**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИИ**  
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

Расчетно-графическое задание

Дисциплина: Обьектно-ориентированное программирование

по теме «Система блогов»

Выполнил: ст. группы ВТ-22  
Ковалев Павел Александрович

Проверил: Буханов Д.Г.

**Белгород 2019Цель работы**: приобретение практических навыков создания приложений на языке С++.

**Вариант задания 13**

Создать приложение реализующее систему блогов.

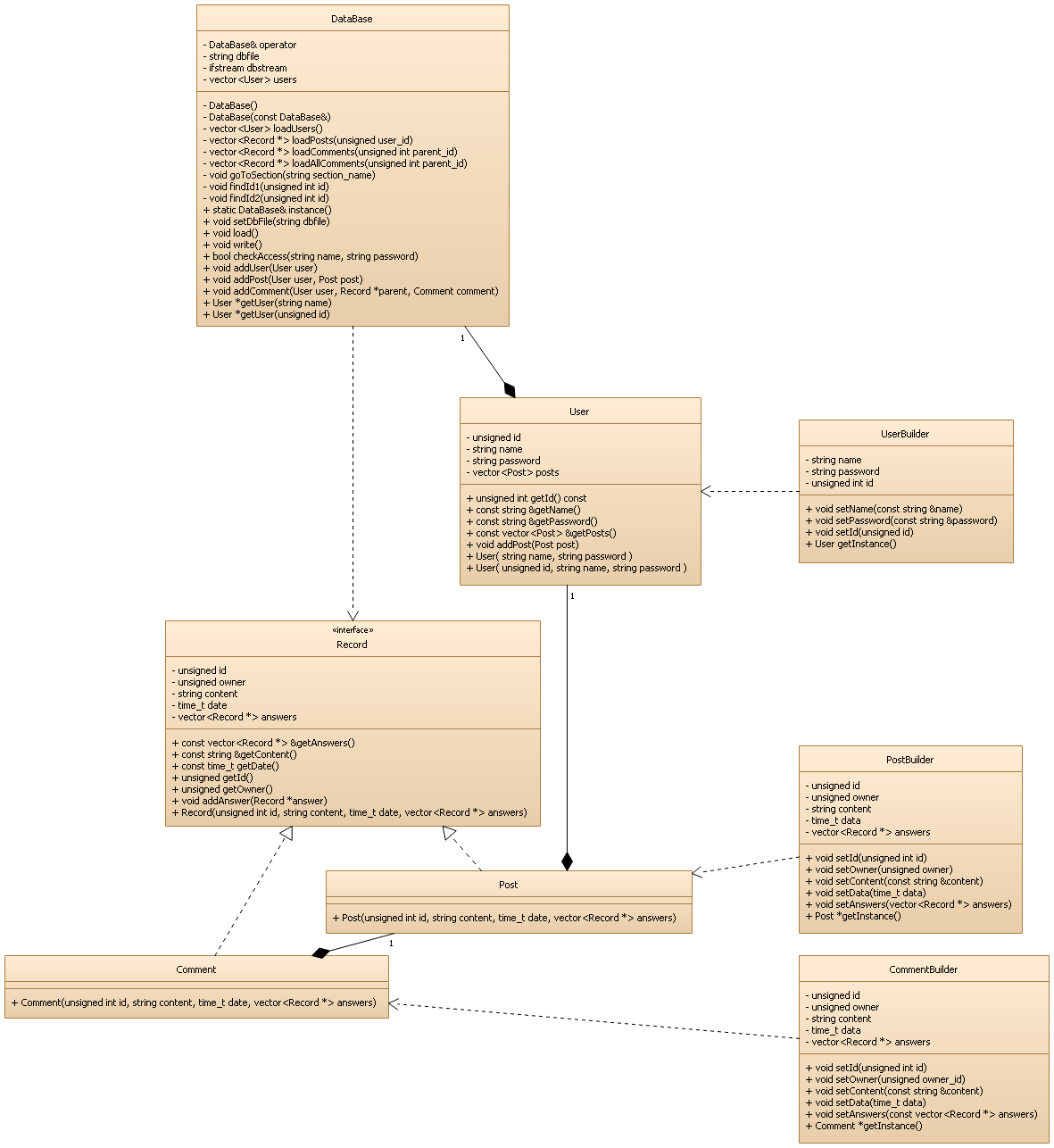
**Постановка задачи:**

Каждый блог принадлежит одному пользователю и состоит из нескольких записей. В момент создания блога в нем записей нет. Пользователь может создавать записи лишь в собственном блоге. Читать записи любого пользователя может каждый пользователь. В каждой записи есть заголовок, дата, текст. На каждой странице блога находятся ссылка на профиль. На начальной странице блога отображаются 10 последних записей (или менее, если в блоге их недостаточно). Если записей в блоге больше 10, то с начальной страницы можно перейти на вторую, где отображается второй десяток записей и т. д. Записи в блоге упорядочены по убыванию даты. Любая запись может быть отредактирована, но дата записи не может быть изменена. Запись может быть удалена автором. Перейдя на страницу отдельной записи, пользователь может ее комментировать, независимо от того, чужая это запись или его собственная. Комментарий состоит из текста и даты, указывающей, когда он написан. Комментарий может быть дан к записи блога или к ранее написанному комментарию. Пользователь может запретить комментирование любой записи в своем блоге. Комментарии, полученные раньше запрета, сохраняются. Запрет на комментирование может быть снят. Все комментарии к записи отображаются под ее текстом в виде дерева. Каждый комментарий помечен именем его автора. Имя может быть использовано для перехода к начальной странице блога с этим именем. Комментарии не могут быть отредактированы, но могут быть удалены автором записи. При удалении комментария также стираются все ответы на него, если таковые были. Любой желающий вести блог должен