Файл accountstable.cpp:

#include "accountstable.h"

AccountsTable::AccountsTable(DBController::AccountsController\* controller) : QTableWidget(0, 3) {

Controller = controller;

QStringList list = QStringList();

list.push\_back("ID");

list.push\_back("First name");

list.push\_back("Second name");

this->setHorizontalHeaderLabels(list); }

Файл adminwindow.cpp:

#include "ui\_adminwindow.h"

#include "adminwindow.h"

AdminWindow::AdminWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::AdminWindow) {

ui->setupUi(this);

ui->tableWidget->UpdateTable();

ui->SchedulesTable->UpdateTable();

UpdateGroupsScrean();

UpdateScheduleScrean();

ui->Calendar->setMinimumDate(QDate::currentDate().addDays(1));

AdminWindow::connect(ui->tableWidget, &AccountsTable::AccountDBChanged,

this, &AdminWindow::SetAccountToDisplayWidget);

AdminWindow::connect(ui->GroupsTable, &GroupsDBTable::AccountDBChanged,

this, &AdminWindow::SetGroup);

AdminWindow::connect(ui->GroupsTableSch, &GroupsDBTable::AccountDBChanged,

this, &AdminWindow::SetGroupSch);

AdminWindow::connect(ui->DeleteButton, &QAbstractButton::clicked,

this, &AdminWindow::DeleteAccount);

AdminWindow::connect(ui->UpdateButton, &QAbstractButton::clicked,

this, &AdminWindow::UpdateAccount);

AdminWindow::connect(ui->AddButton, &QAbstractButton::clicked,

this, &AdminWindow::AddNewAccount);

AdminWindow::connect(ui->DeleteGroup, &QAbstractButton::clicked,

this, &AdminWindow::DeleteGroup);

AdminWindow::connect(ui->UpdateGroup, &QAbstractButton::clicked,

this, &AdminWindow::UpdateGroup);

AdminWindow::connect(ui->AddGroup, &QAbstractButton::clicked,

this, &AdminWindow::AddNewGroup);

AdminWindow::connect(ui->GroupedAccs, &GroupedAccountsTable::AddAcc,

this, &AdminWindow::AddAccToGroup);

AdminWindow::connect(ui->NotAssignedAccs, &UngroupedAccountsTable::DelAcc,

this, &AdminWindow::DeleteAccFromGroup);

AdminWindow::connect(ui->tabWidget, &QTabWidget::currentChanged,

this, &AdminWindow::ChangeCurentSize);

AdminWindow::connect(ui->TaeachersTable, &TeachersTable::AccountDBChanged,

this, &AdminWindow::SetAccToSchedule);

AdminWindow::connect(ui->AddClass, &QAbstractButton::clicked,

this, &AdminWindow::AddClass);

AdminWindow::connect(ui->DeleteClass, &QAbstractButton::clicked,

this, &AdminWindow::DeleteClass);

AdminWindow::connect(ui->Calendar, &QCalendarWidget::clicked,

this, &AdminWindow::DataChanged);

AdminWindow::connect(ui->SchedulesTable, &ScheduleTable::AddSchedule,

this, &AdminWindow::AddSchedule);

AdminWindow::connect(ui->ClassesTable, &PosibleLessonsTable::DeleteSchedule,

this, &AdminWindow::DeleteSchedule);

SetGroup(NULL);

initialWidth = width(); }

void AdminWindow::resizeEvent(QResizeEvent \*event) {

switch (ui->tabWidget->currentIndex()) {

case 0:

initialWidth = width();

break;

case 1:

initialWidth = ((width() / 4) \* 3);

break;

case 2:

initialWidth = (width() / 2);

break; }

QMainWindow::resizeEvent(event); }

void AdminWindow::ChangeCurentSize(int tab) {

switch (tab) {

case 0:

resize(initialWidth, height());

UpdateGroupsScrean();

break;

case 1:

resize(((initialWidth \* 4) / 3), height());

UpdateScheduleScrean();

break;

case 2:

resize((initialWidth \* 2), height());

break; } }

AdminWindow::~AdminWindow() {

delete ui; }

void AdminWindow::SetAccountToDisplayWidget(AccountDB \*acc) {

ui->AccountDisp->setAccountDB(acc); }

void AdminWindow::SetGroup(GroupDB \*grp) {

group = grp;

allCurators.clear();

ui->GroupedAccs->setGroup(grp);

auto accs = DBController::GetInstance()->accs.GetAll();

for(auto acc:accs) {

if(acc->getAdditionalInfo() != NULL) {

auto teach = dynamic\_cast<TeacherInfoDB\*>(acc->getAdditionalInfo());

if(teach != NULL) {

this->allCurators.push\_back(teach); } } }

ui->CuratorComboBox->clear();

ui->CuratorComboBox->addItem("None");

for(auto cur:allCurators) {

ui->CuratorComboBox->addItem((cur->FirstName + " " + cur->SecondName).c\_str()); }

ui->CuratorComboBox->setCurrentText("None");

if(group == NULL) {

ui->GroupNameEdit->setText(""); }

else {

if(group->getCurator() != NULL) {

ui->CuratorComboBox->setCurrentText((group->getCurator()->FirstName + " " + group->getCurator()->SecondName).c\_str()); }

ui->GroupNameEdit->setText(group->Name.c\_str()); } }

void AdminWindow::SetGroupSch(GroupDB \*acc) {

UpdateScheduleScrean(NULL, acc); }

void AdminWindow::SetAccToSchedule(AccountDB \*acc) {

teacher = dynamic\_cast<TeacherInfoDB\*>(acc->getAdditionalInfo());

ui->ClassesTable->setTeacher(dynamic\_cast<TeacherInfoDB\*>(acc->getAdditionalInfo()));

UpdateScheduleScrean(NULL , NULL); }

void AdminWindow::AddNewAccount() {

auto data = ui->AccountDisp->getNewAccount();

auto res = DBController::GetInstance()->accs.Add(data.\_Myfirst.\_Val);

if(data.\_Get\_rest().\_Myfirst.\_Val != NULL) {

auto Info = data.\_Get\_rest().\_Myfirst.\_Val;

Info->setOwner(res);

DBController::GetInstance()->infos.Add(Info, res->AccountType); }

ui->tableWidget->UpdateTable();

UpdateGroupsScrean(); }

void AdminWindow::DeleteAccount() {

auto data = ui->AccountDisp->getAccountForDelete();

if(data == NULL) {

return; }

if(data->getAdditionalInfo() != NULL) {

DBController::GetInstance()->infos.Delete(data->getAdditionalInfo()->\_id); }

DBController::GetInstance()->accs.Delete(data->\_id);

ui->tableWidget->UpdateTable();

ui->AccountDisp->setAccountDB(NULL); }

void AdminWindow::UpdateAccount() {

auto data = ui->AccountDisp->getAccountForDBUpdate();

if(data.\_Myfirst.\_Val == NULL) {

return; }

DBController::GetInstance()->accs.Update(data.\_Myfirst.\_Val);

if(data.\_Get\_rest().\_Myfirst.\_Val != NULL) {

DBController::GetInstance()->infos.Delete(data.\_Get\_rest().\_Myfirst.\_Val->\_id); }

if(data.\_Get\_rest().\_Get\_rest().\_Myfirst.\_Val != NULL) {

auto Info = data.\_Get\_rest().\_Get\_rest().\_Myfirst.\_Val;

Info->setOwner(data.\_Myfirst.\_Val);

DBController::GetInstance()->infos.Add(data.\_Get\_rest().\_Get\_rest().\_Myfirst.\_Val, data.\_Myfirst.\_Val->AccountType); }

ui->tableWidget->UpdateTable(); }

void AdminWindow::AddNewGroup() {

Group\* grp = new Group(ui->GroupNameEdit->text().toStdString());

if(ui->CuratorComboBox->currentIndex() != 0) {

auto iter = allCurators.begin();

for(int i = ui->CuratorComboBox->currentIndex();i > 1;i--) {iter++;}

group->setCurator(\*iter); }

UpdateGroupsScrean(DBController::GetInstance()->grps.Add(grp)); }

void AdminWindow::DeleteGroup() {

if(group != NULL) {

DBController::GetInstance()->grps.Delete(group->\_id); }

group = NULL;

UpdateGroupsScrean(NULL); }

void AdminWindow::UpdateGroup() {

if(group != NULL) {

group->Name = ui->GroupNameEdit->text().toStdString();

if(ui->CuratorComboBox->currentIndex() != 0) {

auto iter = allCurators.begin();

for(int i = ui->CuratorComboBox->currentIndex();i > 1;i--) {iter++;}

group->setCurator(\*iter); }

else group->setCurator(NULL); }

UpdateGroupsScrean(group); }

void AdminWindow::AddClass() {

if(teacher == NULL) {

return; }

if(ui->ClassEdit->text().toStdString().empty()) {

return; }

teacher->lessonsTypes.push\_back(ui->ClassEdit->text().toStdString());

