

Министерство науки и высшего образования РФ
ФГАОУ ВО ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИУ)
Институт естественных и точных наук

Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования

« Разработка приложения для визуального управления проектами »

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ
по дисциплине «Объектно-ориентированное программирование»
ЮУрГУ–01.03.02.2024.036.ПЗ КР

Руководитель,

_____ *Демидов А.К.*

« ____ » _____ 2024 г.

Автор работы:

Студент группы: ЕТ – 212

_____ *Сычев А.Д.*

« ____ » _____ 2024 г.

Работа защищена с оценкой

« ____ » _____ 2024 г.

Челябинск – 2024

Федеральное государственное автономное образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Кафедра «Прикладная математика и программирование»
Направление _____

УТВЕРЖДАЮ
Заведующий кафедрой ПМиП

А.А.Замышляева

2024 г.

ЗАДАНИЕ
на курсовую работу студента
Сычев А.Д.
Группа ЕТ-212

1. Дисциплина Объектно-ориентированное программирование
2. Тема работы Разработка приложения для визуального управления проектами

3. Срок сдачи студентом законченной работы 25 декабря 2023 г.
4. Перечень вопросов, подлежащих разработке
1) разработка иерархии и интерфейса классов;
2) реализация программы (библиотеки классов) на языке C++
3) оформление программной документации (описание программы (библиотеки классов), руководство пользователя, листинг кода) и отчета по курсовой работе
4) презентация проектных решений для защиты КР (иерархия и интерфейсы классов, особенности реализации)
5. Календарный план

Наименование разделов (этапов) курсовой работы	Срок выполнения разделов (этапов) работы	Отметка о выполнении руководителя
анализ предметной области	01.09.2023-10.10.2023	
разработка иерархии и интерфейса классов	20.09.2023-07.11.2023	
реализация основных классов, функций	01.10.2023-20.11.2023	
тестирование программы и/или классов, улучшение и исправление ошибок	20.10.2023-10.12.2023	
оформление программной документации и отчета по курсовой работе	30.10.2023-20.12.2023	
защита курсовой работы	20.12.2023-28.12.2023	

Руководитель работы _____ / _____
Студент _____ / _____
(подпись) (расшифровка)

АННОТАЦИЯ

Сычев А.Д. Разработка приложения для визуального управления проектами. – Челябинск: ЮУрГУ, ЕТ-212, 2024. – 48с., библиографический список – 3 наим., 1 прил.

В курсовой работе описывается разработка многопользовательского приложения для визуального управления проектами используя объектно-ориентированный подход. Работа содержит результаты объектно-ориентированного анализа и проектирования, инструкции по установке и работе с приложением.

В результате работы было разработано приложение, которое позволяет командам управлять проектами, рабочими процессами и заданиям любых типов.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	5
1 ПОСТАНОВКА ЗАДАЧИ.....	6
2 ОПИСАНИЕ ПРОГРАММЫ.....	7
3 ИНСТРУКЦИЯ ПО УСТАНОВКЕ И ТРЕБОВАНИЯ К СИСТЕМЕ.....	12
4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	13
ЗАКЛЮЧЕНИЕ.....	16
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	17
ПРИЛОЖЕНИЕ	18

ВВЕДЕНИЕ

Актуальность темы. Объектно-ориентированный подход является наиболее прогрессивной технологией разработки программных систем, позволяет разрабатывать более сложные системы.

Цель работы – разработать приложение для визуального управления проектами.

Задачи работы:

- изучить приемы объектно-ориентированного анализа;
- научиться разрабатывать программы в объектно-ориентированном стиле;
- овладеть технологиями объектно-ориентированного анализа и проектирования;
- изучить концепции объектно-ориентированного программирования; изучить особенности объектной модели языка программирования С++;
- научиться самостоятельно и творчески использовать знания и полученные практические навыки;
- овладеть навыками самостоятельного получения новых знаний по теории и практике объектного подхода в программировании.

Объект работы – Программа для управления проектами.

Предмет работы – применение объектно-ориентированного подхода для разработки библиотеки.

Результаты работы можно использовать в процессе последующего обучения в соответствии с учебным планом подготовки бакалавров или для любых коммерческих целей.

1 ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать приложение для визуального управления проектами и задачами.

Приложение для визуального управления проектами - это программное приложение, разработанное для помощи пользователям в организации, управлении и отслеживании проектов и задач в удобном и наглядном формате. Такие приложения позволяют эффективно планировать, назначать и отслеживать задачи, а также управлять ресурсами и сроками проектов

Необходимые функции:

- Возможность авторизации для пользователей;
- Возможность просматривать и создавать свои проекты, присоединяться к проектам других людей;
- Возможность добавлять, удалять и изменять состояние поставленных задач в проектах;
- Возможность оставлять текстовые комментарии к задачам;

Анализ предметной области выявляет следующие объекты:

- Пользователь. У пользователя будут такие атрибуты, как имя, уникальный идентификатор, пароль и информация о проектах, в которых он принимает участие;
- Проект. Каждый проект будет иметь название и будет хранить в себе информацию о количестве участников, информацию о поставленных задачах, их количество и сроках проекта;
- Задача. Каждая задача будет иметь название и будет иметь дату создания, сроки задачи, комментарии к ней и её текущее состояние;

Требования к программе:

- Графический интерфейс пользователя;
- Возможность вести работу с мышью;
- База данных для многопользовательской работы;

2 ОПИСАНИЕ ПРОГРАММЫ

2.1 Для разработки программы были использованы:

- Компилятор Microsoft Visual C++ Compiler
- СУБД MariaDB
- MariaDBC++ Connector 1.02
- Интерфейсная библиотека nanogui

2.2 Программа состоит из 3 модулей:

Модуль Module (Интерфейсы в файле Module.hpp, реализации в файле Module.cpp). Этот модуль содержит в себе классы, представляющие основные сущности программы. Например проект, задача. Модуль содержит следующие классы:

```
struct Label {
    int id; // Уникальный номер для каждой метки
    std::string name; // Имя
    Label(int _id, std::string _name) : id(_id), name(_name) {}
// Конструктор
};

struct Comment {
    std::string author; // Имя автора комментария
    std::string text; // текст комментария
    int id; // Уникальный номер для каждого комментария
    Comment(std::string author, std::string text, int id) :
author(author), text(text), id(id){} // Конструктор
};

class BaseTemplate : public Label {
protected:
    std::string dateCreation; // Дата создания.
    std::string deadLine; // Дедлайн.
    std::vector<Comment> comments; // вектор объектов класса
Комментарий
    std::vector<std::string> users; // вектор пользователей.
Нужны только имена пользователей.
    int commentsAmount; // количество комментариев
    int usersAmount; // количество пользователей

public:
    friend class Database; // тк Database постоянно работает с
объектами его дочерних классов, объявляем его классом
дружественным

    BaseTemplate(std::string _name, std::string _dateCreation,
std::string _deadLine, int _id);

    // аксессор и мутатор для имени. Мы можем поменять имя
проекта
    std::string getName() const { return name; }
```

```

void setName(std::string _name) { name = _name; }

// аксессор для даты создания проекта. Менять его нельзя
std::string getDateCreation() const { return dateCreation;
}

// аксессор и мутатор для срока выполнения. Срок выполнения
можно сдвигать
std::string getDeadline() const { return deadLine; }
void setDeadLine(std::string date) { deadLine = date; }

// Методы для работы с сущностями комментариев.
int getCommentsAmount() { return commentsAmount; }
Comment getComment(int index);
void addComment(const Comment& comment) {
comments.push_back(comment); commentsAmount++; }
void deleteComment(int index);
// Методы для работы с пользователями.
void addUser(std::string user);
void deleteUser(std::string user);
int getUsersAmount() { return usersAmount; }
std::string getUser(int index);
};

class Task : public BaseTemplate {
private:
    bool isCompleted; // Поле обозначающее то, что задача
завершена.

public:
    Task(std::string _name, std::string _dateCreation,
std::string _deadline, bool _isCompleted, int _id);

    // Аксессор и мутатор для поля завершенности
    bool getCurrentState() const { return isCompleted; }
    void setCurrentState(bool flag) { isCompleted = flag; }
};

class Project : public BaseTemplate {
private:
    std::vector<Task> tasks; // Вектор задач.

    int tasksAmount; // Количество задач в проекте
    void copyFrom(const Project& other);

public:
    Project(std::string _name, std::string _dateCreation,
std::string _deadLine, int id);

```



```

        // Методы для работы с Задачами. Добавление, удаление,
        получение количества и получения по индексу.
        void addTask(const Task& task);
        void deleteTask(int index);
        int getTasksAmount() const { return tasksAmount; }
        Task& getTask(int index);

        Project& operator=(const Project& other);
};

```

Модуль Session (Интерфейсы в файле Session.hpp, реализации в файле Session.cpp). Этот модуль содержит в себе классы, представляющие возможности многопользовательской работы. Сессия пользователя базы данных и сессия конкретного пользователя программы. Модуль содержит следующие классы:

```

struct User {
    std::string name; // Имя пользователя
    int id; // Уникальный id пользователя, который нужен для
загрузок его проектов.
    User() : id(-1), name("User") {} // Конструктор по
умолчанию
    User(int id, std::string name) : id(id), name(name) {}
};

class Database {
private:
    sql::Driver* driver; // Драйвер базы данных, фактически
является отправной точкой для работы с ней
    sql::SQLString url; // адрес подключения к базе данных
    sql::Properties properties; // свойства подключения,
например данные пользователя
    std::unique_ptr<sql::Connection> conn; // "курсор" базы
данных. Фактически - та самая "мигающая палочка" в терминале,
откуда происходит ввод запросов

    // Этот метод вызывается внутри loadProject
    std::vector<Task> loadTasks(int id); // Загрузка задач из
базы данных по id проекта, к которому они принадлежат.

public:

    Database(const std::string&, const std::string&, const
std::string&, const std::string&);

    std::string addUserToProject(std::string, int);
    bool connect(); // Подключение пользователя. Возвращает
логические тип, что бы понимать произошло ли подключение.
Применяется в сессиях пользователя
    void disconnect(); // Отключение от базы данных

    std::vector<Label> loadProjectsDescription(const User&
currUser); // Первоначальная загрузка проектов что бы представить