зарегистрироваться. При регистрации пользователь заполняет форму, указывая желаемое имя, пароль, список своих интересов, дату рождения, контакты, краткие сведения о себе. Если имя не занято, и форма заполнена полностью, система заводит блог нового пользователя. Сведения из регистрационной формы (кроме пароля) отображаются на странице профиля пользователя. Пользователь может отредактировать собственный профиль, изменив в нем все, кроме имени. Пользователь может изменить свой пароль. Пароль не может быть короче 8 символов. Зарегистрированный пользователь, осуществляет вход в собственный блог, указывая имя и пароль. Все описанные выше действия с записями, комментариями и собственным профилем, действия с друзьями доступны пользователю только после входа. Пользователь может полностью удалить свой блог со всеми записями, комментариями и профилем. После удаления блога регистрационные данные стираются из системы.

**Диаграмма объектов**



**Диаграмма классов**



**Листинг программы:**

**Файл main.cpp**

#include "DataBase.h"

#include "Ui.h"

int main() {

DataBase& db = DataBase::instance();

db.load();

Ui ui;

ui.loginDialog();

db.write();

return 0;

}

**Файл DataBase.h**

#ifndef DATABASE\_H

#define DATABASE\_H

#include <fstream>

#include "User.h"

#include "Comment.h"

#include "Post.h"

#include "UserBuilder.h"

#include "CommentBuilder.h"

#include "PostBuilder.h"

class DataBase{

private:

DataBase(){};

DataBase(const DataBase&);

DataBase& operator=(const DataBase&);

string dbfile;

ifstream dbstream;

vector<User> users;

vector<User> loadUsers();

vector<Record \*> loadPosts(unsigned user\_id);

vector<Record \*> loadComments(unsigned int parent\_id);

vector<Record \*> loadAllComments(unsigned int parent\_id);

void goToSection(string section\_name);

void findId1(unsigned int id);

void findId2(unsigned int id);

public:

static DataBase& instance();

void setDbFile(string dbfile);

void load();

void write();

bool checkAccess(string name, string password);

void addUser(User user);

void addPost(User user, Post post);

void addComment(User user, Record \*parent, Comment comment);