DBController::GetInstance()->infos.Update(teacher);

ui->ClassesTable->UpdateTable(); }

void AdminWindow::DeleteClass() {

if(teacher == NULL) {

return; }

if(ui->ClassEdit->text().toStdString().empty()) {

return; }

if(teacher->lessonsTypes.empty()) {

return; }

auto iterToDelete = teacher->lessonsTypes.begin();

for(auto iter = teacher->lessonsTypes.begin(); iter != teacher->lessonsTypes.end(); iter++) {

if((\*iter).compare(ui->ClassEdit->text().toStdString()) == 0) {

iterToDelete = iter; } }

teacher->lessonsTypes.erase(iterToDelete);

DBController::GetInstance()->infos.Update(teacher);

ui->ClassesTable->UpdateTable(); }

void AdminWindow::AddAccToGroup(AccountDB \*acc) {

if(group == NULL) {

return; }

for(auto stud : group->getStudents()) {

if(stud->getOwner()->\_id == acc->\_id) {

return; } }

group->addStudent(dynamic\_cast<StudentInfoDB\*>(acc->getAdditionalInfo()));

UpdateGroupsScrean(); }

void AdminWindow::DeleteAccFromGroup(AccountDB \*acc) {

if(group != NULL) {

group->deleteStudent(dynamic\_cast<StudentInfoDB\*>(acc->getAdditionalInfo())); }

UpdateGroupsScrean(); }

void AdminWindow::AddSchedule(std::string str, int RowIndex) {

if(schGroupDB == NULL || !schGroupDB->isExist) {

return; }

if(teacher == NULL || !teacher->isExist) {

return; }

auto write = new Schedule(ui->Calendar->selectedDate(), str, static\_cast<Schedule::Para>(RowIndex));

write->setGroup(schGroupDB);

write->setTeacher(teacher);

DBController::GetInstance()->sched.Add(write);

UpdateScheduleScrean(); }

void AdminWindow::DeleteSchedule(ScheduleDB \*acc) {

if(acc == NULL) {

return; }

DBController::GetInstance()->sched.Delete(acc->\_id);

UpdateScheduleScrean(); }

void AdminWindow::UpdateGroupsScrean(GroupDB\* grp) {

SetGroup(grp);

ui->GroupedAccs->UpdateTable();

ui->NotAssignedAccs->UpdateTable();

ui->GroupsTable->UpdateTable(); }

void AdminWindow::DataChanged(QDate date) {

this->date = date;

UpdateScheduleScrean(); }

void AdminWindow::UpdateScheduleScrean(AccountDB \*teacher, GroupDB \*grp) {

if(grp != NULL) {

schGroupDB = grp; }

if(teacher != NULL) {

SetAccToSchedule(teacher); }

ui->TaeachersTable->UpdateTable();

ui->GroupsTableSch->UpdateTable();

ui->ClassesTable->UpdateTable();

if(schGroupDB != NULL) {

ui->SchedulesTable->setSchedules(DBController::GetInstance()->sched.GetAllSchedulesinDate(date.toJulianDay(), schGroupDB)); } }

Файл adminwindow.h:

#ifndef ADMINWINDOW\_H

#define ADMINWINDOW\_H

#include <QMainWindow>

#include "mytable.h"

//#include "model.h"

//#include "dbcontroller.h"

namespace Ui {

class AdminWindow; }

class AdminWindow : public QMainWindow {

Q\_OBJECT

GroupDB\* group = NULL;

TeacherInfoDB\* teacher = NULL;

GroupDB\* schGroupDB = NULL;

QDate date = QDate::currentDate().addDays(1);

std::list<TeacherInfoDB\*> allCurators = std::list<TeacherInfoDB\*>();

int initialWidth = 0;

void UpdateGroupsScrean(GroupDB\* grp = NULL);

void UpdateScheduleScrean(AccountDB\* teacher = NULL, GroupDB\* grp = NULL);

public:

explicit AdminWindow(QWidget \*parent = nullptr);

void resizeEvent(QResizeEvent \*event) override;

~AdminWindow();

public slots:

void ChangeCurentSize(int tab);

void SetAccountToDisplayWidget(AccountDB\* acc);

void SetGroup(GroupDB\* acc);

void SetAccToSchedule(AccountDB\* acc);

void AddNewAccount();

void DeleteAccount();

void UpdateAccount();

void AddNewGroup();

void DeleteGroup();

void UpdateGroup();

void AddClass();

void DeleteClass();

void DataChanged(QDate date);

void SetGroupSch(GroupDB\* acc);

void AddAccToGroup(AccountDB\* acc);

void DeleteAccFromGroup(AccountDB\* acc);

void AddSchedule(std::string str, int RowIndex);

void DeleteSchedule(ScheduleDB\* acc);

private:

Ui::AdminWindow \*ui;

};

#endif // ADMINWINDOW\_H

Файл dbcontroller.cpp:

#include "dbcontroller.h"

DBController\* DBController::singleton = nullptr;

DBController::DBController() {

try {

const auto uri = mongocxx::uri{"mongodb+srv://User:User@cluster.jthkjil.mongodb.net/?retryWrites=true&w=majority"};

mongocxx::options::client client\_options;

const auto api = mongocxx::options::server\_api{mongocxx::options::server\_api::version::k\_version\_1};

client\_options.server\_api\_opts(api);

client = mongocxx::client { uri, client\_options };

DB = client["CBTBase"];

accs = AccountsController(DB["Accounts"]);

grps = GroupsController(DB["Groups"]);

infos = InfosController(DB["Infos"]);

sched = ScheduleController(DB["Schedules"]);

const auto ping\_cmd = bsoncxx::builder::basic::make\_document(bsoncxx::builder::basic::kvp("ping", 1));

DB.run\_command(ping\_cmd.view()); }

catch (const std::exception& e) {

errors::Error(e.what()); } }

DBController\* DBController::GetInstance() {

if(singleton == nullptr){

singleton = new DBController(); }

return singleton; }

void DBController::Test() {

qDebug() << bsoncxx::to\_json(make\_document(kvp("as", "as"))); }

bool DBController::AccountsController::IsAccountExist(LogPass passLog) {return Coll.find\_one(passLog.toDoc()).has\_value();}

AccountDB\* DBController::AccountsController::FindFullAccount(LogPass passLog) {

AccountDB\* ac = new AccountDB(Coll.find\_one(passLog.toDoc()).value());

Cash.push\_back(ac);

return ac; }

template<typename DBType, typename WritеType>

DBController::Controller<DBType, WritеType>::Controller() {}

template<typename DBType, typename WritеType>

DBController::Controller<DBType, WritеType>::Controller(mongocxx::collection collection) : Coll(collection) {}

DBController::AccountsController::AccountsController() : Controller<AccountDB, Account>() {}

DBController::AccountsController::AccountsController(mongocxx::collection collection) : Controller(collection) {}

DBController::InfosController::InfosController() : Controller<AdditionalInfoDB, AdditionalInfo>() {}

DBController::InfosController::InfosController(mongocxx::collection collection) : Controller(collection) {}

DBController::GroupsController::GroupsController() : Controller<GroupDB, Group>() {}

DBController::GroupsController::GroupsController(mongocxx::collection collection) : Controller(collection) {}

DBController::ScheduleController::ScheduleController() : Controller<ScheduleDB, Schedule>() {}

DBController::ScheduleController::ScheduleController(collection collection) : Controller(collection) {}

Файл dbcontroller.h:

#ifndef DBCONTROLLER\_H

#define DBCONTROLLER\_H

#include <string>

#include <cstdint>

#include <iostream>

#include <vector>

#include <bsoncxx/json.hpp>

#include <mongocxx/client.hpp>

#include <mongocxx/instance.hpp>

#include <mongocxx/stdx.hpp>

#include <mongocxx/uri.hpp>

#include <bsoncxx/builder/basic/document.hpp>

#include <mongocxx/exception/bulk\_write\_exception.hpp>

#include "errors.h"

#include "model.h"

using namespace mongocxx;

using bsoncxx::builder::basic::kvp;

using bsoncxx::builder::basic::make\_array;

using bsoncxx::builder::basic::make\_document;

class DBController {

public:

static DBController\* GetInstance();

template<typename DBType, typename WritеType>

class Controller {

protected:

std::list<DBType\*> Cash = std::list<DBType\*>();

bool IsChanged = true;

bool IsAlredyInCash(bsoncxx::oid oid, DBType \*\*res) {

for (auto iter = Cash.begin(); iter != Cash.end(); ++iter) {

if((\*iter)->isExist == false) {

(\*iter)->OwnedCount--;

if((\*iter)->OwnedCount == 0) {

Cash.remove(\*iter); }

continue; }

if((\*iter)->\_id == oid) {

\*res = \*iter;

return true; } }

return false; }

public:

collection Coll;

virtual DBType\* GetFromDB(bsoncxx::oid oid) {

DBType\* res;

if(IsAlredyInCash(oid, &res)) {

return res; }

else {

auto qeRes = this->Coll.find\_one(make\_document(kvp("\_id", oid)));

if(!qeRes.has\_value()) {

return NULL; }

DBType\* acc = new DBType(bsoncxx::document::value((qeRes.value())));