```

пользователю доступные ему проекты.

```
void deleteProject(int id); // Удаление проекта из базы
данных на основе id проекта.
void saveProject(Project&); // Сохранение проекта в базе
данных. Если проект уже существует происходит перезапись проекта в
таблице. Если нет, то создание
Project loadProject(int id); // Загрузка проекта из базы
данных. Загрузка происходит по id проекта.

void saveTask(Task&, int); // Сохранение задачи в таблице.
Если задача есть, то произойдет перезапись, в ином случае
добавление новой записи.
void deleteTask(int id); // Удаление задачи из базы данных
на основе id задачи.

void saveComment(int id, const Comment& comment); //
Сохранение комментария в базу данных
std::vector<Comment> loadComments(int id); // Загрузка
комментариев из базы данных.
std::vector<std::string> loadUsers(int projectId); //
Загрузка ИМЕН пользователей из базы данных
User ConnectUser1(std::string name, std::string password);
// Создание сессии пользователя(программная сессия)

};
```

Модуль UserUI (Интерфейсы в файле UserUI.hpp, реализации в файле UserUI.cpp). Этот модуль содержит в себе классы, представляющие возможности создания графического интерфейса на экране. Модуль содержит следующие классы:

```
class LoginForm : public nanogui::Screen {
private:
    std::string enteredUsername;
    std::string enteredPassword;

public:
    LoginForm();
    void createLoginFormUI(); // Отрисовывает графический
интерфейс нашего окна
    std::string getEnteredUsername() const;
    std::string getEnteredPassword() const;
};

class ProjectSelectionWindow : public nanogui::Screen {
private:
    void createProjectSelectionWindowUI(); // Отрисовывает
графический интерфейс нашего окна
    void renderProjectsLayout(); // Отрисовывает задачи в нашем
окне. Метод вынесен отдельно по причине перерисовки во время
добавления новой задачи. так мы перерисовываем только одну часть
```

```

        Database& cursore; // Курсор базы данных. Нужен что бы
        работать с базой данных. Передается по ссылке начиная с начала
        main
        std::vector<Label> projectsLabel; // Содержит в себе
        названия и id проектов. После выбора подгружается конкретный
        проект из базы данных по этому id
        int selectedProjectId; // Выбранный id
        User& user; // Пользователь, который был авторизован в
        программу(не путать с пользователем базы данных)

    public:
        ProjectSelectionWindow(User& user, Database& cursore); //
        Стандартный конструктор
        void openAddProjectWindow(); // Открыть окно для добавления
        проекта
        int getSelectedProjectIndex() const; // Получить выбранный
        идентификатор после выбора
    };

    class FixedWindow : public nanogui::Window {
    public:
        FixedWindow(nanogui::Widget* parent, const std::string&
        title = "")
            : nanogui::Window(parent, title) {}

        bool mouseDragEvent(const Eigen::Vector2i&, const
        Eigen::Vector2i&, int button, int modifiers) override {
            return true;
        }
    };

    class ProjectWindow : public nanogui::Screen {
    private:
        Project& project; // Сам проект, по ссылке
        Database& cursore; // Курсор базы данных для работы с ней
        User& user; // конкретный пользователь

        FixedWindow* completedTasks; // подокно выполненных задач
        FixedWindow* uncompletedTasks; // подокно невыполненных
        задач

        std::string projectName; // имя проекта
        std::string dateOfCreation; // дата создания проекта
        std::string deadLine; // сроки сдачи проекта
        void renderTaskWindows(); // отрисовка задач в окнах.
        Сделана отдельно что бы удобнее было перерисовывать окно
        void renderComments(Task& task, const FixedWindow& window);
        // Отрисовка комментариев в задаче. Сделана отдельно что бы было
        удобнее перерисовывать.

```

```

public:
    ProjectWindow(Project& _project, Database&, User& user); //
стандартный конструктор
    void createUI(); // отрисовка основного графического
интерфейса окна
    void openInfoAboutProject(); // отрисовка окна, которое
открывается при нажатии кнопки открыть информацию
    void openAddUserWindow(); // отрисовка окна, которое
открывается при добавлении пользователя
    void openTaskInfoWindow(Task& task); // отрисовка окна
которое открывается при выборе задачи
    void openAddTaskWindow(); // отрисовка окна, которое
открывается при добавлении задач
};

```

2.3 Пояснения по программе и особенностям реализации

Ключевая идея состоит в том, что любой проект может быть разбит на более маленькие подзадачи, которые могут быть распределены среди нескольких разработчиков проекта. Таким образом происходит грамотное использование трудовых ресурсов и времени.

Во время запуска необходима авторизация пользователя в программе. Данные для авторизации пользователю предоставляет серверный администратор.

В случае успешной авторизации пользователю становятся видны все его проекты. Пользователь может выбрать для работы один из доступных ему проектов либо создать новый, указав его название и срок выполнения.

После выбора проекта, пользователь попадает в главное окно. В нем показаны все доступные задачи, информация о проекте и возможность нового пользователя и новую задачу в проект..

В случае выбора конкретной задачи, показываются ее сроки, ее название, ее конкретный статус выполнения и комментарии к ней. По желанию, некоторые значения можно изменять.

В случае добавления нового пользователя в проект необходимо указать его имя. Имена пользователей это уникальная сущность, по этой причине ошибок быть не должно.

2.4 Используемые внешние файлы

Приложение имеет доступ к базе данных на сервере.

3 ИНСТРУКЦИЯ ПО УСТАНОВКЕ И ТРЕБОВАНИЯ К СИСТЕМЕ

Для сборки приложения понадобится. Microsoft Visual Studio, либо компилятор MSVC. База данных основанная на СУБД MariaDB, либо MySQL. Библиотека для создания сессий к базе данных MariaDBCPP Connector, библиотека для графического интерфейса nanogui и все библиотеки, которые требуются для работы с OpenGL. В случае установки приложения, понадобится возможность работать с OpenGL, а также постоянное подключение к интернету из-за базы данных.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При запуске программы откроется окно авторизации (см. Рисунок 1).

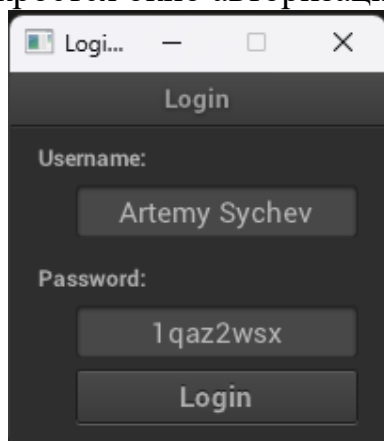


Рисунок 1 - окно авторизации

После авторизации откроется окно с выбором проекта (см. Рисунок 2).

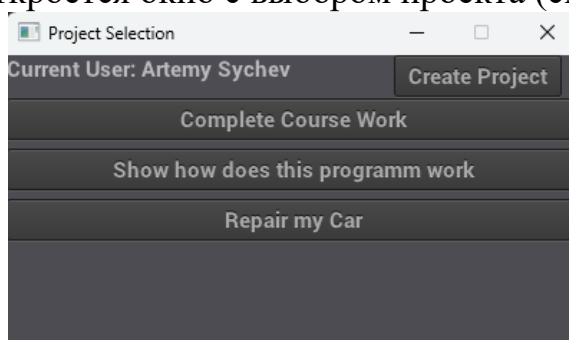


Рисунок 2 - окно выбора проекта

В левом верхнем углу написано имя текущего пользователя. В правом верхнем углу кнопка для создания нового проекта. При нажатии этой кнопки откроется окно, в котором вы сможете ввести названия и сроки для нового проекта (см. Рисунок 3)

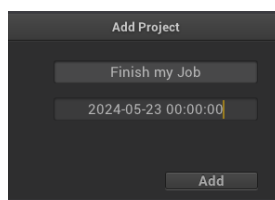


Рисунок 3 - Добавление проекта

После выбора проекта, откроется главное окно, в котором будут два поля, содержащие в себе невыполненные и выполненные задачи, кнопка с информацией о проекте, кнопка для добавления нового пользователя в проект, кнопка для добавления новой задачи в проект и кнопка выхода.(см. Рисунок 4)

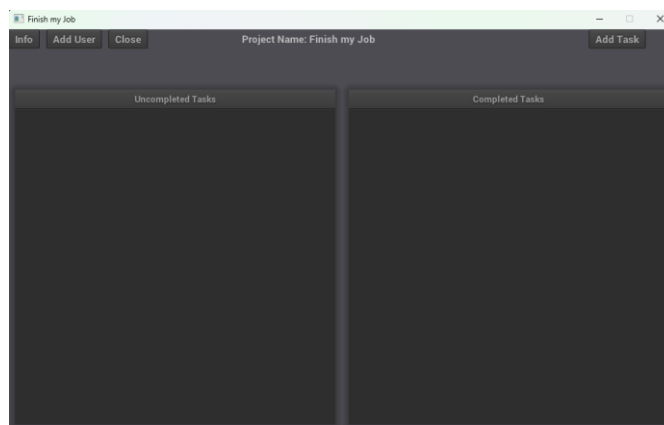


Рисунок 4 - окно выбранного проекта

При нажатии на кнопку «Info» откроется окно, которое содержит информацию об имени проекта, его сроках и пользователях. (см. Рисунок 5)