User \*getUser(string name);

User \*getUser(unsigned id);

};

#endif //DATABASE\_H

**Файл Record.h**

#ifndef RECORD\_H

#define RECORD\_H

#include <ctime>

#include <string>

#include <vector>

using namespace std;

class Record{

private:

unsigned id;

unsigned owner;

string content;

time\_t date;

vector<Record \*> answers;

public:

const vector<Record \*> &getAnswers() const {

return answers;

}

const string &getContent() const {

return content;

}

const time\_t getDate() const {

return date;

}

unsigned getId() const{

return id;

}

unsigned getOwner() const{

return owner;

}

void addAnswer(Record \*answer);

Record(unsigned int id, string content, time\_t date, vector<Record \*> answers) : id(id), content(content), date(date), answers(answers) {}

};

#endif //RECORD\_H

**Файл Comment.h**

#ifndef COMMENT\_H

#define COMMENT\_H

#include "Record.h"

class Comment : public Record{

public:

Comment(unsigned int id, string content, time\_t date, vector<Record \*> answers) : Record(id, content, date, answers){};

};

#endif //UNTITLED1\_COMMENT\_H

**Файл Post.h**

#ifndef UNTITLED1\_POST\_H

#define UNTITLED1\_POST\_H

#include "Record.h"

class Post : public Record{

public:

Post(unsigned int id, string content, time\_t date, vector<Record \*> answers) : Record(id, content, date, answers){};

};

#endif //POST\_H

**Файл User.h**

#ifndef USER\_H

#define USER\_H

#include "Post.h"

class User {

private:

unsigned id;

std::string name;

public:

unsigned int getId() const;

const std::string &getName() const;

const std::string &getPassword() const;

const std::vector<Post> &getPosts() const;

void addPost(Post post);

private:

std::string password;

std::vector<Post> posts;

public:

User(

std::string name,

std::string password

);

User(

unsigned id,

std::string name,

std::string password

);

};

#endif //USER\_H

**Файл Ui.h**

#ifndef UI\_H

#define UI\_H

#include <iostream>

using namespace std;

#include "User.h"

#include "Post.h"

#include "Comment.h"

#include "UserBuilder.h"

#include "PostBuilder.h"

#include "CommentBuilder.h"

#include "DataBase.h"

class Ui{

public:

void loginDialog();

void registrationDialog();

void userDialog(string user\_name);

void postDialog(Post post);

void commentDialog(Record comment);

};

#endif //UI\_H

**Файл UserBuilder.h**

#ifndef USERBUILDER\_H

#define USERBUILDER\_H

#include "User.h"

class UserBuilder{

private:

string name;

string password;

unsigned int id;

public:

void setName(const string &name);

void setPassword(const string &password);

void setId(unsigned id);

User getInstance();

};

#endif //USERBUILDER\_H

**Файл PostBuilder.h**

#ifndef POSTBUILDER\_H

#define POSTBUILDER\_H

#include <string>

#include <ctime>

#include <vector>

#include "Post.h"

using namespace std;

class PostBuilder{

private:

unsigned id;

unsigned owner;

string content;

time\_t data;

vector<Record \*> answers;

public:

void setId(unsigned int id);

void setOwner(unsigned owner);

void setContent(const string &content);

void setData(time\_t data);

void setAnswers(vector<Record \*> answers);

Post \*getInstance();

};

#endif //POSTBUILDER\_H

**Файл CommentBuilder.h**

#ifndef COMMENTBUILDER\_H

#define COMMENTBUILDER\_H

#include <string>

#include <ctime>

#include <vector>

using namespace std;

