# **4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

Листинг программы представлен в Приложении А.

## **Создание базы данных PostgreSQL**

Первоначально необходимо создать новую базу данных для разработанного приложения. Ее можно создать, используя стандартные средства, предоставляемые Postgres, такими как pgAdmin (версии 4). Это стандартная утилита, предоставляющая графический интерфейс для упрощенного управления базами данных. Для создания новой базы данных необходимо включить утилиту.

Если запуск pgAdmin будет первым, то приложение выведет на экран окно «Set Master Password», в котором мы должны задать «мастер-пароль». Это нужно для дополнительного шифрования паролей. После установки мастер-пароля все существующие сохраненные пароли будут повторно зашифрованы с использованием мастер-пароля.

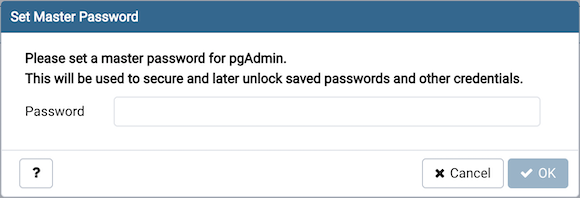


Рисунок 4.1 - Окно «Set Master Password» в pgAdmin 4

Если же вход в приложение не первый, то оно предложит ввести «мастер-пароль», выведя на экран окно «Unlock Saved Passwords».

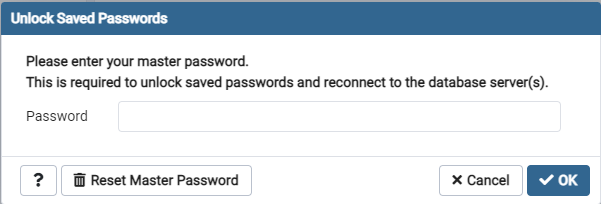


Рисунок 4.2 - Окно «Unlock Saved Passwords» в pgAdmin 4

После авторизации в приложении можно создавать новую базу данных. Для этого необходимо выполнить действия, пошагово продемонстрированные на рисунке 4.3:

Шаг 1. Правой кнопкой мыши нажать около слова «Databases» (1).

Шаг 2. В появившемся меню навести на кнопку «Create» (2).

Шаг 3. В новом меню выбрать пункт «Database…» (3).

Шаг 4. Далее, в новом окне (4), необходимо указать имя базы данных, которое будет использоваться в дальнейшем (5).

Шаг 5. Последний же шаг, подтверждение созхдания новой базы данных.

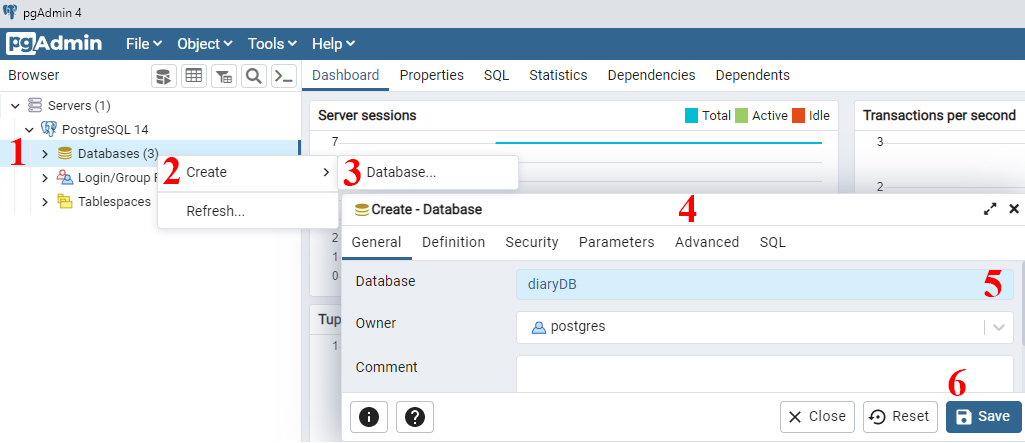


Рисунок 4.3 - Создание новой базы данных посредством pgAdmin 4

## **Создание пользователей базы данных PostgreSQL**

Следующий шаг в разработке приложения – создание пользователей с разными правами. Для этого был написан скрипт на языке SQL с использованием приложения DataGrip. Данная среда разработки необходима для написания SQL-запросов и работы с базами данных.