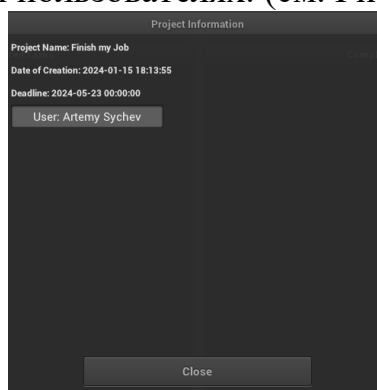


Рисунок 5 - информация о проекте

При нажатии на кнопку «Add User» откроется окно, в котором надо будет ввести имя пользователя которого надо добавить. (см. Рисунок 6)

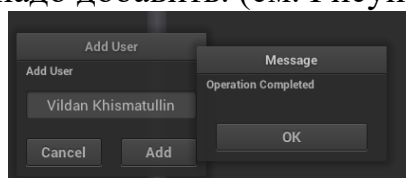


Рисунок 6 - Добавление пользователя

При нажатии на кнопку «Add Task» откроется окно, в котором надо будет ввести название для задачи и ее сроки выполнения. (см. Рисунок 7)

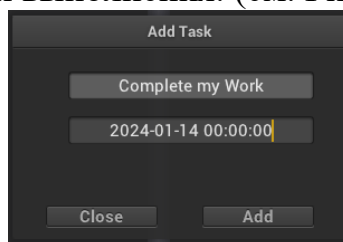


Рисунок 7 - Добавление задачи

При выборе конкретной задачи открывается окно с информацией о ней. Ее названии, сроках, статусе и комментариях. Данные можно менять, в таком случае надо нажать на кнопку «Save Task». Помимо этого задачу можно удалить, для этого надо нажать «Delete Task». (см. Рисунок 9)

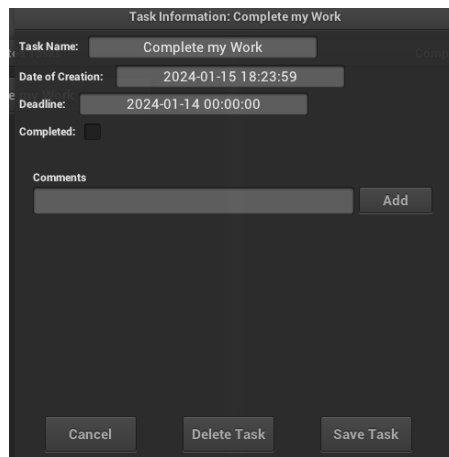


Рисунок 9 - Выбор конкретной задачи

Для добавления комментария, надо ввести его в специально выделенную форму и нажать кнопку «Add». Чтобы комментарий сохранился необходимо так же нажать «Save Task».

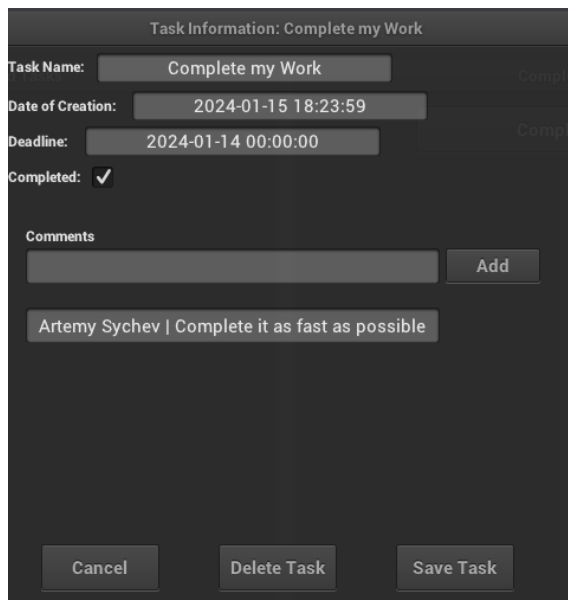


Рисунок 10 - Добавление комментария.

Для выхода необходимо нажать «Close».

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были выявлены объекты предметной области и определена система классов для них. После объектно-ориентированного проектирования классы были реализованы на языке C++. Разработанный код был проверен на контрольных теста. Для приложения была разработана документация, описывающая её установку и использование. Таким образом, цель работы была достигнута, задачи – решены.

Результаты работы можно использовать в процессе последующего обучения в форме навыков практического применения объектно-ориентированного подхода для разработки сложных программных систем, понимания порядка этапов разработки программного обеспечения и достигаемых на каждом этапе результатов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования. [Электронный ресурс] / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. — Электрон. дан. — М. : ДМК Пресс, 2007. — 368 с. — Режим доступа: <http://e.lanbook.com/book/1220>
2. Липман, С. Язык программирования C++. Полное руководство. [Электронный ресурс] / С. Липман, Ж. Лажоие. — Электрон. дан. — М. : ДМК Пресс, 2006. — 1105 с. — Режим доступа: <http://e.lanbook.com/book/1216>
3. Алгоритм построения суффиксного автомата за линейное время. – Дата обновления: 20.05.2012. URL: http://e-maxx.ru/algo/suffix_automata#8 (дата обращения: 22.02.2016).

ПРИЛОЖЕНИЕ А

А.1 Файл Module.hpp

```
#ifndef MODULE_H
#define MODULE_H

#include <iostream>
#include <vector>
#include <string>
#include <chrono>

class Task;

//class LABEL
/* Этот класс используется для того, что бы во время выбора
проекта в памяти не хранились все объекты из базы данных.
Благодаря чему, мы экономим память. А для выбора нам хватит
названия. id же нужен, что бы потом загрузить объект класса
Project из базы данных. Куда мы передаем id и загружаем проект по
его project_id из таблицы. */
struct Label {
    int id; // Уникальный номер для каждой метки
    std::string name; // Имя
    Label(int _id, std::string _name) : id(_id), name(_name) {} //
Конструктор
};

//class COMMENT
/*Этот класс нужен, что бы хранить информацию о комментарии. У
любого комментария есть автор, текст и дата создания. Пришла идея
реализовать через класс, так как так легче и лучше всего собирать
разные сущности в одно. Раньше вы сказали, что нельзя делать
комментарий только текстом.*/

struct Comment {
    std::string author; // Имя автора комментария
    std::string text; // текст комментария
    int id; // Уникальный номер для каждого комментария
    Comment(std::string author, std::string text, int id) :
author(author), text(text), id(id){} // Конструктор
};

// class BASETEMPLATE
/*Этот класс нужен для описания стандартных полей для Проекта и
Задачи*/

class BaseTemplate : public Label {
protected:
    std::string dateCreation; // Дата создания.
    std::string deadLine; // Дедлайн.
```

```

        std::vector<Comment> comments; // вектор объектов класса
Комментарий
        std::vector<std::string> users; // вектор пользователей. Нужны
только имена пользователей.
        int commentsAmount; // количество комментариев
        int usersAmount; // количество пользователей

public:
        friend class Database; // тк Database постоянно работает с
объектами его дочерних классов, объявляем его классом
дружественным

        BaseTemplate(std::string _name, std::string _dateCreation,
std::string _deadLine, int _id);

        // аксессор и мутатор для имени. Мы можем поменять имя проекта
std::string getName() const { return name; }
void setName(std::string _name) { name = _name; }

        // аксессор для даты создания проекта. Менять его нельзя
std::string getDateCreation() const { return dateCreation; }

        // аксессор и мутатор для срока выполнения. Срок выполнения
можно сдвигать
std::string getDeadline() const { return deadLine; }
void setDeadLine(std::string date) { deadLine = date; }

        // Методы для работы с сущностями комментариев.
int getCommentsAmount() { return commentsAmount; }
Comment getComment(int index);
void addComment(const Comment& comment) {
comments.push_back(comment); commentsAmount++; }
void deleteComment(int index);

        // Методы для работы с пользователями.
void addUser(std::string user);
void deleteUser(std::string user);
int getUsersAmount() { return usersAmount; }
std::string getUser(int index);
};

// class TASK
/*Этот класс нужен для представление задач. Как правило - задачу
сущность атомарная.*/

class Task : public BaseTemplate {
private:
        bool isCompleted; // Поле обозначающее то, что задача
завершена.