#include "Comment.h"

class CommentBuilder{

private:

unsigned id;

unsigned owner;

string content;

time\_t data;

vector<Record \*> answers;

public:

void setId(unsigned int id);

void setOwner(unsigned owner\_id);

void setContent(const string &content);

void setData(time\_t data);

void setAnswers(const vector<Record \*> answers);

Comment \*getInstance();

};

#endif //COMMENTBUILDER\_H

**Файл DataBase.cpp**

#include "DataBase.h"

vector<User> DataBase::loadUsers() {

vector<User> result;

this->goToSection("Users");

char temp[256];

UserBuilder user\_builder;

this->dbstream.getline(temp, 256);

while (temp[0] != '['){

user\_builder.setId(atoi(temp));

this->dbstream.getline(temp, 256);

user\_builder.setName(temp);

this->dbstream.getline(temp, 256);

user\_builder.setPassword(temp);

this->dbstream.getline(temp, 256);

result.push\_back(user\_builder.getInstance());

}

for(auto it = result.begin(); it != result.end(); it++){

auto id = it->getId();

this->loadPosts(id);

}

return result;

}

vector<Record \*> DataBase::loadPosts(unsigned user\_id) {

vector<Record \*> result;

this->goToSection("UserPosts");

this->findId1(user\_id);

int count;

dbstream >> count;

vector<unsigned> posts\_ids;

unsigned temp\_id;

char temp[256];

for (int i = 0; i<count; i++){

dbstream >> temp\_id;

posts\_ids.push\_back(temp\_id);

}

PostBuilder post\_builder;

for(auto it = posts\_ids.begin(); it != posts\_ids.end(); it++){

this->goToSection("Posts");

this->findId2(\*it);

this->dbstream.getline(temp, 256);

post\_builder.setId(atoi(temp));

this->dbstream.getline(temp, 256);

post\_builder.setOwner(atoi(temp));

this->dbstream.getline(temp, 256);

post\_builder.setData(atoi(temp));

this->dbstream.getline(temp, 256);

post\_builder.setContent(temp);

post\_builder.setAnswers(this->loadComments(\*it));

result.push\_back(post\_builder.getInstance());

}

return result;

}

vector<Record \*> DataBase::loadComments(unsigned parent\_id) {

vector<Record \*> result;

this->goToSection("PostComments");

this->findId1(parent\_id);

int count;

dbstream >> count;

vector<unsigned> answers\_ids;

unsigned temp\_id;

char temp[256];

for (int i = 0; i<count; i++){

dbstream >> temp\_id;

answers\_ids.push\_back(temp\_id);

}

CommentBuilder comment\_builder;

for(auto it = answers\_ids.begin(); it != answers\_ids.end(); it++){

this->goToSection("Comments");

this->findId2(\*it);

this->dbstream.getline(temp, 256);

comment\_builder.setId(\*it);

this->dbstream.getline(temp, 256);

comment\_builder.setOwner(atoi(temp));

this->dbstream.getline(temp, 256);

comment\_builder.setData(atoi(temp));

this->dbstream.getline(temp, 256);

comment\_builder.setContent(temp);

comment\_builder.setAnswers(this->loadComments(\*it));

result.push\_back(comment\_builder.getInstance());

}

return result;

}

void DataBase::load() {

this->users = this->loadUsers();

}

void DataBase::write() {

ofstream exportstream("export.txt");

// ofstream exportstream(dbfile);

auto users = loadUsers();

exportstream << "[Users]" << endl;

for (auto it = users.begin(); it != users.end(); it++){

exportstream << it->getId() << endl;

exportstream << it->getName() << endl;

exportstream << it->getPassword() << endl;

}

exportstream << "[UserPosts]" << endl;

for (auto it = users.begin(); it != users.end(); it++) {

auto posts = it->getPosts();

exportstream << it->getId() << " ";

exportstream << posts.size() << " ";

for (auto p = posts.begin(); p != posts.end(); p++) {

exportstream << p->getId() << " ";