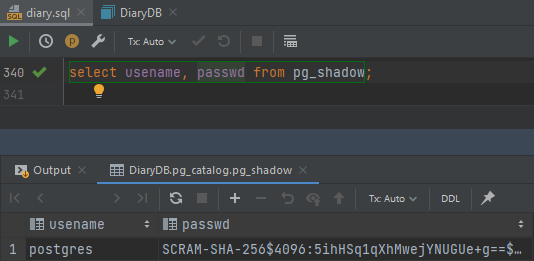


Рисунок 4.4 - Первоначальные пользователи базы данных PostgreSQL

Для просмотра существующих ролей и пользователей можно использовать запрос к таблице «pg\_shadow», которая содержит информацию об имени роли и пароле пользователя, действительности пароля пользователя, лимите подключения пользователя и роли, автоматически наследующей привилегии ролей, членом которых он является, подробную информацию о пользователях и управлении привилегиями. Для демонстрации создаваемых автоматически пользователей используем запрос, который отобразит имя пользователя и пароль для него:

select usename, passwd from pg\_shadow;

Для разделения ролей будет создана новая роль auth\_user. Ее суть – предоставления права для чтения данных из таблиц, без возможности их модернизации. Пароль задается сугубо для примера «pgsecret», что не является надежным, однако хорошим для демонстрации написания запроса.

CREATE USER auth\_user WITH ENCRYPTED PASSWORD 'pgsecret';

GRANT USAGE ON SCHEMA public to auth\_user;

ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO auth\_user;

GRANT CONNECT ON DATABASE "DiaryDB" to auth\_user;

ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL ON TABLES TO auth\_user;

GRANT USAGE ON SCHEMA public to auth\_user;

GRANT SELECT ON ALL SEQUENCES IN SCHEMA public TO auth\_user;

GRANT SELECT ON ALL TABLES IN SCHEMA public TO auth\_user;

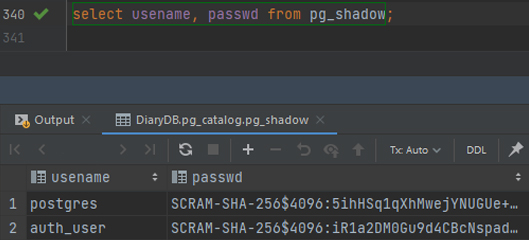


Рисунок 4.5 - Все пользователи базы данных PostgreSQL

После выполнения данного запроса можно проверить его выполнение уже описанным ранее запросом (рисунок 4.5).

## **Создание резервных копий базы данных PostgreSQL**

При написании данного дипломного проекта рассматривается автономное создание резервной копии базы данных. Для этого необходимо написать скрипт с расширением «.cmd». Шаги по написанию данного скрипта:

Шаг 1. Установим кодовую страницу, соответствующую Windows кодировке.

Шаг 2. Создадим путь до корневой папки PostgreSQL.

Шаг 3. Укажем адрес сервера.

Шаг 4. Укажем порт сервера.

Шаг 5. Установим имя пользователя, имеющего права на данную базу данных.

Шаг 6. Укажем пароль пользователя, указанного на шагу 5.

Шаг 7. Создадим переменную, указывающую на текущее время.

Шаг 8. Укажем имя базы данных, для которой создается бэкап.

Шаг 9. Выполнение бэкапа с указанием конечного места копирования.

Шаг 10. Также для уменьшения объема памяти, занимаемого приложением, удаляются старые копии, старше указанного срока (на примере указано 29 дней).

CHCP 1251

SET PGBIN=C:\Program Files\PostgreSQL\14\bin

SET PGHOST=localhost

SET PGPORT=3306

SET PGUSER= DBadmin

SET PGPASSWORD=diary

SET DATETIME=%DATE:~6,4%-%DATE:~3,2%-%DATE:~0,2%

SET PGDATABASE=diaryDB

CALL "%PGBIN%\pg\_dump.exe" --format=custom --verbose --file=C:\14\back\%PGDATABASE%\_%DATETIME%.dump

forfiles /p C:\14\ /m \*.dump /s /d -29 /c "cmd /c del @path /q"

Следующим будет добавление данного скрипта в планировщик заданий. Сделать это не является проблемным. Необходимо зайти в «планировщик заданий», нажать правой кнопкой мыши в районе существующих задач и выбрать пункт «создание простой задачи». Далее необходимо указать название задачи, частоту запуска и путь к созданному ранее скрипту. После чего активировать данный скрипт. В результате мы получим автономно выполняющиеся задачи по созданию резервной копии, созданной ранее базы данных, необходимой для повышенной безопасности, в случае потери данных, или «неудачи» по внесению обновлению базы данных, что может привести к утрате актуальных данных.