```

```

public:
    Task(std::string _name, std::string _dateCreation, std::string
        _deadline, bool _isCompleted, int _id);

    // Аксессор и мутатор для поля завершенности
    bool getCurrentState() const { return isCompleted; }
    void setCurrentState(bool flag) { isCompleted = flag; }
};

// class PROJECT
/*Этот класс нужен для объединения задач по общему признаку
выполнения. Например разбиение алгоритма наливания чая на задачи.
Вскипятить воду, и так далее.*/

class Project : public BaseTemplate {
private:
    std::vector<Task> tasks; // Вектор задач. Нужен что бы мы
    могли работать с задачами относящимися к одному проекту
    int tasksAmount; // Количество задач в проекте
    void copyFrom(const Project& other);

public:

    Project(std::string _name, std::string _dateCreation,
        std::string _deadLine, int id);

    // Методы для работы с Задачами. Добавление, удаление,
    получение количества и получения по индексу.
    void addTask(const Task& task);
    void deleteTask(int index);
    int getTasksAmount() const { return tasksAmount; }
    Task& getTask(int index);

    Project& operator=(const Project& other);
};

#endif

```

A.2 Файл Module.cpp

```

#include "Module.hpp"

// BASETEMPLATE IMPLEMENTATION
// Конструктор объектов Базового шаблона
BaseTemplate::BaseTemplate(std::string _name, std::string
    _dateCreation, std::string _deadLine, int _id)
    : Label(_id, _name), dateCreation(_dateCreation),
    deadLine(_deadLine), comments(), users() {
}

Comment BaseTemplate::getComment(int index) {

```

```

        if (index >= 0 && index < comments.size()) {
            return comments[index];
        }
        else {
            std::cout << "Index out of range\n";
        }
    }

// Удалить комментарий из базового шаблона (наследуется задаче и проекту)
void BaseTemplate::deleteComment(int index) {
    if (index >= 0 && index < comments.size()) {
        comments.erase(comments.begin() + index);
    }
    else {
        std::cout << "This element doesn't exist\n";
    }
}

// Добавить пользователя в объект базового шаблона (наследуется задаче и проекту)
void BaseTemplate::addUser(std::string user) {
    if (std::find(users.begin(), users.end(), user) ==
users.end()) {
        users.push_back(user);
        usersAmount++;
    }
    else {
        std::cout << "User already exists";
    }
}

std::string BaseTemplate::getUser(int index) {
    if (index >= 0 && index < usersAmount) {
        return users[index];
    }
    else {
        std::cout << "User already exists";
    }
}

// Удалить пользователя из объекта базового шаблона (наследуется задаче и проекту)
void BaseTemplate::deleteUser(std::string user) {
    auto it = std::find(users.begin(), users.end(), user);
    if (it != users.end()) {
        users.erase(it);
        usersAmount--;
    }
    else {
        std::cout << "User doesn't exist";
    }
}

```

```

    }

}

// PROJECT IMPLEMENTATION
// Конструктор проекта.
Project::Project(std::string _name, std::string _dateCreation,
std::string _deadLine, int _id)
    : BaseTemplate(_name, _dateCreation, _deadLine, _id) {
    tasks = std::vector<Task>();
}

// Добавляет задачу в проект.
void Project::addTask(const Task& task) {
    tasks.push_back(task);
    tasksAmount++;
}

// Удаление задачи из проекта, по индексу.
void Project::deleteTask(int index) {
    for (int i = 0; i < tasksAmount; i++) {
        if (tasks[i].id == index) {
            tasks.erase(tasks.begin() + i);
            tasksAmount = tasks.size();
            return;
        }
    }
    std::cout << "Task with id " << index << " not found." <<
std::endl;
}

// Получение задачи по ее индексу.
Task& Project::getTask(int index) {
    if (index >= 0 && index < tasksAmount) {
        return tasks[index];
    }
}

// Копирование проекта.
void Project::copyFrom(const Project& other) {
    name = other.name;
    dateCreation = other.dateCreation;
    deadLine = other.deadLine;
    users = other.users;
    usersAmount = other.usersAmount;

    tasks.clear();
    for (const Task& task : other.tasks) {
        tasks.push_back(task);
    }
}

```

```

        tasksAmount = other.tasksAmount;
    }

Project& Project::operator=(const Project& other) {
    if (this != &other) {
        copyFrom(other);
    }
    return *this;
}

// TASK IMPLEMENTATION

// Конструктор задач
Task::Task(std::string _name,      std::string _dateCreation,
std::string _deadLine, bool _isCompleted, int _id)
    :   BaseTemplate(_name,      _dateCreation,      _deadLine,      _id),
isCompleted(_isCompleted) {
}

```

A.3 Файл Session.hpp

```

#ifndef SESSION_H
#define SESSION_H

#include <conncpp.hpp>
#include <iostream>
#include <vector>
#include <string>
#include "Module.hpp"

// class USER
// Класс User нужен для того, что бы хранить информацию о сессии.
// Фактически для загрузки проектов по id пользователя, который ее
// вызывает

struct User {
    std::string name; // Имя пользователя
    int id; // Уникальный id пользователя, который нужен для
загрузок его проектов.
    User() : id(-1), name("User") {} // Конструктор по умолчанию
    User(int id, std::string name) : id(id), name(name) {}
};

// class DATABASE
/*Этот класс нужен для работы с базой данных. Является
дружественным к BaseTemplate, тк постоянно работает с его
полями.*/

```

```

class Database {
private:
    sql::Driver* driver; // Драйвер базы данных, фактически
    является отправной точкой для работы с ней
    sql::SQLString url; // адрес подключения к базе данных
    sql::Properties properties; // свойства подключения, например
    данные пользователя
    std::unique_ptr<sql::Connection> conn; // "курсор" базы
    данных. Фактически - та самая "мигающая палочка" в терминале,
    откуда происходит ввод запросов

    // Этот метод вызывается внутри loadProject
    std::vector<Task> loadTasks(int id); // Загрузка задач из базы
    данных по id проекта, к которому они принадлежат. Например я
    вызываю загрузка задач из проекта с id 15.

public:

    Database(const std::string&, const std::string&, const
    std::string&, const std::string&);

    std::string addUserToProject(std::string, int);

    bool connect(); // Подключение пользователя. Возвращает
    логический тип, что бы понимать произошло ли подключение.
    Применяется в сессиях пользователя
    void disconnect(); // Отключение от базы данных

    std::vector<Label> loadProjectsDescription(const User&
    currUser); // Первоначальная загрузка проектов что бы представить
    пользователю доступные ему проекты.

    void deleteProject(int id); // Удаление проекта из базы данных
    на основе id проекта.
    void saveProject(Project&); // Сохранение проекта в базе
    данных. Если проект уже существует происходит перезапись проекта в
    таблице. Если нет, то создание
    Project loadProject(int id); // Загрузка проекта из базы
    данных. Загрузка происходит по id проекта.

    void saveTask(Task&, int); // Сохранение задачи в таблице.
    Если задача есть, то произойдет перезапись, в ином случае
    добавление новой записи.
    void deleteTask(int id); // Удаление задачи из базы данных на
    основе id задачи.

    void saveComment(int id, const Comment& comment); //
    Сохранение комментария в базу данных
    std::vector<Comment> loadComments(int id); // Загрузка
    комментариев из базы данных.
    std::vector<std::string> loadUsers(int projectId); // Загрузка
    ИМЕН пользователей из базы данных

```



```

        User ConnectUser1(std::string name, std::string password); //
        Создание сессии пользователя(программная сессия)

};
#endif

```

A.4 Файл Module.cpp

```

#include "Session.hpp"

// Конструктор. Во время создания объекта создается подключение.
// Иначе выбрасывается исключение.
Database::Database(const std::string& host, const std::string&
user, const std::string& password, const std::string& database)
:
    driver(sql::mariadb::get_driver_instance()),
    url("jdbc:mariadb://" + host + "/" + database),
    properties({ {"user", user}, {"password", password} }),
    conn(driver->connect(url, properties))
{
    if (!conn) {
        throw std::runtime_error("Failed to connect to the
database");
    }
}

// Отключение от базы данных. нужно во время выхода.
void Database::disconnect() {

    if (conn) {
        conn->close();
        conn.reset();
    }
}

// Загрузка описания таблицы. Нужна, что бы не подгружать все
// проекты с их полями из базы данных. нужно для экономии памяти
std::vector<Label> Database::loadProjectsDescription(const User&
currentUser) {
    std::vector<Label> projectDescriptions;

    try {
        int userId = currentUser.id;

        sql::PreparedStatement* stmt = conn->prepareStatement(
            "SELECT    projects.project_id,    projects.name    FROM
projects "
            "JOIN user_project ON user_project.user_id = ? "
            "WHERE projects.project_id = user_project.project_id"
        );
    }
}

```

```

        stmt->setInt(1, userId);

        sql::ResultSet* res = stmt->executeQuery();

        while (res->next()) {
            int projectId = res->getInt("project_id");
            std::string name = res->getString("name").c_str();

            projectDescriptions.emplace_back(projectId, name);
        }

        delete stmt;
    }
    catch (sql::SQLException& e) {
        std::cerr << "SQL Exception: " << e.what() << std::endl;
    }

    return projectDescriptions;
}

// Сохранение проекта в таблице. При его сохранении автоматически
сохраняются все его задачи
void Database::saveProject(Project& project) {
    try {
        sql::PreparedStatement* pstmt;

        if (project.id == -1) {
            pstmt = conn->prepareStatement("INSERT IGNORE INTO
projects (name, date_creation, deadline) VALUES (?, ?, ?)");
            pstmt->setString(1, project.name);
            pstmt->setString(2, project.dateCreation);
            pstmt->setString(3, project.deadLine);
            pstmt->executeUpdate();
            delete pstmt;

            int lastInsertId = -1;
            sql::Statement* stmt = conn->createStatement();
            sql::ResultSet* res = stmt->executeQuery("SELECT
LAST_INSERT_ID()");
            if (res->next()) {
                lastInsertId = res->getInt(1);
                addUserToProject("Artemy Sychev", lastInsertId);
            }
            delete stmt;
        }
        else {
            pstmt = conn->prepareStatement("UPDATE projects SET
name=?, date_creation=?, deadline=? WHERE project_id=?");
            pstmt->setString(1, project.name);
            pstmt->setString(2, project.dateCreation);
            pstmt->setString(3, project.deadLine);