Cash.push\_back(acc);

acc->OwnedCount++;

return acc; } }

int GetCount() {

return Coll.count\_documents(make\_document()); }

virtual std::list<DBType\*>GetAll() {

if(!IsChanged) {

return std::list<DBType\*>(Cash); }

std::list<DBType\*> res = std::list<DBType\*>();

auto qres = Coll.find(make\_document());

for(auto doc : qres) {

DBType\* inst;

if(IsAlredyInCash(doc["\_id"].get\_oid().value, &inst)) {

res.push\_back(inst); }

else {

inst = new DBType(bsoncxx::document::value(doc));

res.push\_back(inst);

Cash.push\_back(inst); } }

IsChanged = false;

return res; }

virtual DBType\* Add(WritеType\* element) {

IsChanged = true;

DBType\* res;

mongocxx::stdx::optional<result::insert\_one> queryRes = Coll.insert\_one(element->toDoc());

delete element;

if(!queryRes.has\_value()) {

return NULL; }

if(queryRes.value().result().inserted\_count() != 1) {

//throw std::exception("Things getting crazy. Document writen multiple or zero times."); }

auto doc = Coll.find\_one(make\_document(kvp("\_id", queryRes.value().inserted\_id()))).value();

res = new DBType(std::move(doc));

res->OwnedCount++;

Cash.push\_back(res);

return res; }

bool Delete(bsoncxx::oid oid) {

IsChanged = true;

auto iterForDelete = Cash.begin();

for(auto i = Cash.begin(); i != Cash.end();i++) {

if((\*i)->\_id == oid) {

iterForDelete = i;

break; } }

((DBType\*)\*iterForDelete)->OwnedCount--;

if(((DBType\*)\*iterForDelete)->OwnedCount == 0) {

delete ((DBType\*)\*iterForDelete); }

else {

((DBType\*)\*iterForDelete)->isExist = false; }

Cash.remove(\*iterForDelete);

return Coll.delete\_one(make\_document(kvp("\_id", oid)))->deleted\_count() == 1; }

bool Update(DBType\* el) {

IFromDB\* temp = el;

qDebug() << bsoncxx::to\_json(temp->toDoc());

auto a = Coll.replace\_one(make\_document(kvp("\_id", temp->\_id)), temp->toDoc());

temp->doc = bsoncxx::document::value(Coll.find\_one(make\_document(kvp("\_id", temp->\_id))).value());

return a.value().matched\_count() == 1; }

Controller();

Controller(mongocxx::collection collection);

};

class AccountsController : public Controller<AccountDB, Account> {

private:

public:

AccountsController();

AccountsController(mongocxx::collection collection);

virtual std::list<AccountDB\*>GetAllTeachers() {

std::list<AccountDB\*> res = std::list<AccountDB\*>();

auto qres = Coll.find(make\_document(kvp("AccountType", 1)));

for(auto doc : qres) {

AccountDB\* inst;

if(IsAlredyInCash(doc["\_id"].get\_oid().value, &inst)) {

res.push\_back(inst); }

else {

inst = new AccountDB(bsoncxx::document::value(doc));

res.push\_back(inst);

Cash.push\_back(inst);

inst->OwnedCount++; } }

return res; }

bool IsAccountExist(LogPass passLog);

AccountDB\* FindFullAccount(LogPass passLog);

};

AccountsController accs;

class GroupsController : public Controller<GroupDB, Group> {

private:

public:

GroupsController();

GroupsController(mongocxx::collection collection);

};

GroupsController grps;

class ScheduleController : public Controller<ScheduleDB, Schedule> {

private:

public:

ScheduleController();

virtual std::list<ScheduleDB\*>GetAllSchedulesinDate(int JulianDate, GroupDB\* group) {

std::list<ScheduleDB\*> res = std::list<ScheduleDB\*>();

auto qres = Coll.find(make\_document(kvp("Date", JulianDate), kvp("Group", group->\_id)));

for(auto doc : qres) {

ScheduleDB\* inst;

if(IsAlredyInCash(doc["\_id"].get\_oid().value, &inst)) {

res.push\_back(inst); }

else {

inst = new ScheduleDB(bsoncxx::document::value(doc));

res.push\_back(inst);

Cash.push\_back(inst);

inst->OwnedCount++; } }

return res; }

ScheduleController(mongocxx::collection collection);

};

ScheduleController sched;

class InfosController : public Controller<AdditionalInfoDB, AdditionalInfo> {

private:

public:

AdditionalInfoDB\* GetFromDB(bsoncxx::oid oid, AccountType type) {

AdditionalInfoDB\* res;

if(IsAlredyInCash(oid, &res)) {

return res; }

else {

auto qeRes = this->Coll.find\_one(make\_document(kvp("\_id", oid)));

AdditionalInfoDB\* acc = NULL;

if(!qeRes.has\_value()) {

return NULL; }

if(AccountType::Admin != type) {

if(AccountType::Student == type) {

acc = new StudentInfoDB(bsoncxx::document::value((qeRes.value()))); }

else if(AccountType::Teacher == type) {

acc = new TeacherInfoDB(bsoncxx::document::value((qeRes.value()))); }

acc->OwnedCount++;

Cash.push\_back(acc); }

return acc; } }

virtual std::list<AdditionalInfoDB\*> GetAll() override {

if(!IsChanged) {

return std::list<AdditionalInfoDB\*>(Cash); }

std::list<AdditionalInfoDB\*> res = std::list<AdditionalInfoDB\*>();

auto qres = Coll.find(make\_document());

for(auto doc : qres) {

AdditionalInfoDB\* inst;

if(doc["Owner"].type() != bsoncxx::type::k\_oid) {

continue; }

if(IsAlredyInCash(doc["\_id"].get\_oid().value, &inst)) {

res.push\_back(inst); }

else {

inst = new AdditionalInfoDB(bsoncxx::document::value(doc));

res.push\_back(inst);

Cash.push\_back(inst); } }

IsChanged = false;

return res; }

AdditionalInfoDB\* Add(AdditionalInfo\* accInf, AccountType type) {

IsChanged = true;

AdditionalInfoDB\* res = NULL;

document::view temp = accInf->toDoc();

delete accInf;

auto insQeRes = Coll.insert\_one(temp);

if(!insQeRes.has\_value()) {

return res; }

auto qeRes = this->Coll.find\_one(make\_document(kvp("\_id", insQeRes.value().inserted\_id())));

if(!qeRes.has\_value()) {

//throw std::exception("Was written, but can't find. Probably due to loss of internet"); }

if(AccountType::Admin != type) {

AccountDB\* tempAcc;

if(AccountType::Student == type) {

StudentInfoDB\* temp = new StudentInfoDB(bsoncxx::document::value(qeRes.value()));

tempAcc = temp->getOwner();

if(temp->getGroup()) {

temp->getGroup()->addStudent(temp); }

res = temp; }

else if(AccountType::Teacher == type) {

TeacherInfoDB\* temp = new TeacherInfoDB(bsoncxx::document::value(qeRes.value()));

tempAcc = temp->getOwner();

res = temp; }

tempAcc->setAdditionalInfo(res);

DBController::GetInstance()->accs.Update(tempAcc);

Cash.push\_back(res);

res->OwnedCount++; }

return res; }

InfosController();

InfosController(mongocxx::collection collection);

};

InfosController infos;

void Test();

private:

DBController();

DBController( const DBController& );

DBController& operator=( DBController& );

static DBController\* singleton;

mongocxx::instance instance {};

mongocxx::client client;

mongocxx::database DB;

};

#endif // DBCONTROLLER\_H

Файл displaywidgets.cpp:

#include "displaywidgets.h"

void DisplayAccountWidget::setType(AccountType type) {

Type = type;

setLayoutAcc(); }

void DisplayAccountWidget::setLayoutAcc() {

while(layout->rowCount() > 3) {

layout->removeRow(3); }

AdditionalInfoDB\* addInf;

if(Acc == NULL) {

addInf = NULL; }

else {

addInf = Acc->getAdditionalInfo();

editLogin->setText(Acc->Login.c\_str());

editPassword->setText(Acc->Password.c\_str()); }

if(Type == AccountType::Student) {

editFirstName = new QLineEdit();

editSecondName = new QLineEdit();

editCurs = new QLineEdit();

editGroup = new QComboBox();

auto grps = DBController::GetInstance()->grps.GetAll();

editGroup->addItem("None");

for(auto grp : grps) {

editGroup->addItem(QString(grp->Name.c\_str())); }

editGroup->setCurrentText("None");

layout->addRow("First name", editFirstName);

layout->addRow("Second name", editSecondName);

layout->addRow("Curs", editCurs);

layout->addRow("Group", editGroup);

if(addInf == NULL) {

return; }

StudentInfoDB\* studInf = dynamic\_cast<StudentInfoDB\*>(addInf);

if(Type == Acc->AccountType) {

editFirstName->setText(studInf->FirstName.c\_str());

editSecondName->setText(studInf->SecondName.c\_str());

editCurs->setText(std::to\_string(studInf->Curs).c\_str());

if(studInf->getGroup() != NULL) {

editGroup->setCurrentText(studInf->getGroup()->Name.c\_str()); } } }

else if(Type == AccountType::Teacher) {

editFirstName = new QLineEdit();

editSecondName = new QLineEdit();

editFaculty = new QLineEdit();

layout->addRow("First name", editFirstName);

layout->addRow("Second name", editSecondName);

layout->addRow("Faculty", editFaculty);

if(addInf == NULL) {

return; }

TeacherInfoDB\* teachInf = dynamic\_cast<TeacherInfoDB\*>(addInf);

if(Type == Acc->AccountType) {

editFirstName->setText(teachInf->FirstName.c\_str());

editSecondName->setText(teachInf->SecondName.c\_str());

editFaculty->setText(teachInf->Faculty.c\_str()); } } }

GroupDB\* DisplayAccountWidget::getGroupFromEdit() {

auto Grps = DBController::GetInstance()->grps.GetAll();

foreach (auto grp, Grps) {

if(grp->Name == editGroup->currentText().toStdString()) {

return grp; } }

return NULL; }

void DisplayAccountWidget::OnCurrentIndexChanged(int index) {

setType((AccountType)index); }

void DisplayAccountWidget::setAccountDB(AccountDB\* acc) {

Acc = acc;

if(Acc == NULL) {

editType->setCurrentIndex(AccountType::Admin);

setType(AccountType::Admin); }

else {

editType->setCurrentIndex(Acc->AccountType);

setType(acc->AccountType); } }

std::tuple<Account\*, AdditionalInfo\*> DisplayAccountWidget::getNewAccount() {

Account\* resAcc = new Account(editLogin->text().toStdString(), editPassword->text().toStdString(), ((AccountType)editType->currentIndex()));

AdditionalInfo\* resAddInf;

StudentInfo\* studInf;

TeacherInfo\* teachInf;

switch (resAcc->AccountType) {

case AccountType::Student:

studInf = new StudentInfo(editFirstName->text().toStdString(), editSecondName->text().toStdString(), editCurs->text().toInt());

if(getGroupFromEdit() != NULL) {

studInf->setGroup(getGroupFromEdit()); }

resAddInf = studInf;

break;

case AccountType::Teacher:

teachInf = new TeacherInfo(editFirstName->text().toStdString(), editSecondName->text().toStdString(), editFaculty->text().toStdString());

resAddInf = teachInf;

break;

default:

resAddInf = NULL;

break; }

return std::tuple<Account\*, AdditionalInfo\*>(resAcc, resAddInf); }

std::tuple<AccountDB\*, AdditionalInfoDB\*, AdditionalInfo\*> DisplayAccountWidget::getAccountForDBUpdate() {

AccountDB\* resAcc = Acc;

resAcc->Login = editLogin->text().toStdString();

resAcc->Password = editPassword->text().toStdString();

resAcc->AccountType = ((AccountType)editType->currentIndex());

AdditionalInfoDB\* prevAddInf = resAcc->getAdditionalInfo();

AdditionalInfo\* newAddInf = NULL;

StudentInfo\* studInf;

TeacherInfo\* teachInf;

switch (resAcc->AccountType) {

case AccountType::Student:

studInf = new StudentInfo(editFirstName->text().toStdString(),

editSecondName->text().toStdString(),

editCurs->text().toInt());

studInf->setGroup(getGroupFromEdit());

newAddInf = studInf;

break;

case AccountType::Teacher:

teachInf = new TeacherInfo(editFirstName->text().toStdString(),

editSecondName->text().toStdString(),

editFaculty->text().toStdString());

newAddInf = teachInf;

break;

case AccountType::Admin:

break; }

return std::tuple<AccountDB\*, AdditionalInfoDB\*, AdditionalInfo\*>(resAcc, prevAddInf, newAddInf); }

AccountDB\* DisplayAccountWidget::getAccountForDelete() {

return Acc; }

DisplayAccountWidget::DisplayAccountWidget(QWidget\* parent) : QWidget(parent) {

editType->addItem("Admin");

editType->addItem("Teacher");

editType->addItem("Student");

this->connect(editType, &QComboBox::currentIndexChanged,

this, &DisplayAccountWidget::OnCurrentIndexChanged);

Acc = NULL;

layout->addRow("Login", editLogin);

layout->addRow("Password", editPassword);

layout->addRow("Type", editType); }

Файл displaywidgets.h:

#ifndef DISPLAYWIDGETS\_H

#define DISPLAYWIDGETS\_H

#include <QFormLayout>

#include <QLabel>

#include <QLineEdit>

#include <QComboBox>

#include "model.h"

#include "dbcontroller.h"

class DisplayAccountWidget : public QWidget {

QFormLayout\* layout = new QFormLayout(this);

AccountDB\* Acc;

AccountType Type;

QLineEdit\* editLogin = new QLineEdit();

QLineEdit\* editPassword = new QLineEdit();

QLineEdit\* editFirstName;

QLineEdit\* editSecondName;

QComboBox\* editType = new QComboBox();

QLineEdit\* editCurs;

QComboBox\* editGroup;

QLineEdit\* editFaculty;

void setType(AccountType type);

void setLayoutAcc();

GroupDB\* getGroupFromEdit();

public:

void setAccountDB(AccountDB\* acc);

std::tuple<Account\*, AdditionalInfo\*> getNewAccount();

std::tuple<AccountDB\*, AdditionalInfoDB\*, AdditionalInfo\*> getAccountForDBUpdate();

AccountDB\* getAccountForDelete();

DisplayAccountWidget(QWidget\* parent = 0);

public slots:

void OnCurrentIndexChanged(int index);

};

#endif // DISPLAYWIDGETS\_H

Файл errors.cpp:

#include "errors.h"

void errors::Error(std::string msg) {

const QString cmsg = QString::fromStdString(msg);

QMessageBox messageBox;

messageBox.critical(0,"Error",cmsg);

messageBox.setFixedSize(500,200); }

void errors::MSG(std::string msg) {

const QString cmsg = QString::fromStdString(msg);

QMessageBox messageBox;

messageBox.information(0,"Info",cmsg);

messageBox.setFixedSize(500,200); }

Файл errors.h:

#ifndef ERRORS\_H

#include <string>

#include <QMessageBox>

#include <Qstring>

class errors {

public:

static void Error(std::string msg);

static void MSG(std::string msg);

};

#endif // ERRORS\_H

#define ERRORS\_H

Файл main.cpp:

#include "startwindow.h"

#include "dbcontroller.h"

#include <QApplication>

int main(int argc, char \*argv[]) {

QApplication a(argc, argv);

StartWindow w;

w.show();

return a.exec(); }

Файл model.cpp:

#include "model.h"

#include "dbcontroller.h"

std::string myto\_string(enum AccountType val){

switch (val) {

case AccountType::Admin:

return std::string("Admin");

break;

case AccountType::Teacher:

return std::string("Teacher");

break;

case AccountType::Student:

return std::string("Student");

break;

default:

return std::string("");

break; } }

document::view Group::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

Group::Build(builder);

return builder->view(); }

document::view AdditionalInfo::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

AdditionalInfo::Build(builder);

return builder->view(); }

document::view StudentInfo::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

AdditionalInfo::Build(builder);

StudentInfo::Build(builder);

return builder->view(); }

document::view TeacherInfo::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

AdditionalInfo::Build(builder);

TeacherInfo::Build(builder);

return builder->view(); }

document::view GroupDB::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

IFromDB::Build(builder);

Group::Build(builder);

return builder->view(); }

document::view ScheduleDB::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

IFromDB::Build(builder);

Schedule::Build(builder);

return builder->view(); }

document::view StudentInfoDB::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

IFromDB::Build(builder);

AdditionalInfo::Build(builder);

StudentInfo::Build(builder);

return builder->view(); }

document::view TeacherInfoDB::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

IFromDB::Build(builder);

AdditionalInfo::Build(builder);

TeacherInfo::Build(builder);

return builder->view(); }

document::view Schedule::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

Schedule::Build(builder);

return builder->view(); }

document::view LogPass::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

LogPass::Build(builder);

return builder->view(); }

document::view Account::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

LogPass::Build(builder);

Account::Build(builder);

return builder->view(); }

document::view AccountDB::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

IFromDB::Build(builder);

LogPass::Build(builder);

Account::Build(builder);

return builder->view(); }

builder::stream::document \*Schedule::Build(builder::stream::document \*builder) {