}

}

exportstream << "[Posts]" << endl;

for (auto it = users.begin(); it != users.end(); it++){

auto posts = it->getPosts();

for (auto p = posts.begin(); p != posts.end(); p++){

exportstream << p->getId() << endl;

exportstream << p->getOwner() << endl;

exportstream << p->getDate() << endl;

exportstream << p->getContent() << endl;

}

}

exportstream << "[PostComments]" << endl;

for (auto it = users.begin(); it != users.end(); it++){

auto posts = it->getPosts();

for (auto p = posts.begin(); p != posts.end(); p++){

exportstream << p->getId() << " ";

auto comments = this->loadAllComments(p->getId());

exportstream << comments.size() << " ";

for (auto c = comments.begin(); c!= comments.end(); c++){

exportstream << (\*c)->getId() << " ";

}

}

}

exportstream << "[Comments]" << endl;

for (auto it = users.begin(); it != users.end(); it++){

auto posts = it->getPosts();

for (auto p = posts.begin(); p != posts.end(); p++){

exportstream << p->getId() << " ";

auto comments = this->loadAllComments(p->getId());

exportstream << comments.size() << " ";

for (auto c = comments.begin(); c!= comments.end(); c++){

exportstream << (\*c)->getId() << endl;

exportstream << (\*c)->getOwner() << endl;

exportstream << (\*c)->getDate() << endl;

exportstream << (\*c)->getContent() << endl;

}

}

}

exportstream.close();

}

void DataBase::setDbFile(string dbfile){

this->dbfile = dbfile;

this->dbstream.open(dbfile);

}

DataBase &DataBase::instance() {

static DataBase instance;

return instance;

}

void DataBase::goToSection(string section\_name) {

dbstream.seekg(0, dbstream.beg);

char temp[256];

do{

dbstream.getline(temp, 256);

}while (string(temp) != "[" + section\_name + "]");

}

void DataBase::findId1(unsigned id){

unsigned temp\_id = 0;

char temp[256];

dbstream >> temp\_id;

while (temp\_id != id){

dbstream.getline(temp, 256);

dbstream >> temp\_id;

}

}

void DataBase::findId2(unsigned id){

unsigned temp\_id;

char temp[256];

dbstream >> temp\_id;

while (temp\_id != id){

for(int i = 0; i<4; i++){

dbstream.getline(temp, 256);

}

dbstream >> temp\_id;

}

}

vector<Record \*> DataBase::loadAllComments(unsigned int parent\_id) {

vector<Record \*> result;

this->goToSection("PostComments");

this->findId1(parent\_id);

int count;

dbstream >> count;

vector<unsigned> answers\_ids;

unsigned temp\_id;

char temp[256];

for (int i = 0; i<count; i++){

dbstream >> temp\_id;

answers\_ids.push\_back(temp\_id);

}

CommentBuilder comment\_builder;

for(auto it = answers\_ids.begin(); it != answers\_ids.end(); it++){

this->goToSection("Comments");

this->findId2(\*it);

this->dbstream.getline(temp, 256);

comment\_builder.setId(\*it);

this->dbstream.getline(temp, 256);

comment\_builder.setOwner(atoi(temp));

this->dbstream.getline(temp, 256);

comment\_builder.setData(atoi(temp));

this->dbstream.getline(temp, 256);

comment\_builder.setContent(temp);

auto answers = this->loadAllComments(\*it);

comment\_builder.setAnswers(answers);

result.insert(result.end(), answers.begin(), answers.end());

result.push\_back(comment\_builder.getInstance());

}

return result;

}

bool DataBase::checkAccess(string name, string password) {

auto users = loadUsers();

for (auto it = users.begin(); it!=users.end(); it++){

if (it->getName() == name && it->getPassword() == password){

return true;

}

}

return false;

}

void DataBase::addUser(User user){

this->users.push\_back(user);