```

```

        pstmt->setInt(4, project.id);
        pstmt->executeUpdate();
    }

    delete pstmt;

    for (int i = 0; i < project.getTasksAmount(); i++) {
        saveTask(project.getTask(i), project.id);
    }
}
catch (sql::SQLException& e) {
    std::cerr << "SQL Exception: " << e.what() << std::endl;
}
}

// Загрузка проекта из таблицы по ее id.
Project Database::loadProject(int id) {
    try {
        sql::Statement* stmt = conn->createStatement();
        std::string query = "SELECT * FROM projects WHERE
project_id = " + std::to_string(id);

        sql::ResultSet* res = stmt->executeQuery(query);

        res->next();
        int id = res->getInt(1);
        std::string name = res->getString(2).c_str();
        std::string dateCreation = res->getString(4).c_str();
        std::string deadLine = res->getString(3).c_str();
        Project tempProject = Project(name, dateCreation,
deadLine, id);
        std::vector<std::string> tempUsers =
loadUsers(tempProject.id);

        for (std::string user : tempUsers)
            tempProject.addUser(user);

        std::vector<Task> tasks = loadTasks(id);
        for (int i = 0; i < tasks.size(); i++) {
            tempProject.addTask(tasks[i]);
        }
        delete stmt;

        return tempProject;
    }
    catch (sql::SQLException& e) {
        std::cerr << "SQL Exception: " << e.what() << std::endl;
    }
}
}

```

// Удаление проекта из таблицы. Так как в таблице настроен auto cascade, при удалении проекта автоматически удалятся и задачи которые относятся к нему. так же таблице связей с пользователями удалит лишнии записи.

```
void Database::deleteProject(int taskId) {
    try {
        sql::PreparedStatement* pstmt = conn-
>prepareStatement("DELETE FROM projects WHERE project_id = ?");
        pstmt->setInt(1, taskId);
        pstmt->execute();
        delete pstmt;
    }
    catch (sql::SQLException& e) {
        std::cerr << "SQL Exception: " << e.what() << std::endl;
    }
}
```

// сохранение задачи в таблице. Если id == -1, то задача была создана в программе и ее записываем. Если id != -1, то в таком случае обновляем запись

```
void Database::saveTask(Task& task, int project_id) {
    try {
        sql::PreparedStatement* pstmt;

        if (task.id == -1) {
            pstmt = conn->prepareStatement("INSERT INTO tasks
(name, date_creation, is_completed, deadline, project_id) VALUES
(?, ?, ?, ?, ?)");
        }
        else {
            pstmt = conn->prepareStatement("UPDATE tasks SET
name=?, date_creation=?, is_completed=?, deadline=?, project_id=?
WHERE task_id=?");
            pstmt->setInt(6, task.id);
        }

        pstmt->setString(1, task.name);
        pstmt->setString(2, task.dateCreation);
        pstmt->setString(3,
std::to_string(task.getCurrentState()));
        pstmt->setString(4, task.deadLine);
        pstmt->setString(5, std::to_string(project_id));

        pstmt->executeUpdate();

        for (int i = 0; i < task.getCommentsAmount(); i++) {
            saveComment(task.id, task.getComment(i));
        }
    }
}
```

```

    }

    delete pstmt;

}
catch (sql::SQLException& e) {
    std::cerr << "SQL Exception: " << e.what() << std::endl;
}
}

// Загрузка задач из таблицы. Происходит по id проекта, к которому
задачи относятся.
std::vector<Task> Database::loadTasks(int id) {
    std::vector<Task> tasks;

    try {
        sql::PreparedStatement* pstmt = conn-
>prepareStatement("SELECT * FROM tasks WHERE project_id = ?");
        pstmt->setInt(1, id);

        sql::ResultSet* res = pstmt->executeQuery();

        while (res->next()) {
            int task_id = res->getInt("task_id");
            std::string name = res->getString("name").c_str();
            std::string dateCreation = res-
>getString("date_creation").c_str();
            bool isCompleted = res->getBoolean("is_completed");
            std::string deadLine = res-
>getString("deadline").c_str();
            Task tempTask = Task(name, dateCreation, deadLine,
isCompleted, task_id);

            std::vector<Comment> tempComments =
loadComments(tempTask.id);

            for (Comment& comment : tempComments) {
                tempTask.addComment(comment);
            }

            tasks.push_back(tempTask);
        }

        delete pstmt;
    } catch (sql::SQLException& e) {
        std::cerr << "SQL Exception: " << e.what() << std::endl;
    }

    return tasks;
}

```

```

// Удаление задачи из таблицы. Так как в таблице настроен auto
cascade, при удалении задачи автоматически удалятся и комментарии,
которые относятся к ней.
void Database::deleteTask(int taskId) {
    try {
        sql::PreparedStatement* stmt = conn-
>prepareStatement("DELETE FROM tasks WHERE task_id = ?");
        stmt->setInt(1, taskId);
        stmt->execute();
        delete stmt;
    }
    catch (sql::SQLException& e) {
        std::cerr << "SQL Exception: " << e.what() << std::endl;
    }
}

```

// Загрузка комментариев из таблицы. Так как сущность комментария принадлежит задаче, поэтому связываем ее с таблицей задачи с помощью Primary Key.

```

std::vector<Comment> Database::loadComments(int id) {
    std::vector<Comment> comments;

    try {
        sql::PreparedStatement* pstmt = conn-
>prepareStatement("SELECT name, text, comment_id FROM
task_comments WHERE task_id = ?");
        pstmt->setInt(1, id);

        sql::ResultSet* res = pstmt->executeQuery();

        while (res->next()) {
            std::string author = res->getString("name").c_str();
            std::string text = res->getString("text").c_str();
            int id = res->getInt("comment_id");
            comments.push_back(Comment(author, text, id));
        }

        delete pstmt;
    }
    catch (sql::SQLException& e) {
        std::cerr << "SQL Exception: " << e.what() << std::endl;
    }

    return comments;
}

```

```

void Database::saveComment(int id, const Comment& comment) {
    try {

```

```

        if (comment.id == -1) {
            sql::PreparedStatement* pstmt = conn-
>prepareStatement("INSERT INTO task_comments (task_id, name, text)
VALUES (?, ?, ?)");
            pstmt->setInt(1, id);
            pstmt->setString(2, comment.author);
            pstmt->setString(3, comment.text);
            pstmt->executeUpdate();
            delete pstmt;
        }
    }
    catch (sql::SQLException& e) {
        std::cerr << "SQL Exception: " << e.what() << std::endl;
    }
}

```

```

// Этот метод нужен для подключения пользователя. В случае того,
// если пользователь с такими данными отсутствует, выбрасывается
// исключение. Иначе создается объект с id пользователя
User Database::ConnectUser1(std::string name, std::string
password) {
    try {
        sql::PreparedStatement* pstmt = conn-
>prepareStatement("SELECT * FROM users WHERE name = ? AND password
= ?");
        pstmt->setString(1, name);
        pstmt->setString(2, password);

        sql::ResultSet* res = pstmt->executeQuery();

        User currentUser;

        if (res->next()) {
            int user_id = res->getInt(1);
            std::string name = res->getString(2).c_str();
            currentUser = User(user_id, name);
        }
        else {
            std::cout << "User not found." << std::endl;
        }

        delete pstmt;

        return currentUser;
    }
    catch (sql::SQLException& e) {
        std::cerr << "SQL Exception: " << e.what() << std::endl;
    }
}

```

```

std::string Database::addUserToProject(std::string name, int
project_id) {
    try {
        sql::PreparedStatement* pstmt = conn-
>prepareStatement("SELECT * FROM users WHERE name = ?");
        pstmt->setString(1, name);

        sql::ResultSet* res = pstmt->executeQuery();

        int user_id = -1;

        if (res->next()) {
            user_id = res->getInt(1);
        }
        else {
            return "User not found.";
        }

        pstmt = conn->prepareStatement("SELECT * FROM user_project
WHERE user_id = ? AND project_id = ?");
        pstmt->setString(1, std::to_string(user_id));
        pstmt->setString(2, std::to_string(project_id));

        res = pstmt->executeQuery();

        if (!(res->next())) {
            pstmt = conn->prepareStatement("INSERT INTO
user_project (user_id, project_id) VALUES (?, ?)");
            pstmt->setString(1, std::to_string(user_id));
            pstmt->setString(2, std::to_string(project_id));
            pstmt->executeUpdate();
        }
        return "Operation Completed";
        delete pstmt;
    }
    catch (sql::SQLException& e) {
        std::cerr << "SQL Exception: " << e.what() << std::endl;
    }
}

```

```

std::vector<std::string> Database::loadUsers(int projectId) {
    std::vector<std::string> projectUsers;

    try {
        sql::PreparedStatement* pstmt = conn->prepareStatement(
            "SELECT u.name FROM users u "
            "JOIN user_project up ON u.id = up.user_id "
            "WHERE up.project_id = ?"
        );
    }
}

```



```

        pstmt->setInt(1, projectId);

        sql::ResultSet* res = pstmt->executeQuery();

        while (res->next()) {
            std::string userName = res->getString("name").c_str();
            projectUsers.push_back(userName);
        }

        delete pstmt;
    }
    catch (sql::SQLException& e) {
        std::cerr << "SQL Exception: " << e.what() << std::endl;
    }

    return projectUsers;
}

```

A.5 Файл UserUI.hpp

```

#ifndef UI_HPP
#define UI_HPP

#include <nanogui/nanogui.h>
#include "Module.hpp"
#include "Session.hpp"

// Класс, который нужен для работы с окном авторизации
пользователя.
class LoginForm : public nanogui::Screen {
private:
    std::string enteredUsername;
    std::string enteredPassword;

public:
    LoginForm();
    void createLoginFormUI(); // Отрисовывает графический
интерфейс нашего окна
    std::string getEnteredUsername() const;
    std::string getEnteredPassword() const;
};

// Класс, который нужен для работы с окном выбора доступных ему
проектов у конкретного пользователя (или создание нового).
Удаление происходит только в таблице базы данных привелегированным
пользователем
class ProjectSelectionWindow : public nanogui::Screen {
private:
    void createProjectSelectionWindowUI(); // Отрисовывает
графический интерфейс нашего окна
    void renderProjectsLayout(); // Отрисовывает задачи в нашем
окне. Метод вынесен отдельно по причине перерисовки во время