\*builder << "Name" << Name

<< "Date" << Date.toJulianDay()

<< "Para" << (int)ParaNumber;

if(Teacher == NULL || !Teacher->isExist) {

\*builder << "Teacher" << NULL; }

else {

\*builder << "Teacher" << Teacher->\_id; }

if(Group == NULL || !Group->isExist) {

\*builder << "Group" << NULL; }

else {

\*builder << "Group" << Group->\_id; }

return builder; }

builder::stream::document \*AdditionalInfo::Build(builder::stream::document \*builder) {

\*builder << "FirstName" << FirstName

<< "SecondName" << SecondName;

if(Owner == NULL || !Owner->isExist) {

\*builder << "Owner" << NULL; }

else {

\*builder << "Owner" << Owner->\_id; }

return builder; }

builder::stream::document\* IFromDB::Build(builder::stream::document\* builder) {

\*builder << "\_id" << \_id;

return builder; }

builder::stream::document\* LogPass::Build(builder::stream::document\* builder) {

\*builder << "Login" << Login

<< "Password" << Password;

return builder; }

builder::stream::document\* Account::Build(builder::stream::document\* builder) {

\*builder << "AccountType" << AccountType;

if(AdditionalInfo == NULL || !AdditionalInfo->isExist) {

\*builder << "AdditionalInfo" << NULL; }

else {

\*builder << "AdditionalInfo" << AdditionalInfo->\_id; }

return builder; }

builder::stream::document\* TeacherInfo::Build(builder::stream::document\* builder) {

\*builder << "Faculty" << Faculty;

builder::stream::array arr;

for(auto lessonType : lessonsTypes) {

arr << lessonType; }

\*builder << "LessonsTypes" << arr;

return builder; }

builder::stream::document\* StudentInfo::Build(builder::stream::document\* builder) {

\*builder << "Curs" << Curs;

if(Group == NULL || !Group->isExist) {

\*builder << "Group" << NULL; }

else {

\*builder << "Group" << Group->\_id; }

return builder; }

builder::stream::document\* Group::Build(builder::stream::document\* builder) {

\*builder << "Name" << Name;

if(Curator == NULL || !Curator->isExist) {

\*builder << "Curator" << NULL; }

else {

\*builder << "Curator" << Curator->\_id; }

if(Students.empty()) {

\*builder << "Members" << NULL; }

else {

builder::stream::array arr;

for(auto iter = Students.begin(); iter != Students.end(); iter++) {

if((\*iter)->isExist) {

arr << (\*iter)->\_id; } }

\*builder << "Members" << arr; }

return builder; }

builder::stream::document \*AdditionalInfoDB::Build(builder::stream::document \*builder) {

return builder; }

document::view AdditionalInfoDB::toDoc() {

builder::stream::document\* builder = new builder::stream::document();

IFromDB::Build(builder);

AdditionalInfoDB::Build(builder);

return builder->view(); }

LogPass::LogPass(std::string login, std::string password) : Login(login), Password(password) {}

Account::Account(std::string login, std::string password, enum AccountType accountType) : LogPass(login, password) {

AccountType = accountType; }

Group::Group(std::string name) {

Name = name; }

void Account::setAdditionalInfo(AdditionalInfoDB \*adInf) {

AdditionalInfo = adInf;

if(AdditionalInfo != NULL) {

AdditionalInfo->OwnedCount++; } }

void Group::setCurator(TeacherInfoDB \*teacher) {

Curator = teacher;

if(Curator != NULL) {

Curator->OwnedCount++; } }

void Group::setStudents(std::list<StudentInfoDB \*> students) {

Students = students;

for(auto stud: students) {

stud->OwnedCount++; } }

void StudentInfo::setGroup(GroupDB \*group) {

Group = group;

if(group != NULL) {

Group->OwnedCount++; } }

void AdditionalInfo::setOwner(AccountDB \*acc) {

Owner = acc;

if(Owner != NULL) {

Owner->OwnedCount++; } }

StudentInfo::StudentInfo(std::string firstName, std::string secondName, int curs) {

FirstName = firstName;

SecondName = secondName;

Curs = curs; }

TeacherInfo::TeacherInfo(std::string firstName, std::string secondName, std::string faculty) {

FirstName = firstName;

SecondName = secondName;

Faculty = faculty; }

IFromDB::IFromDB(document::value fromDoc) :

doc(fromDoc),

\_id(fromDoc["\_id"].get\_oid().value) {}

bool GroupDB::addStudent(StudentInfoDB \*stud) {

Students.push\_back(stud);

DBController::GetInstance()->grps.Update(this);

stud->setGroup(this);

DBController::GetInstance()->infos.Update((AdditionalInfoDB\*)stud);

stud->OwnedCount++;

return true; }

bool GroupDB::deleteStudent(StudentInfoDB \*stud) {

for (auto var = Students.begin(); var != Students.end(); ++var) {

if((\*var)->\_id == stud->\_id) {

Students.erase(var);

DBController::GetInstance()->grps.Update(this);

stud->setGroup(NULL);

DBController::GetInstance()->infos.Update(stud);

stud->OwnedCount--;

return true; } }

return false; }

std::list<StudentInfoDB \*> GroupDB::getStudents() {

Students.clear();

if(doc["Members"].type() == bsoncxx::type::k\_array) {

for (auto studId : doc["Members"].get\_array().value) {

auto res = dynamic\_cast<StudentInfoDB\*>(DBController::GetInstance()->infos.GetFromDB(studId.get\_value().get\_oid().value, AccountType::Student));

if(res == NULL) {

continue; }

if(res->isExist) {

Students.push\_back(res); } } }

return Students;//todo; }

TeacherInfoDB\* GroupDB::getCurator() {

if(Curator == NULL) {

auto DB = DBController::GetInstance();

if(doc["Curator"].type() != bsoncxx::type::k\_oid) {

return NULL; }

Curator = dynamic\_cast<TeacherInfoDB\*>(DB->infos.GetFromDB(doc["Curator"].get\_oid().value, AccountType::Teacher));

if(Curator == NULL) {

return Curator; } }

if(Curator->isExist) {

return Curator; }

else {

Curator->OwnedCount--;

if(Curator->OwnedCount == 0) {

delete Curator; }

Curator = NULL; }

return Curator; }

AccountDB \*TeacherInfoDB::getOwner() {

if(Owner == NULL) {

auto DB = DBController::GetInstance();

if(doc["Owner"].type() != bsoncxx::type::k\_oid) {

return NULL; }

Owner = DB->accs.GetFromDB(doc["Owner"].get\_oid().value);

if(Owner == NULL) {

return Owner; } }

if(Owner->isExist) {

return Owner; }

else {

Owner->OwnedCount--;

if(Owner->OwnedCount == 0) {

delete Owner; }

Owner = NULL; }

return Owner; }

AccountDB \*StudentInfoDB::getOwner() {

if(Owner == NULL) {

auto DB = DBController::GetInstance();

if(doc["Owner"].type() != bsoncxx::type::k\_oid) {

return NULL; }

Owner = DB->accs.GetFromDB(doc["Owner"].get\_oid().value);

if(Owner == NULL) {

return Owner; } }

if(Owner->isExist) {

return Owner; }

else {

Owner->OwnedCount--;

if(Owner->OwnedCount == 0) {

delete Owner; }

Owner = NULL; }

return Owner; }

GroupDB \*StudentInfoDB::getGroup() {

if(Group == NULL) {

auto DB = DBController::GetInstance();

if(doc["Group"].type() != bsoncxx::type::k\_oid) {

return NULL; }

Group = DB->grps.GetFromDB(doc["Group"].get\_oid().value);

if(Group == NULL) {

return Group; } }

if(Group->isExist) {

return Group; }

else {

Group->OwnedCount--;

if(Group->OwnedCount == 0) {

delete Group; }

Group = NULL; }

return Group; }

GroupDB\* ScheduleDB::getGroup() {

if(Group == NULL) {

auto DB = DBController::GetInstance();

if(doc["Group"].type() != bsoncxx::type::k\_oid) {

return NULL; }

Group = DB->grps.GetFromDB(doc["Group"].get\_oid().value);

if(Group == NULL) {

return Group; } }

if(Group->isExist) {

return Group; }

else {

Group->OwnedCount--;

if(Group->OwnedCount == 0) {

delete Group; }

Group = NULL; }

return Group; }

AdditionalInfoDB\* AccountDB::getAdditionalInfo() {

if(this->AdditionalInfo == NULL) {

auto el = doc["AdditionalInfo"];

if(el.type() != bsoncxx::type::k\_oid) {

return NULL; }

auto DB = DBController::GetInstance();

AdditionalInfo = DB->infos.GetFromDB(el.get\_oid().value, AccountType);

if(AdditionalInfo == NULL) {

return AdditionalInfo; } }

if(AdditionalInfo->isExist) {

return AdditionalInfo; }

else {

AdditionalInfo->OwnedCount--;

if(AdditionalInfo->OwnedCount == 0) {

delete AdditionalInfo; }

AdditionalInfo = NULL; }

return AdditionalInfo; }

TeacherInfoDB \*ScheduleDB::getTeacher()