}

void DataBase::addPost(User user, Post post){

for(auto it = users.begin(); it != users.end(); it++){

if (it->getId() == user.getId()){

it->addPost(post);

}

}

}

void DataBase::addComment(User user, Record \*parent, Comment comment){

parent->addAnswer(&comment);

}

User \*DataBase::getUser(string name) {

for (auto it = users.begin(); it != users.end(); it++){

if (it->getName() == name){

return &(\*it);

}

}

return nullptr;

}

User \*DataBase::getUser(unsigned id) {

for (auto it = users.begin(); it != users.end(); it++){

if (it->getId() == id){

return &(\*it);

}

}

return nullptr;

}

**Файл Record.cpp**

#include "Record.h"

void Record::addAnswer(Record \*answer) {

answers.push\_back(answer);

}

**Файл User.cpp**

#include <ctime>

#include <vector>

#include <string>

#include <fstream>

#include <sstream>

#include <iostream>

#include "User.h"

User::User(std::string name, std::string password) : name(name), password(password) {}

User::User(unsigned id, std::string name, std::string password) : id(id), name(name), password(password) {}

unsigned int User::getId() const {

return id;

}

const std::string &User::getName() const {

return name;

}

const std::string &User::getPassword() const {

return password;

}

const std::vector<Post> &User::getPosts() const {

return posts;

}

**Файл Ui.cpp**

#include "Ui.h"

#include <QTextCodec>

void Ui::**loginDialog**() {

system("cls");

cout << QString::fromUtf8("Вы находитесь на странице входа").toLocal8Bit().data() << endl;

string name;

string password;

cout << QString::fromUtf8("Введите имя пользователя:").toLocal8Bit().data() << endl;

cin >> name;

cout << QString::fromUtf8("Введите пароль:").toLocal8Bit().data() << endl;

cin >> password;

if (DataBase::instance().checkAccess(name, password)){

cout << QString::fromUtf8("Вход успешен. В блог какого пользователя перейти? ").toLocal8Bit().data() << endl;

cin >> name;

userDialog(name);

}else{

cout << QString::fromUtf8("Пользователь не зарегистрирован. Повторить попытку (1), перейти к регистрации (2) или выйти (0)?").toLocal8Bit().data() << endl;

int choice;

cin >> choice;

if (choice == 1){

loginDialog();

}else if (choice == 2){

registrationDialog();

}

}

}

void Ui::**registrationDialog**() {

system("cls");

cout << QString::fromUtf8("Вы находитесь на странице регистрации").toLocal8Bit().data() << endl;

UserBuilder ub;

string name;

cout << QString::fromUtf8("Введите имя пользователя:").toLocal8Bit().data() << endl;

cin >> name;

ub.setName(name);

string password;

cout << QString::fromUtf8("Введите пароль:").toLocal8Bit().data() << endl;

cin >> password;

ub.setPassword(password);

DataBase::instance().addUser(ub.getInstance());

loginDialog();

}

void Ui::**userDialog**(string user\_name) {

system("cls");

cout << QString::fromUtf8("Вы просматриваете информацию о пользователе").toLocal8Bit().data() << endl;

User \*u = DataBase::instance().getUser(user\_name);

cout << QString::fromUtf8("Имя: ").toLocal8Bit().data() << u->getName() << endl;

cout << QString::fromUtf8("Созданные посты: ").toLocal8Bit().data() << endl;

for (auto it = u->getPosts().begin(); it != u->getPosts().end(); it++){

cout << it->getId() << endl;

}

cout << QString::fromUtf8("Введите номер поста для просмотра или 0 чтобы выйти").toLocal8Bit().data() << endl;

int choice;

cin >> choice;

if (choice){

for (auto it = u->getPosts().begin(); it != u->getPosts().end(); it++){

if (it->getId() == choice){

postDialog(\*it);