```

```

добавления новой задачи. так мы перерисовываем только одну часть
    Database& cursore; // Курсор базы данных. Нужен что бы
работать с базой данных. Передается по ссылке начиная с начала
main
    std::vector<Label> projectsLabel; // Содержит в себе названия
и id проектов. После выбора подгружается конкретный проект из базы
данных по этому id
    int selectedProjectId; // Выбранный id
    User& user; // Пользователь, который был авторизован в
программу(не путать с пользователем базы данных)

public:
    ProjectSelectionWindow(User& user, Database& cursore); //
Стандартный конструктор
    void openAddProjectWindow(); // Открыть окно для добавления
проекта
    int getSelectedProjectIndex() const; // Получить выбранный
идентификатор после выбора
};

// Стандартный класс Window имеет виртуальный метод
mouseDragEvent. Поэтому я отнаследовал свой класс, в котором я
запретил двигать это окно.
class FixedWindow : public nanogui::Window {
public:
    FixedWindow(nanogui::Widget* parent, const std::string& title
= "")
        : nanogui::Window(parent, title) {}

    bool mouseDragEvent(const Eigen::Vector2i&, const
Eigen::Vector2i&, int button, int modifiers) override {
        return true;
    }
};

// Окно в котором представлен конкретный проект. Открывается после
выбора проекта
class ProjectWindow : public nanogui::Screen {
private:
    Project& project; // Сам проект, по ссылке
    Database& cursore; // Курсор базы данных для работы с ней
    User& user; // конкретный пользователь

    FixedWindow* completedTasks; // подокно выполненных задач
    FixedWindow* uncompletedTasks; // подокно невыполненных задач

    std::string projectName; // имя проекта
    std::string dateOfCreation; // дата создания проекта
    std::string deadLine; // сроки сдачи проекта
    void renderTaskWindows(); // отрисовка задач в окнах. Сделана

```

отдельно что бы удобнее было перерисовывать окно
void renderComments(Task& task, const FixedWindow& window); // Отрисовка комментариев в задаче. Сделана отдельно что бы было удобнее перерисовывать.

```
public:
    ProjectWindow(Project& _project, Database&, User& user); // стандартный конструктор
    void createUI(); // отрисовка основного графического интерфейса окна
    void openInfoAboutProject(); // отрисовка окна, которое открывается при нажатии кнопки открыть информацию
    void openAddUserWindow(); // отрисовка окна, которое открывается при добавлении пользователя
    void openTaskInfoWindow(Task& task); // отрисовка окна которое открывается при выборе задачи
    void openAddTaskWindow(); // отрисовка окна, которое открывается при добавлении задач
};
```

#endif

A.6 Файл UserUI.cpp

```
#include "UserUI.hpp"
#include <chrono>
#include <ctime>
#include <iomanip>
#include <sstream>
```

// Функция для получения текущего времени. Нужна что бы получать время создания проектов или задач для записи в базу данных

```
std::string getCurrentDateTime() {
    auto now = std::chrono::system_clock::now();
    std::time_t currentTime =
std::chrono::system_clock::to_time_t(now);
    std::tm localTime;

    localtime_s(&localTime, &currentTime);

    std::ostringstream oss;
    oss << std::put_time(&localTime, "%y-%m-%d %H:%M:%S");
    return oss.str();
}
```

```

// Конструктор создающий само окно.
LoginForm::LoginForm() : nanogui::Screen(Eigen::Vector2i(200,
200), "Login Window", false) {
    createLoginFormUI();
}
// Отрисовка самого окна
void LoginForm::createLoginFormUI() {

    nanogui::Window* window = new nanogui::Window(this, "Login");
    window->setLayout(new nanogui::GroupLayout());

    nanogui::Label* usernameLabel = new nanogui::Label(window,
"Username:", "sans-bold");
    nanogui::TextBox* usernameTextBox = new
nanogui::TextBox(window);
    usernameTextBox->setEditable(true);
    usernameTextBox->setPlaceholder("Enter your username");
    usernameTextBox->setValue("");

    nanogui::Label* passwordLabel = new nanogui::Label(window,
"Password:", "sans-bold");
    nanogui::TextBox* passwordTextBox = new
nanogui::TextBox(window);
    passwordTextBox->setEditable(true);
    passwordTextBox->setPlaceholder("Enter your password");
    passwordTextBox->setValue("");

    nanogui::Button* loginButton = new nanogui::Button(window,
"Login");
    loginButton->setCallback([this, usernameTextBox,
passwordTextBox] {
        enteredUsername = usernameTextBox->value();
        enteredPassword = passwordTextBox->value();
        setVisible(false);
    });

    window->setPosition(Eigen::Vector2i(0, 0));
    window->setFixedSize(Eigen::Vector2i(200, 200));
    performLayout();
}

// Получение значения из формы
std::string LoginForm::getEnteredUsername() const {
    return enteredUsername;
}
// Получение значения из формы
std::string LoginForm::getEnteredPassword() const {
    return enteredPassword;
}

```

```
// Конструктор для окна, в котором мы выбираем проекты.
ProjectSelectionWindow::ProjectSelectionWindow(User& user,
Database& coursore)
    : nanogui::Screen(Eigen::Vector2i(400, 600), "Project
Selection", false),
    selectedProjectId(-1), user(user), coursore(coursore) {
    projectsLabel = coursore.loadProjectsDescription(user);
    createProjectSelectionWindowUI();
}
```

```
// Окно, которое открывается во время нажатия кнопки Add Project.
void ProjectSelectionWindow::openAddProjectWindow() {
    nanogui::Window* taskFormWindow = new nanogui::Window(this,
    "Add Project");
    taskFormWindow->setSize(Eigen::Vector2i(300, 200));

    nanogui::Label* taskNameLabel = new
nanogui::Label(taskFormWindow, "Project Name:", "sans-bold");
    nanogui::TextBox* taskNameTextBox = new
nanogui::TextBox(taskFormWindow);
    taskNameTextBox->setEditable(true);
    taskNameTextBox->setPlaceholder("Enter Project Name");
    taskNameTextBox->setSize(Eigen::Vector2i(220, 25));
    taskNameTextBox->setPosition(Eigen::Vector2i(50, 50));
    taskNameTextBox->setValue("");

    nanogui::Label* deadlineLabel = new
nanogui::Label(taskFormWindow, "Deadline:", "sans-bold");
    nanogui::TextBox* deadlineTextBox = new
nanogui::TextBox(taskFormWindow);
    deadlineTextBox->setEditable(true);
    deadlineTextBox->setPlaceholder("Example: 2023-09-08
07:06:08");
    deadlineTextBox->setSize(Eigen::Vector2i(220, 25));
    deadlineTextBox->setPosition(Eigen::Vector2i(50, 90));
    deadlineTextBox->setValue("");

    nanogui::Button* addButton = new
nanogui::Button(taskFormWindow, "Add");
    addButton->setPosition(Eigen::Vector2i(170, 170));
    addButton->setSize(Eigen::Vector2i(100, 20));
    addButton->setCallback([this, taskNameTextBox,
deadlineTextBox, taskFormWindow] {
        std::string projectName = taskNameTextBox->value();
        std::string deadline = deadlineTextBox->value();
    });
}
```

```

        std::string dateCreation = getCurrentDateTime();

        int projectId = -1;

        Project tempProject = Project(projectName, deadline,
dateCreation, projectId);
        course.saveProject(tempProject);

        createProjectSelectionWindowUI();

        taskFormWindow->dispose();
    });

    taskFormWindow->setModal(true);
}

// Отрисовка главного окна, в котором мы выбираем проект. В
конструкторе мы просто его создаем и даем ему параметры.а здесь мы
добавляем в него различные виджеты, кнопки и тд.
void ProjectSelectionWindow::createProjectSelectionWindowUI() {
    nanogui::Label* greetingLabel = new nanogui::Label(this,
"Current User: " + user.name, "sans-bold");
    greetingLabel->setFontSize(20);

    nanogui::Button* createProjectButton = new
nanogui::Button(this, "Create Project");
    createProjectButton->setPosition(Eigen::Vector2i(270, 0));
    createProjectButton->setCallback([this] {
        openAddProjectWindow();
    });

    nanogui::Widget* buttonPanel = new nanogui::Widget(this);
    buttonPanel->setPosition(Eigen::Vector2i(0, 30));
    buttonPanel->setLayout(new
nanogui::BoxLayout(nanogui::Orientation::Vertical,
nanogui::Alignment::Middle, 0, 5));

    for (const Label& label : projectsLabel) {
        nanogui::Button* projectButton = new
nanogui::Button(buttonPanel, label.name);
        projectButton->setFixedHeight(30);
        projectButton->setFixedWidth(400);
        projectButton->setCallback([this, label] {
            selectedProjectId = label.id;
            setVisible(false);
        });
    }

    performLayout();
}