{ if(this->Teacher == NULL) {

auto el = doc["AdditionalInfo"];

if(el.type() != bsoncxx::type::k\_oid) {

return NULL; }

auto DB = DBController::GetInstance();

Teacher = dynamic\_cast<TeacherInfoDB\*>(DB->infos.GetFromDB(el.get\_oid().value, AccountType::Teacher));

if(Teacher == NULL) {

return Teacher; } }

if(Teacher->isExist) {

return Teacher; }

else {

Teacher->OwnedCount--;

if(Teacher->OwnedCount == 0) {

delete Teacher; }

Teacher = NULL; }

return Teacher; }

ScheduleDB::ScheduleDB(document::value fromDoc) :

Schedule(QDate::fromJulianDay(fromDoc["Date"].get\_int64()),

(std::string)fromDoc["Name"].get\_string(),

static\_cast<enum Para>((int)fromDoc["Para"].get\_int32())),

IFromDB(fromDoc) {}

GroupDB::GroupDB(document::value fromDoc) :

Group((std::string)fromDoc["Name"].get\_string()),

IFromDB(fromDoc) {}

AccountDB::AccountDB(document::value fromDoc) :

Account((std::string)fromDoc["Login"].get\_string(),

(std::string)fromDoc["Password"].get\_string(),

static\_cast<enum AccountType>((int)fromDoc["AccountType"].get\_int32())),

IFromDB(fromDoc) {}

TeacherInfoDB::TeacherInfoDB(document::value fromDoc) :

AdditionalInfoDB(fromDoc),

TeacherInfo((std::string)fromDoc["FirstName"].get\_string(),

(std::string)fromDoc["SecondName"].get\_string(),

(std::string)fromDoc["Faculty"].get\_string()) {

auto arr = doc["LessonsTypes"];

if(arr) {

for (auto studId : arr.get\_array().value) {

auto res = studId.get\_value().get\_string().value;

lessonsTypes.push\_back(std::string(res)); } } }

StudentInfoDB::StudentInfoDB(document::value fromDoc) :

AdditionalInfoDB(fromDoc),

StudentInfo((std::string)fromDoc["FirstName"].get\_string(),

(std::string)fromDoc["SecondName"].get\_string(),

(int)fromDoc["Curs"].get\_int32()) {}

AdditionalInfoDB::AdditionalInfoDB(document::value fromDoc) :

IFromDB(fromDoc) {}

Schedule::Schedule(QDate date, std::string name, enum Para paraNumber) : Date(date), Name(name), ParaNumber(paraNumber){}

void Schedule::setTeacher(TeacherInfoDB \*teacher) {

this->Teacher = teacher;

if(teacher != NULL) {

this->Teacher->OwnedCount++; } }

void Schedule::setGroup(GroupDB \*acc) {

this->Group = acc;

if(acc != NULL) {

this->Group->OwnedCount++; } }

Файл model.h:

#ifndef MODEL\_H

#define MODEL\_H

#include <QDateTime>

#include <list>

#include <string>

#include <cstdlib>

#include <bsoncxx/builder/stream/array.hpp>

#include <bsoncxx/builder/stream/document.hpp>

#include <bsoncxx/builder/stream/helpers.hpp>

#include <bsoncxx/types.hpp>

//#include <bsoncxx/config/prelude.hpp>

using namespace bsoncxx;

enum AccountType{

Admin,

Teacher,

Student,

};

std::string myto\_string(enum AccountType val);

struct IDocable {

public:

virtual document::view toDoc() = 0;

protected:

virtual builder::stream::document\* Build(builder::stream::document\* builder) = 0;

};

struct IFromDB : IDocable {

protected:

public:

int OwnedCount = 0;

bool isExist = true;

bsoncxx::document::value doc;

bsoncxx::oid \_id;

IFromDB(document::value fromDoc);

virtual builder::stream::document\* Build(builder::stream::document\* builder) override;

};

struct AdditionalInfoDB;

struct StudentInfoDB;

struct TeacherInfoDB;

struct AccountDB;

struct GroupDB;

struct StudentInfo;

struct TeacherInfo;

struct Account;

struct Group : public IDocable {

protected:

std::list<StudentInfoDB\*> Students = std::list<StudentInfoDB\*>();

TeacherInfoDB\* Curator = NULL;

virtual builder::stream::document\* Build(builder::stream::document\* builder) override;

public:

Group(std::string name);

std::string Name;

void setCurator(TeacherInfoDB\* teacher);

void setStudents(std::list<StudentInfoDB\*> students);

virtual document::view toDoc() override;

};

struct LogPass : public IDocable {

protected:

virtual builder::stream::document\* Build(builder::stream::document\* builder) override;

public:

LogPass(std::string login, std::string password);

std::string Login;

std::string Password;

virtual document::view toDoc() override;

};

struct Account : public LogPass {

protected:

AdditionalInfoDB\* AdditionalInfo = NULL;

virtual builder::stream::document\* Build(builder::stream::document\* builder) override;

public:

Account(std::string login, std::string password, AccountType accountType);

AccountType AccountType;

void setAdditionalInfo(AdditionalInfoDB\* adInf);

virtual document::view toDoc() override;

};

struct AdditionalInfo : public IDocable {

protected:

virtual builder::stream::document\* Build(builder::stream::document\* builder) override;

AccountDB\* Owner = NULL;

public:

std::string FirstName;

std::string SecondName;

void setOwner(AccountDB\* acc);

virtual document::view toDoc() override;

};

struct Schedule : public IDocable {

protected:

virtual builder::stream::document\* Build(builder::stream::document\* builder) override;

TeacherInfoDB\* Teacher;

GroupDB\* Group;

public:

QDate Date;

std::string Name;

enum Para{

First,

Second,

Third,

Fourth,

Fifth,

Sixth,

Seventh,

Eighth,

};

enum Para ParaNumber;

Schedule(QDate date, std::string name, enum Para paraNumber);

void setTeacher(TeacherInfoDB\* teacher);

void setGroup(GroupDB\* acc);

virtual document::view toDoc() override;

};

struct StudentInfo : public AdditionalInfo {

protected:

GroupDB\* Group = NULL;

virtual builder::stream::document\* Build(builder::stream::document\* builder) override;

public:

StudentInfo(std::string firstName, std::string secondName, int curs);

int Curs;

void setGroup(GroupDB\* acc);

virtual document::view toDoc() override;

};

struct TeacherInfo : public AdditionalInfo {

protected:

virtual builder::stream::document\* Build(builder::stream::document\* builder) override;

public:

TeacherInfo(std::string fritstName, std::string secondName, std::string faculty);

std::string Faculty;

std::list<std::string> lessonsTypes = std::list<std::string>();

virtual document::view toDoc() override;

};

struct AccountDB : public Account, public IFromDB {

public:

AccountDB(document::value fromDoc);

AdditionalInfoDB\* getAdditionalInfo();

virtual document::view toDoc() override;

};

struct ScheduleDB : public Schedule, public IFromDB {

public:

TeacherInfoDB\* getTeacher();

GroupDB\* getGroup();

ScheduleDB(document::value fromDoc);

virtual document::view toDoc() override;

};

struct GroupDB : public Group, public IFromDB {

public:

TeacherInfoDB\* getCurator();

std::list<StudentInfoDB\*> getStudents();

bool addStudent(StudentInfoDB\* stud);

bool deleteStudent(StudentInfoDB\* stud);

GroupDB(document::value fromDoc);

virtual document::view toDoc() override;

};

struct AdditionalInfoDB : public IFromDB {

protected:

virtual builder::stream::document\* Build(builder::stream::document\* builder) override;

public:

AdditionalInfoDB(document::value fromDoc);

virtual document::view toDoc() override;

};

struct TeacherInfoDB : public AdditionalInfoDB, public TeacherInfo {

public:

TeacherInfoDB(document::value fromDoc);

AccountDB\* getOwner();

virtual document::view toDoc() override;

};

struct StudentInfoDB : public AdditionalInfoDB, public StudentInfo {

public:

StudentInfoDB(document::value fromDoc);

GroupDB\* getGroup();

AccountDB\* getOwner();

virtual document::view toDoc() override;

};

#endif // MODEL\_H

Файл mytable.cpp:

#include "mytable.h"

#include <QtDebug>

std::list<AccountDB\*> AccountsTable::getAccounts() {

return DBController::GetInstance()->accs.GetAll(); }

AccountsTable::AccountsTable(QWidget \*parent) : QTableWidget(parent) {

connect(this, &AccountsTable::currentCellChanged,

this, &AccountsTable::ActiveRowChanged); }

void AccountsTable::UpdateTable() {

//DBController\* inst = DBController::GetInstance();

