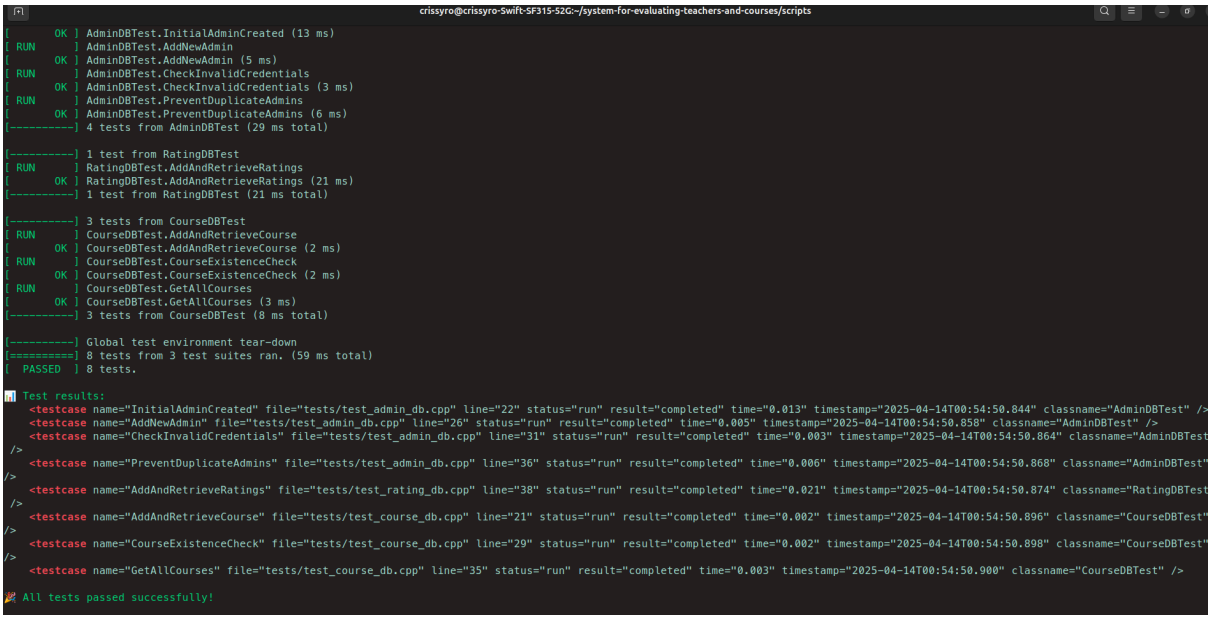
```
    cat build/test_results.xml | grep -E '<testcase|</failure>'
--color=always
}

main() {
    echo -e "${GREEN}✅ Checking test dependencies...${NC}"
    if ! command -v lcov &> /dev/null; then
        echo -e "${RED}Error: lcov required for coverage${NC}"
        exit 1
    fi

    run_tests
    echo -e "\n${GREEN}🎉 All tests passed successfully!${NC}"
}

main
```

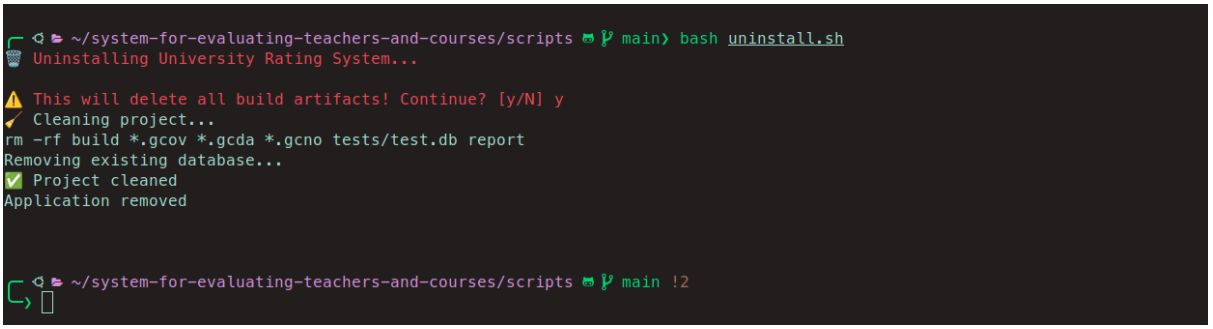
uninstall.sh

```
#!/usr/bin/env bash


RED='\033[0;31m'
NC='\033[0m'

uninstall_app() {
    cd ..
    echo -e "${RED}🗑️  Uninstalling University Rating System...${NC}"

    if [ -f "build/university_app" ]; then
        cd scripts
        bash clean.sh
        echo "Application removed"
    else
        echo "Application not found!"
    fi

    echo -e "\n${RED}👋 Uninstall complete!${NC}"
}
```

```
main() {
    uninstall_app
}

main
```







Вывод: Создал систему оценки преподавателей и учебных курсов, обеспечивающей сбор, анализ и хранение данных на основе мнений студентов.