```

```

}

int ProjectSelectionWindow::getSelectedProjectIndex() const {
    return selectedProjectId;
}

// Конструктор для окна, в котором мы работаем с конкретным
// проектом. Внес completedTasks и uncompletedTasks по причине того,
// чтобы проще было отчищать их

ProjectWindow::ProjectWindow(Project& _project, Database&
coursore, User& user)
: nanogui::Screen(Eigen::Vector2i(1000, 600),
_project.getName(), false), project(_project), coursore(coursore),
user(user) {
    completedTasks = new FixedWindow(this, "Completed Tasks");
    uncompletedTasks = new FixedWindow(this, "Uncompleted Tasks");
    renderTaskWindows();

    projectName = "Project Name: " + project.getName();
    dateOfCreation = "Date of Creation: " +
project.getDateCreation();
    deadLine = "Deadline: " + project.getDeadline();
    createUI();
}

// Открыть окно, в котором показывается информация о конкретном
// проекте

void ProjectWindow::openInfoAboutProject() {
    FixedWindow* infoWindow = new FixedWindow(this, "Project
Information");
    infoWindow->setPosition(Eigen::Vector2i(0, 0));
    infoWindow->setSize(Eigen::Vector2i(500, 500));

    nanogui::Label* nameLabel = new nanogui::Label(infoWindow,
projectName, "sans-bold");
    nameLabel->setFontSize(24);
    nameLabel->setPosition(Eigen::Vector2i(5, 30));
    nameLabel->setSize(Eigen::Vector2i(300, 30));

    nanogui::Label* dateLabel = new nanogui::Label(infoWindow,
deadLine, "sans-bold");
    dateLabel->setFontSize(24);
    dateLabel->setPosition(Eigen::Vector2i(5, 90));
    dateLabel->setSize(Eigen::Vector2i(300, 30));

    nanogui::Label* deadlineLabel = new nanogui::Label(infoWindow,
dateOfCreation, "sans-bold");

```

```

deadlineLabel->setFontSize(24);
deadlineLabel->setPosition(Eigen::Vector2i(5, 60));
deadlineLabel->setSize(Eigen::Vector2i(300, 30));

for (int i = 0; i < project.getUsersAmount(); i++) {
    nanogui::TextBox*      userBox      =      new
nanogui::TextBox(infoWindow);
    userBox->setEditable(false);
    userBox->setValue("User: " + project.getUser(i));
    userBox->setSize(Eigen::Vector2i(200, 30));
    userBox->setPosition(Eigen::Vector2i(5, 120 + 50 * i));
    infoWindow->addChild(userBox);
}

nanogui::Button* closeButton = new nanogui::Button(infoWindow,
"Close");
closeButton->setPosition(Eigen::Vector2i(100, 450));
closeButton->setSize(Eigen::Vector2i(300, 40));

closeButton->setCallback([infoWindow] {
    infoWindow->dispose();
});

infoWindow->addChild(nameLabel);
infoWindow->addChild(dateLabel);
infoWindow->addChild(deadlineLabel);
infoWindow->addChild(closeButton);

infoWindow->setVisible(true);
infoWindow->setModal(true);
infoWindow->center();
}

void ProjectWindow::renderComments(Task& task, const FixedWindow&
window) {
    nanogui::Widget* commentsWidget = new nanogui::Widget(window);
    commentsWidget->setPosition(Eigen::Vector2i(20, 250));
}

// Открыть окно, в котором показывается информация о конкретной
ЗАДАЧЕ

void ProjectWindow::openTaskInfoWindow(Task& task) {
    FixedWindow* taskInfoWindow = new FixedWindow(this, "Task
Information: " + task.getName());
    taskInfoWindow->setSize(Eigen::Vector2i(width() / 2 - 20,
height() - 100));

    // Выше мы создали наше окно, которое нельзя двигать и сейчас

```


мы создаем для него дочерние объекта. различные надписи и текстовые поля.

```
nanogui::Label*          titleLabel          =          new
nanogui::Label(taskInfoWindow, "Task Name:", "sans-bold");
titleLabel->setFontSize(36);
titleLabel->setSize(Eigen::Vector2i(500, 20));
titleLabel->setPosition(Eigen::Vector2i(5, 37));

nanogui::TextBox*        tasknameTextBox      =          new
nanogui::TextBox(taskInfoWindow);
tasknameTextBox->setEditable(true);
tasknameTextBox->setValue(task.getName());
tasknameTextBox->setSize(Eigen::Vector2i(250, 25));
tasknameTextBox->setPosition(Eigen::Vector2i(80, 35));

nanogui::Label* dateLabel = new nanogui::Label(taskInfoWindow,
"Date of Creation: ", "sans-bold");
dateLabel->setSize(Eigen::Vector2i(500, 20));
dateLabel->setPosition(Eigen::Vector2i(5, 70));

nanogui::TextBox*        dateTextBox          =          new
nanogui::TextBox(taskInfoWindow);
dateTextBox->setEditable(false);
dateTextBox->setValue(task.getDateCreation());
dateTextBox->setSize(Eigen::Vector2i(250, 25));
dateTextBox->setPosition(Eigen::Vector2i(110, 67));

nanogui::Label*          deadlineLabel        =          new
nanogui::Label(taskInfoWindow, "Deadline: ", "sans-bold");
deadlineLabel->setSize(Eigen::Vector2i(500, 20));
deadlineLabel->setPosition(Eigen::Vector2i(5, 100));

nanogui::TextBox*        deadlineTextBox      =          new
nanogui::TextBox(taskInfoWindow);
deadlineTextBox->setEditable(true);
deadlineTextBox->setValue(task.getDeadline());
deadlineTextBox->setSize(Eigen::Vector2i(250, 25));
deadlineTextBox->setPosition(Eigen::Vector2i(70, 97));

nanogui::Label*          commentsLabel        =          new
nanogui::Label(taskInfoWindow, "Comments", "sans-bold");
commentsLabel->setFontSize(36);
commentsLabel->setSize(Eigen::Vector2i(500, 20));

commentsLabel->setPosition(Eigen::Vector2i(20, 180));
nanogui::TextBox*        addCommentTextBox    =          new
nanogui::TextBox(taskInfoWindow);
addCommentTextBox->setEditable(true);
addCommentTextBox->setValue("");
```

```

addCommentTextBox->setSize(Eigen::Vector2i(350, 30));
addCommentTextBox->setPosition(Eigen::Vector2i(20, 200));

// Создаем объект класса кнопки на добавление окмментария и
привязываем к нему коллбек
nanogui::Button* addCommentButton = new
nanogui::Button(taskInfoWindow, "Add");
addCommentButton->setSize(Eigen::Vector2i(80, 30));
addCommentButton->setPosition(Eigen::Vector2i(375, height() -
400));
addCommentButton->setCallback([this, taskInfoWindow,
addCommentTextBox, &task] {
    if (addCommentTextBox->value() != "") {
        std::string User = user.name;
        std::string commentText = addCommentTextBox->value();
        int id = -1;

        Comment tempComment(User, commentText, id);
        task.addComment(tempComment);
        addCommentTextBox->setValue("");
        taskInfoWindow->dispose();
        openTaskInfoWindow(task);
    }
});

nanogui::Label* statusLabel = new
nanogui::Label(taskInfoWindow, "Completed: ", "sans-bold");
statusLabel->setSize(Eigen::Vector2i(500, 20));
statusLabel->setPosition(Eigen::Vector2i(5, 130));

nanogui::CheckBox* statusCheckBox = new
nanogui::CheckBox(taskInfoWindow, "");
statusCheckBox->setChecked(task.getCurrentState());

statusCheckBox->setSize(Eigen::Vector2i(20, 20));
statusCheckBox->setPosition(Eigen::Vector2i(75, 130));

for (int i = 0; i < task.getCommentsAmount(); i++) {
    nanogui::TextBox* commentBox = new
nanogui::TextBox(taskInfoWindow);
commentBox->setEditable(false);
commentBox->setValue(task.getComment(i).author + " | " +
task.getComment(i).text);
commentBox->setSize(Eigen::Vector2i(350, 30));
commentBox->setPosition(Eigen::Vector2i(20, 250 + 50 *
i));
    taskInfoWindow->addChild(commentBox);
}

nanogui::Button* saveTaskButton = new
nanogui::Button(taskInfoWindow, "Save Task");
saveTaskButton->setSize(Eigen::Vector2i(100, 40));

```

```

        saveTaskButton->setPosition(Eigen::Vector2i(333, height() -
150));
        saveTaskButton->setCallback([this, taskInfoWindow,
tasknameTextBox, deadlineTextBox, statusCheckBox, &task] {
            task.setName(tasknameTextBox->value());
            task.setDeadLine(deadlineTextBox->value());
            task.setCurrentState(statusCheckBox->checked());
            coursore.saveProject(project);
            taskInfoWindow->dispose();
            createUI();
        });

        nanogui::Button* closeButton = new
nanogui::Button(taskInfoWindow, "Cancel");
        closeButton->setSize(Eigen::Vector2i(100, 40));
        closeButton->setPosition(Eigen::Vector2i(33, height() - 150));
        closeButton->setCallback([taskInfoWindow, &task] {
            taskInfoWindow->dispose();
        });

        nanogui::Button* deleteTaskButton = new
nanogui::Button(taskInfoWindow, "Delete Task");
        deleteTaskButton->setSize(Eigen::Vector2i(100, 40));
        deleteTaskButton->setPosition(Eigen::Vector2i(183, height() -
150));
        deleteTaskButton->setCallback([this, taskInfoWindow, &task] {
            project.deleteTask(task.id);
            coursore.deleteTask(task.id);
            createUI();
            taskInfoWindow->dispose();
        });

        taskInfoWindow->addChild(titleLabel);
        taskInfoWindow->addChild(tasknameTextBox);
        taskInfoWindow->addChild(dateLabel);
        taskInfoWindow->addChild(dateTextBox);
        taskInfoWindow->addChild(deadlineLabel);
        taskInfoWindow->addChild(deadlineTextBox);
        taskInfoWindow->addChild(statusLabel);
        taskInfoWindow->addChild(statusCheckBox);
        taskInfoWindow->addChild(addCommentTextBox);
        taskInfoWindow->addChild(closeButton);
        taskInfoWindow->addChild(addCommentButton);
        taskInfoWindow->addChild(commentsLabel);
        taskInfoWindow->center();
        taskInfoWindow->setVisible(true);
        taskInfoWindow->setModal(true);
    }