//inst->Test();

accs = getAccounts();

this->setRowCount((int)accs.size());

int r = 0;

for(auto acc : accs) {

if(acc == NULL) {

continue; }

PayloadedItem\* item;

AdditionalInfoDB\* info = acc->getAdditionalInfo();

if(acc->AccountType == AccountType::Student) {

StudentInfoDB\* infoStud = dynamic\_cast<StudentInfoDB\*>(info);

item = new PayloadedItem(infoStud->FirstName);

item->Payload = acc;

this->setItem(r, 0, item);

item = new PayloadedItem(infoStud->SecondName);

this->setItem(r, 1, item); }

else if(acc->AccountType == AccountType::Teacher) {

TeacherInfoDB\* infoTeach = dynamic\_cast<TeacherInfoDB\*>(info);

item = new PayloadedItem(infoTeach->FirstName);

item->Payload = acc;

this->setItem(r, 0, item);

item = new PayloadedItem(infoTeach->SecondName);

this->setItem(r, 1, item); }

else if(acc->AccountType == AccountType::Admin) {

item = new PayloadedItem("ADMIN");

item->Payload = acc;

this->setItem(r, 0, item);

item = new PayloadedItem(acc->\_id.to\_string());

this->setItem(r, 1, item); }

r++; } }

void AccountsTable::ActiveRowChanged(int currentRow, int currentColumn, int previousRow, int previousColumn) {

auto res = this->item(currentRow, 0);

//res->data(0);

if(res != NULL) {

emit AccountDBChanged((AccountDB\*)dynamic\_cast<PayloadedItem\*>(res)->Payload); } }

std::list<AccountDB\*> UngroupedAccountsTable::getAccounts() {

auto accs = DBController::GetInstance()->accs.GetAll();

std::list<AccountDB\*> res = std::list<AccountDB\*>();

for(auto acc: accs) {

StudentInfoDB\* stud;

if((stud = dynamic\_cast<StudentInfoDB\*>(acc->getAdditionalInfo())) != NULL) {

if(stud->getGroup() == NULL) {

res.push\_back(acc); } } }

return res; }

void ScheduleTable::UpdateTable() {

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

if(schedules[i] == NULL) {

PayloadedItem\* item = new PayloadedItem("");

item->Payload = NULL;

this->setItem(i, 1, item); }

else {

PayloadedItem\* item = new PayloadedItem(schedules[i]->Name);

item->Payload = schedules[i];

this->setItem(i, 1, item); } }

this->setItem(0, 0, new QTableWidgetItem("9.00 - 10.20"));

this->setItem(1, 0, new QTableWidgetItem("10.35 - 11.55"));

this->setItem(2, 0, new QTableWidgetItem("12.25 - 13.45"));

this->setItem(3, 0, new QTableWidgetItem("14.00 - 15.20"));

this->setItem(4, 0, new QTableWidgetItem("15.50 - 17.10"));

this->setItem(5, 0, new QTableWidgetItem("17.25 - 18.45"));

this->setItem(6, 0, new QTableWidgetItem("19.00 - 20.20"));

this->setItem(7, 0, new QTableWidgetItem("20.40 - 22.00")); }

std::list<AccountDB \*> TeachersTable::getAccounts() {

return DBController::GetInstance()->accs.GetAllTeachers(); }

std::list<AccountDB\*> GroupedAccountsTable::getAccounts() {

std::list<AccountDB\*> res = std::list<AccountDB\*>();

if(group == NULL) {

return res; }

if(group->isExist == false) {

return res; }

auto infos = group->getStudents();

for(auto info: infos) {

res.push\_back(info->getOwner()); }

return res; }

void GroupedAccountsTable::dropEvent(QDropEvent \*event) {

auto src = dynamic\_cast<QTableWidget\*>(event->source());

emit AddAcc((AccountDB\*)dynamic\_cast<PayloadedItem\*>(src->item(src->currentRow(), 0))->Payload); }

void UngroupedAccountsTable::dropEvent(QDropEvent \*event) {

auto src = dynamic\_cast<QTableWidget\*>(event->source());

emit DelAcc((AccountDB\*)dynamic\_cast<PayloadedItem\*>(src->item(src->currentRow(), 0))->Payload); }

void GroupedAccountsTable::setGroup(GroupDB\* grp) {

group = grp;

this->UpdateTable(); }

UngroupedAccountsTable::UngroupedAccountsTable(QWidget \*parent) : AccountsTable(parent) {

setAcceptDrops(true); }

GroupedAccountsTable::GroupedAccountsTable(QWidget \*parent) : AccountsTable(parent) {

setAcceptDrops(true); }

TeachersTable::TeachersTable(QWidget \*parent): AccountsTable(parent) {

TeachersTable::connect(this, &TeachersTable::currentCellChanged,

this, &TeachersTable::ActiveRowChanged); }

std::list<std::string> PosibleLessonsTable::getLessonsTypes() {

if(teacher == NULL) {

return std::list<std::string>(); }

else {

return teacher->lessonsTypes; } }

PosibleLessonsTable::PosibleLessonsTable(QWidget \*parent) : QTableWidget(parent) {}

ScheduleTable::ScheduleTable(QWidget \*parent) : QTableWidget(parent) {

this->setAcceptDrops(true);

this->setRowCount(8);

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

schedules[i] = NULL; } }

void ScheduleTable::dropEvent(QDropEvent \*event) {

auto row = this->rowAt(event->position().y());

auto src = dynamic\_cast<QTableWidget\*>(event->source());

emit AddSchedule(dynamic\_cast<PayloadedItem\*>(src->item(src->currentRow(), 0))->text().toStdString(), row); }

std::list<GroupDB \*> GroupsDBTable::getGroups() {

return grpController->GetAll(); }

GroupsDBTable::GroupsDBTable(QWidget \*parent) {

grpController = &DBController::GetInstance()->grps;

GroupsDBTable::connect(this, &GroupsDBTable::currentCellChanged,

this, &GroupsDBTable::ActiveRowChanged); }

void GroupsDBTable::UpdateTable() {

grps = getGroups();

this->setRowCount((int)grps.size());

int r = 0;

for(auto grp : grps) {

PayloadedItem\* item = new PayloadedItem(grp->Name);

item->Payload = grp;

this->setItem(r, 0, item);

r++; } }

void PosibleLessonsTable::setTeacher(TeacherInfoDB \*teach) {

teacher = teach;

UpdateTable(); }

void PosibleLessonsTable::UpdateTable() {

std::list<std::string> Lessons = getLessonsTypes();

this->setRowCount((int)Lessons.size());

int r = 0;

for(auto lesson : Lessons) {

PayloadedItem\* item = new PayloadedItem(lesson);

item->Payload = NULL;

this->setItem(r, 0, item);

r++; } }

void PosibleLessonsTable::dropEvent(QDropEvent \*event) {

auto src = dynamic\_cast<QTableWidget\*>(event->source());

PayloadedItem\* item = dynamic\_cast<PayloadedItem\*>(src->item(src->currentRow(), 1));

emit DeleteSchedule((ScheduleDB\*)item->Payload); }

void GroupsDBTable::ActiveRowChanged(int currentRow, int currentColumn, int previousRow, int previousColumn) {

if(currentRow >= 0) {

auto res = this->item(currentRow, 0);

emit AccountDBChanged((GroupDB\*)(dynamic\_cast<PayloadedItem\*>(res)->Payload)); } }

PayloadedItem::PayloadedItem() : QTableWidgetItem(1001) {}

PayloadedItem::PayloadedItem(std::string str) : QTableWidgetItem(str.c\_str(), 1001) {}

void ScheduleTable::setSchedules(std::list<ScheduleDB\*> scheds) {

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

schedules[i] = NULL; }

for(auto sched : scheds) {

schedules[sched->ParaNumber] = sched; }

UpdateTable(); }

Файл mytable.h:

#ifndef MYTABLE\_H

#define MYTABLE\_H

#define SchedulesArrayLenght 8

#include <QTableWidget>

#include <QDrag>

#include <QDragEnterEvent>

#include <QMimeData>

#include "dbcontroller.h"

#include "errors.h"

class ImyQTableWidget {

virtual void UpdateTable() = NULL;

};

class AccountsTable : public QTableWidget, ImyQTableWidget {

Q\_OBJECT

std::list<AccountDB\*> accs;

virtual std::list<AccountDB\*> getAccounts();

public:

AccountsTable(QWidget \*parent = NULL);

void UpdateTable();

public slots:

void ActiveRowChanged(int currentRow, int currentColumn, int previousRow, int previousColumn);

signals:

void AccountDBChanged(AccountDB\* acc);

};

class PayloadedItem : public QTableWidgetItem {

public:

void\* Payload;

PayloadedItem();

PayloadedItem(std::string str);

};

class UngroupedAccountsTable : public AccountsTable {

Q\_OBJECT

virtual std::list<AccountDB\*> getAccounts() override;

protected:

virtual void dropEvent(QDropEvent \*event) override;

public:

UngroupedAccountsTable(QWidget \*parent = 0);

signals:

void DelAcc(AccountDB\* acc);

};

class GroupedAccountsTable : public AccountsTable {

Q\_OBJECT

GroupDB\* group = NULL;

virtual std::list<AccountDB\*> getAccounts() override;

protected:

virtual void dropEvent(QDropEvent \*event) override;

public:

void setGroup(GroupDB\* grp);

GroupedAccountsTable(QWidget \*parent = 0);

signals:

void AddAcc(AccountDB\* acc);

};

class TeachersTable : public AccountsTable {

Q\_OBJECT

virtual std::list<AccountDB\*> getAccounts() override;

public:

TeachersTable(QWidget \*parent = 0);

};

class GroupsDBTable : public QTableWidget, ImyQTableWidget {

Q\_OBJECT

DBController::GroupsController\* grpController;

std::list<GroupDB\*> grps;

std::list<GroupDB\*> getGroups();

public:

GroupsDBTable(QWidget\* parent = 0);

void UpdateTable();

public slots:

void ActiveRowChanged(int currentRow, int currentColumn, int previousRow, int previousColumn);

signals:

void AccountDBChanged(GroupDB\* acc);

};

class PosibleLessonsTable : public QTableWidget, ImyQTableWidget {

Q\_OBJECT

TeacherInfoDB\* teacher = NULL;

std::list<std::string> getLessonsTypes();

public:

void UpdateTable();

virtual void dropEvent(QDropEvent \*event) override;

PosibleLessonsTable(QWidget\* parent = NULL);

void setTeacher(TeacherInfoDB\* teach);

signals:

void DeleteSchedule(ScheduleDB\* acc);

};

class ScheduleTable : public QTableWidget, ImyQTableWidget {

Q\_OBJECT

ScheduleDB\* schedules[SchedulesArrayLenght];

public:

void setSchedules(std::list<ScheduleDB\*> scheds);

void UpdateTable();

ScheduleTable(QWidget\* parent = NULL);

virtual void dropEvent(QDropEvent \*event) override;

signals:

void AddSchedule(std::string str, int RowIndex);

};

#endif // MYTABLE\_H

Файл startwindow.cpp:

#include "startwindow.h"

#include "ui\_startwindow.h"

#include "dbcontroller.h"

#include "model.h"

StartWindow::StartWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::StartWindow) {

ui->setupUi(this);

StartWindow::connect(ui->SwitchStateButton, &QAbstractButton::clicked,

this, &StartWindow::SwitchState);

StartWindow::connect(ui->ReadyButton, &QAbstractButton::clicked,

this, &StartWindow::LogIn);

this->state = StatesEnum::Login;

//std::string str = bsoncxx::to\_json(make\_document(kvp("asas", "asda"))); }

StartWindow::~StartWindow() {

delete ui; }

void StartWindow::SetState(StatesEnum state) {

this->state = state;

switch (state) {

case StatesEnum::Login:

ui->SwitchStateButton->setText(QString("or up"));

ui->ReadyButton->setText(QString("Sing in"));

break;

case StatesEnum::Signup:

ui->SwitchStateButton->setText(QString("or in"));

ui->ReadyButton->setText(QString("Sign up"));

break;

default:

break; } }

StartWindow::StatesEnum StartWindow::GetState() {

return StatesEnum::Login; }

void StartWindow::SwitchState() {

if(this->state == StatesEnum::Login) {

SetState(StatesEnum::Signup); }

else {

SetState(StatesEnum::Login); } }

void StartWindow::LogIn() {

LogPass logPass = LogPass(ui->LoginLineEdit->text().toStdString(), ui->PasswordLineEdit->text().toStdString());

DBController\* dbc = DBController::GetInstance();

if(state == StatesEnum::Login) {

if(!dbc->accs.IsAccountExist(logPass)) {

return; }

AccountDB\* fullAcc = dbc->accs.FindFullAccount(logPass);

if(fullAcc->AccountType == AccountType::Student || fullAcc->AccountType == AccountType::Teacher) {

wStud.setAccAndFill(fullAcc);

wStud.show();

this->close(); }

else if(fullAcc->AccountType == AccountType::Admin) {

wAdmin.show();

this->close(); } }

else if (state == StatesEnum::Signup) {

if(dbc->accs.IsAccountExist(logPass)) {

return; }

Account\* ac = new Account(logPass.Login, logPass.Password, AccountType::Student);

dbc->accs.Add(ac); } }

Файл startwindow.h:

#ifndef STARTWINDOW\_H

#define STARTWINDOW\_H

#include <QMainWindow>

#include <string>

#include "studentwindow.h"

#include "adminwindow.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class StartWindow; }

QT\_END\_NAMESPACE

class StartWindow : public QMainWindow {

Q\_OBJECT

public:

StartWindow(QWidget \*parent = nullptr);

~StartWindow();

enum StatesEnum{

Login,

Signup,

};

void SetState(StatesEnum state);

StatesEnum GetState();

public slots:

void SwitchState();

void LogIn();

private:

StudentWindow wStud;

AdminWindow wAdmin;

StatesEnum state;

Ui::StartWindow \*ui;

};

#endif

Файл studentwindow.cpp:

#include "studentwindow.h"

#include "ui\_studentwindow.h"

StudentWindow::StudentWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::StudentWindow) {

ui->setupUi(this);

ui->calendarWidget->setMinimumDate(QDate::currentDate());

date = QDate::currentDate();

this->connect(ui->calendarWidget, &QCalendarWidget::clicked,

this, &StudentWindow::Setdate); }

void StudentWindow::Setdate(QDate date) {

this->date = date;

UpdateWindow(); }

void StudentWindow::UpdateWindow() {

ui->Schedule->setSchedules(DBController::GetInstance()->sched.GetAllSchedulesinDate(date.toJulianDay(), dynamic\_cast<StudentInfoDB\*>(acc->getAdditionalInfo())->getGroup())); }

StudentWindow::~StudentWindow() {

delete ui; }

void StudentWindow::setAccAndFill(AccountDB\* Acc) {

acc = Acc;

UpdateWindow();

ui->labelPasswordVal->setText(acc->Password.c\_str());

ui->labelLoginVal->setText(acc->Login.c\_str());

ui->labelTypeVal->setText(myto\_string(acc->AccountType).c\_str());

auto addInf = dynamic\_cast<StudentInfoDB\*>(Acc->getAdditionalInfo());

GroupDB\* grp = NULL;

if(addInf != NULL) {

ui->labelFirstName->setText(addInf->FirstName.c\_str());

ui->labelSeconDame->setText(addInf->SecondName.c\_str());

ui->labelCurs->setText(std::to\_string(addInf->Curs).c\_str());

grp = addInf->getGroup(); }

else {

ui->labelFirstName->setText("");

ui->labelSeconDame->setText("");

ui->labelCurs->setText(""); }

if(grp != NULL) {

ui->labelGroup->setText(grp->Name.c\_str()); }

else {

ui->labelGroup->setText(""); } }

Файл studentwindow.h:

#ifndef STUDENTWINDOW\_H

#define STUDENTWINDOW\_H

#include "model.h"

#include <QMainWindow>

namespace Ui {

class StudentWindow; }

class StudentWindow : public QMainWindow {

Q\_OBJECT

AccountDB\* acc;

Ui::StudentWindow \*ui;

QDate date;

public:

explicit StudentWindow(QWidget \*parent = nullptr);

void Setdate(QDate date);

void UpdateWindow();

~StudentWindow();

void setAccAndFill(AccountDB\* Acc);

};

#endif // STUDENTWINDOW\_H

Файл teacherwindow.cpp:

#include "teacherwindow.h"

#include "ui\_teacherwindow.h"

TeacherWindow::TeacherWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::TeacherWindow) {

ui->setupUi(this); }

TeacherWindow::~TeacherWindow() {

delete ui; }

Файл teacherwindow.h:

#ifndef TEACHERWINDOW\_H

#define TEACHERWINDOW\_H

#include <QMainWindow>

namespace Ui {

class TeacherWindow; }

class TeacherWindow : public QMainWindow {

Q\_OBJECT

public:

explicit TeacherWindow(QWidget \*parent = nullptr);

~TeacherWindow();

private:

Ui::TeacherWindow \*ui;

};

#endif // TEACHERWINDOW\_H

Файл tst\_dbcontrollertests.cpp:

#include <gtest/gtest.h>

using namespace testing;

TEST(lazyTests, dbControllerTests) {

EXPECT\_EQ(1, 1);

EXPECT\_EQ(1, 1); }

Файл tst\_dbctests.cpp:

#include <gtest/gtest.h>

#include <gmock/gmock-matchers.h>

using namespace testing;

TEST(CBTests, DBCTests) {

EXPECT\_EQ(1, 1);

ASSERT\_THAT(0, Eq(0)); }