}

}

}

}

void Ui::**postDialog**(Post post) {

system("cls");

cout << QString::fromUtf8("Вы просматриваете информацию о посте").toLocal8Bit().data() << endl;

cout << QString::fromUtf8("Автор поста: ").toLocal8Bit().data() << DataBase::instance().getUser(post.getOwner()) << endl;

time\_t raw\_time = post.getDate();

cout << QString::fromUtf8("Дата создания: ").toLocal8Bit().data() << asctime(localtime(&raw\_time));

cout << QString::fromUtf8("Содержание: ").toLocal8Bit().data() << endl << post.getContent() << endl;

cout << QString::fromUtf8("Комментарии: ").toLocal8Bit().data() << endl;

for (auto it = post.getAnswers().begin(); it != post.getAnswers().end(); it++){

cout << (\*it)->getId() << endl;

}

cout << QString::fromUtf8("Введите номер комментария для просмотра или 0 чтобы выйти").toLocal8Bit().data() << endl;

int choice;

cin >> choice;

if (choice){

for (auto it = post.getAnswers().begin(); it != post.getAnswers().end(); it++){

if ((\*it)->getId() == choice){

commentDialog(\*\*it);

}

}

}

}

void Ui::**commentDialog**(Record comment) {

system("cls");

cout << QString::fromUtf8("Вы просматриваете информацию о комментарии").toLocal8Bit().data() << endl;

cout << QString::fromUtf8("Автор комментария: ").toLocal8Bit().data() << DataBase::instance().getUser(comment.getOwner()) << endl;

time\_t raw\_time = comment.getDate();

cout << QString::fromUtf8("Дата создания: ").toLocal8Bit().data() << asctime(localtime(&raw\_time));

cout << QString::fromUtf8("Содержание: ").toLocal8Bit().data() << endl << comment.getContent() << endl;

cout << QString::fromUtf8("Ответы: ").toLocal8Bit().data() << endl;

for (auto it = comment.getAnswers().begin(); it != comment.getAnswers().end(); it++){

cout << (\*it)->getId() << endl;

}

cout << QString::fromUtf8("Введите номер комментария для просмотра или 0 чтобы выйти").toLocal8Bit().data() << endl;

int choice;

cin >> choice;

if (choice){

for (auto it = comment.getAnswers().begin(); it != comment.getAnswers().end(); it++){

if ((\*it)->getId() == choice){

commentDialog(\*\*it);

}

}

}

}

**Файл UserBuilder.cpp**

#include "UserBuilder.h"

void UserBuilder::setName(const string &name) {

this->name = name;

}

void UserBuilder::setPassword(const string &password) {

this->password = password;

}

User UserBuilder::getInstance() {

return User(this->id, this->name, this->password);

}

void UserBuilder::setId(unsigned id) {

this->id = id;

}

**Файл PostBuilder.cpp**

#include "PostBuilder.h"

void PostBuilder::setId(unsigned int id) {

this->id = id;

}

void PostBuilder::setOwner(unsigned owner) {

this->owner = owner;

}

void PostBuilder::setContent(const string &content) {

this->content = content;

}

void PostBuilder::setData(time\_t data) {

this->data = data;

}

void PostBuilder::setAnswers(vector<Record \*> answers) {

this->answers = answers;

}

Post \* PostBuilder::getInstance() {

return new Post(

this->id,

this->content,

this->data,

this->answers

);

}

**Файл CommentBuilder.cpp**

#include "CommentBuilder.h"

void CommentBuilder::setId(unsigned int id) {

this->id = id;

}

void CommentBuilder::setOwner(unsigned owner\_id) {

this->owner = owner;

}

void CommentBuilder::setContent(const string &content) {

this->content = content;

}

void CommentBuilder::setData(time\_t data) {

this->data = data;

}

void CommentBuilder::setAnswers(const vector<Record \*> answers) {

this->answers = answers;

}

Comment \*CommentBuilder::getInstance() {

return new Comment(

this->id,

this->content,

this->data,

this->answers

);

}