```

// Открыть окно, в котором можно добавить пользователя в ПРОЕКТ

```

void ProjectWindow::openAddUserWindow() {
    nanogui::Window* window = new nanogui::Window(this, "Add
User");

    nanogui::Label* usernameLabel = new nanogui::Label(window,
"Add User", "sans-bold");
    usernameLabel->setPosition(Eigen::Vector2i(10, 30));
    usernameLabel->setSize(Eigen::Vector2i(200, 20));

    nanogui::TextBox* usernameTextBox = new
nanogui::TextBox(window);
    usernameTextBox->setEditable(true);
    usernameTextBox->setPlaceholder("Enter username");
    usernameTextBox->setValue("");
    usernameTextBox->setSize(Eigen::Vector2i(180, 30));
    usernameTextBox->setPosition(Eigen::Vector2i(10, 60));

    nanogui::Button* loginButton = new nanogui::Button(window,
"Add");
    loginButton->setSize(Eigen::Vector2i(80, 30));
    loginButton->setPosition(Eigen::Vector2i(110, 110));
    loginButton->setCallback([this, usernameTextBox, window] {
        window->setModal(false);
        std::string answer =
coursore.addUserToProject(usernameTextBox->value(), project.id);
        if(answer == "This user succesfully added.")
            project.addUser(usernameTextBox->value());
        nanogui::Window* messageWindow = new nanogui::Window(this,
"Message");
        messageWindow->setSize(Eigen::Vector2i(200, 120));

        nanogui::Label* messageLabel = new
nanogui::Label(messageWindow, answer, "sans-bold");
        messageLabel->setPosition(Eigen::Vector2i(10, 30));
        messageLabel->setSize(Eigen::Vector2i(180, 20));

        nanogui::Button* okButton = new
nanogui::Button(messageWindow, "OK");
        okButton->setPosition(Eigen::Vector2i(20, 80));
        okButton->setSize(Eigen::Vector2i(160, 30));
        okButton->setCallback([messageWindow, window] {
            messageWindow->dispose();
            window->dispose();
        });

        messageWindow->setModal(true);
        messageWindow->setPosition(Eigen::Vector2i(500, 300));
    });
}

```

```

        nanogui::Button* closeButton = new nanogui::Button(window,
"Cancel");
        closeButton->setSize(Eigen::Vector2i(80, 30));
        closeButton->setPosition(Eigen::Vector2i(10, 110));
        closeButton->setCallback([window] {
            window->dispose();
        });

        window->setSize(nanogui::Vector2i(200, 150));
        window->addChild(closeButton);
        window->addChild(loginButton);
        window->setModal(true);
        window->center();
    }
    // Открыть окно для добавления ЗАДАЧИ в ПРОЕКТ
    void ProjectWindow::openAddTaskWindow() {
        nanogui::Window* taskFormWindow = new nanogui::Window(this,
"Add Task");
        taskFormWindow->setSize(Eigen::Vector2i(300, 200));

        nanogui::Label* taskNameLabel = new
nanogui::Label(taskFormWindow, "Task Name:", "sans-bold");
        nanogui::TextBox* taskNameTextBox = new
nanogui::TextBox(taskFormWindow);
        taskNameTextBox->setEditable(true);
        taskNameTextBox->setPlaceholder("Enter Task Name");
        taskNameTextBox->setSize(Eigen::Vector2i(220, 25));
        taskNameTextBox->setPosition(Eigen::Vector2i(50, 50));
        taskNameTextBox->setValue("");

        nanogui::Label* deadlineLabel = new
nanogui::Label(taskFormWindow, "Deadline:", "sans-bold");
        nanogui::TextBox* deadlineTextBox = new
nanogui::TextBox(taskFormWindow);
        deadlineTextBox->setEditable(true);
        deadlineTextBox->setPlaceholder("Example: 2023-09-08
07:06:08");
        deadlineTextBox->setSize(Eigen::Vector2i(220, 25));
        deadlineTextBox->setPosition(Eigen::Vector2i(50, 90));
        deadlineTextBox->setValue("");

        nanogui::Button* addButton = new
nanogui::Button(taskFormWindow, "Add");
        addButton->setPosition(Eigen::Vector2i(170, 170));
        addButton->setSize(Eigen::Vector2i(100, 20));
        addButton->setCallback([this, taskNameTextBox,
deadlineTextBox, taskFormWindow] {
            std::string taskName = taskNameTextBox->value();
            std::string deadline = deadlineTextBox->value();
            bool isCompleted = 0;

```

```

        std::string dateCreation = getCurrentDateTime();

        int taskId = -1;
        Task tempTask = Task(taskName, dateCreation, deadline,
isCompleted, taskId);
        project.addTask(tempTask);
        coursore.saveProject(project);
        taskFormWindow->dispose();
        createUI();
    });

    nanogui::Button*          closeButton          =          new
nanogui::Button(taskFormWindow, "Close");
    closeButton->setPosition(Eigen::Vector2i(30, 170));
    closeButton->setSize(Eigen::Vector2i(100, 20));
    closeButton->setCallback([this, taskFormWindow] {
        taskFormWindow->dispose();
    });

    taskFormWindow->center();
    taskFormWindow->setVisible(true);
    taskFormWindow->addChild(taskNameLabel);
    taskFormWindow->addChild(taskNameTextBox);
    taskFormWindow->addChild(deadlineLabel);
    taskFormWindow->addChild(deadlineTextBox);
    taskFormWindow->addChild(addButton);
    taskFormWindow->setModal(true);
}

//Метод который отрисовывает два подокна с задачами(Выполненные и
Невыполненные).

void ProjectWindow::renderTaskWindows() {

    completedTasks->dispose();
    uncompletedTasks->dispose();

    completedTasks = new FixedWindow(this, "Completed Tasks");
    uncompletedTasks = new FixedWindow(this, "Uncompleted Tasks");
    completedTasks->setPosition(Eigen::Vector2i(width() / 2 + 10,
90));
    completedTasks->setSize(Eigen::Vector2i(width() / 2 - 20,
height() - 100));
    uncompletedTasks->setPosition(Eigen::Vector2i(10, 90));
    uncompletedTasks->setSize(Eigen::Vector2i(width() / 2 - 20,
height() - 100));

    int first = 0;
    int second = 0;
    for (int i = 0; i < project.getTasksAmount(); ++i) {

```

```

        nanogui::Button* openButton = new nanogui::Button(
            project.getTask(i).getCurrentState() ? completedTasks
: uncompletedTasks,
            project.getTask(i).getName()
        );

        openButton->setFixedSize(Eigen::Vector2i(300, 40));
        openButton->setPosition(Eigen::Vector2i(100, 40 + 60 *
(project.getTask(i).getCurrentState() ? second : first)));
        openButton->setCallback([this, i] {
            openTaskInfoWindow(project.getTask(i));
        });

        if (project.getTask(i).getCurrentState()) {
            second++;
        }
        else {
            first++;
        }
    }
}

void ProjectWindow::createUI() {

    project = cursore.loadProject(project.id);

    nanogui::Label* projectInfo = new nanogui::Label(this,
projectName, "sans-bold");
    projectInfo->setFontSize(20);
    projectInfo->setPosition(Eigen::Vector2i(350, 5));
    projectInfo->setSize(Eigen::Vector2i(300, 30));
    int buttonWidth = 100;
    int buttonHeight = 30;
    int margin = 10;
    int rightMargin = 10;

    int buttonPanelX = width() - buttonWidth - rightMargin;
    int buttonPanelY = height() - margin - buttonHeight;

    nanogui::Button* infoButton = new nanogui::Button(this,
"Info");
    infoButton->setSize(Eigen::Vector2i(buttonWidth,
buttonHeight));
    infoButton->setPosition(Eigen::Vector2i(buttonPanelX,
buttonPanelY));

    infoButton->setCallback([this]() {
        openInfoAboutProject();
    });

    nanogui::Button* addUserButton = new nanogui::Button(this,

```

```

"Add User");
    addUserButton->setCallback([this] {
        openAddUserWindow();
    });

    nanogui::Button* closeButton = new nanogui::Button(this,
"Close");
    closeButton->setCallback([this] { setVisible(false); });

    nanogui::Widget* buttonPanel = new nanogui::Widget(this);
    buttonPanel->setLayout(new
nanogui::BoxLayout(nanogui::Orientation::Horizontal,
nanogui::Alignment::Minimum, 0, 10));
    buttonPanel->setFixedSize(nanogui::Vector2i(buttonWidth,
buttonHeight * 2 + 10));
    buttonPanel->setPosition(nanogui::Vector2i(buttonPanelX,
buttonPanelY));

    buttonPanel->addChild(infoButton);
    buttonPanel->addChild(addUserButton);
    buttonPanel->addChild(closeButton);

    buttonPanel->setPosition(Eigen::Vector2i(400, height() -
buttonPanel->height()));
    nanogui::Widget* taskWidget = new nanogui::Widget(this);
    taskWidget->setPosition(Eigen::Vector2i(10, 10));

    renderTaskWindows();

    nanogui::Button* addTaskButton = new nanogui::Button(this,
"Add Task");
    addTaskButton->setPosition(Eigen::Vector2i(width() - 130, 0));
    addTaskButton->setCallback([this] {
        openAddTaskWindow();
    });

    performLayout();
}

```