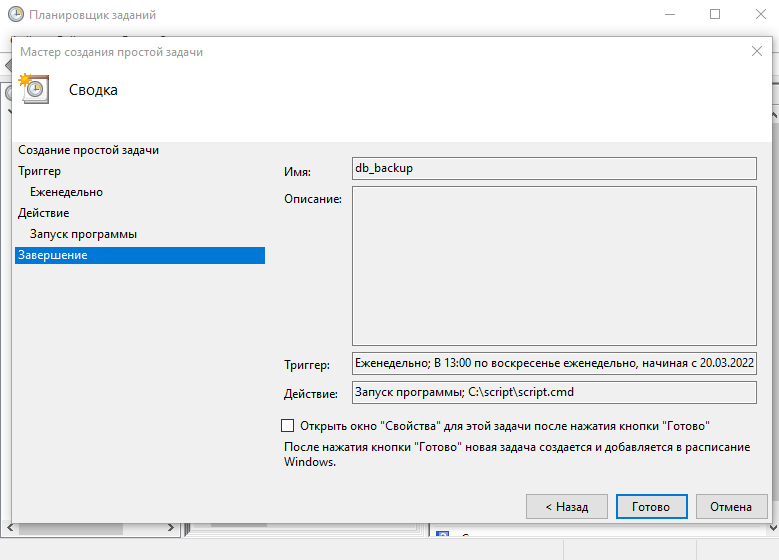


Рисунок 4.6 - Создание задачи по резервному копированию базы данных

## **Создание таблиц и связей базы данных PostgreSQL с использованием**

Созданная заранее ER-модель преобразуется в реляционную модель согласно шагам, описываемым в методическом пособии к лабораторным работам [13]. В результате выполнения всех шагов, можно получить файл, с расширением «.dll». Выполнив преобразования (не все типы данных, существующие в приложении Oracle SQL Developer Data Modeler, поддерживаются другими приложениями, например, Data Grip) получаем следующий скрипт, по созданию базы данных. Ниже представлены несколько таблиц и связь между ними, чтобы не загромождать описываемый раздел. Полный скрипт будет описан в приложении.

CREATE TABLE ids (

student\_id INTEGER NOT NULL,

teacher\_id INTEGER NOT NULL

);

ALTER TABLE ids ADD CONSTRAINT ids\_pk PRIMARY KEY ( student\_id, id );

CREATE TABLE teacher (

full\_name\_t VARCHAR,

gender VARCHAR,

id INTEGER NOT NULL,

rank VARCHAR,

phone\_number VARCHAR,

subject\_id INTEGER NOT NULL,

birth\_date\_t DATE

);

ALTER TABLE teacher ADD CONSTRAINT teacher\_pk PRIMARY KEY ( id );

CREATE TABLE student (

full\_name\_st VARCHAR,

gender VARCHAR,

id INTEGER NOT NULL,

studying\_group\_id INTEGER NOT NULL,

birth\_date\_st DATE,

stud\_parent\_id INTEGER NOT NULL

);

ALTER TABLE student ADD CONSTRAINT student\_pk PRIMARY KEY ( id );

ALTER TABLE ids

ADD CONSTRAINT ids\_student\_fk FOREIGN KEY ( student\_id )

REFERENCES student ( id );

ALTER TABLE ids

ADD CONSTRAINT ids\_teacher\_fk FOREIGN KEY ( teacher\_id )

REFERENCES teacher ( id );

С помощью запроса, описанного ниже, можно увидеть, что таблицы действительно создались:

select \* from information\_schema.tables where table\_schema='public';

Результат выполнения запроса предоставлен на рисунке 4.7.

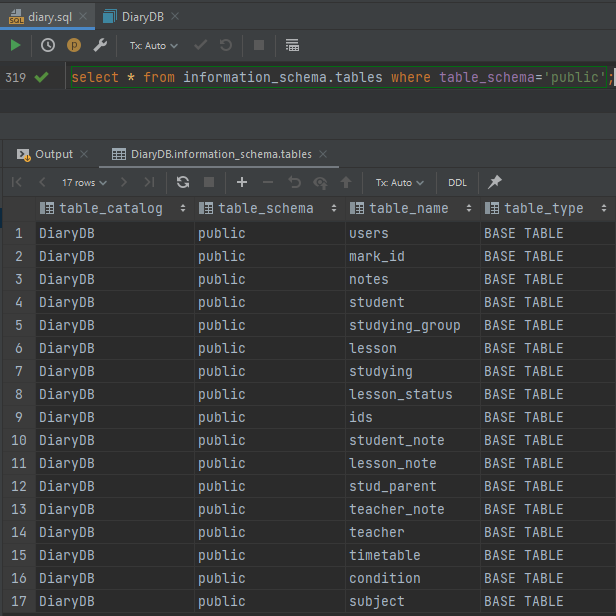


Рисунок 4.7 - Результат проверки создания таблиц в базе данных

## **Регистрация приложения** **в API Яндекс ID**

Регистрация нового приложения является простой операцией, не требующей особых усилий. Необходимо перейти на сайт данного API, расположенного по адресу <https://oauth.yandex.ru/>. После чего сайт потребует пользователя авторизоваться. Стоит отметить, что аккаунт, с которого будет произведена авторизация в дальнейшем будет считаться «администратором». Поэтому заранее рекомендуется защитить авторизуемый аккаунт сильным паролем. После авторизации необходимо нажать на кнопку «зарегистрировать новое приложения». Следующий экран предложит «добавить» приложение, указав его название, описание, логотип, ссылку на сайт. При выборе веб-сервисов указывается его callback URI (адрес, на который пользователь возвращается после того, как он разрешил или отказал приложению в доступе (соответствует redirect\_uri протокола OAuth)). Также можно добавить тот факт, что «Яндекс» предоставляет собственный адрес «<https://oauth.yandex.ru/verification_code>», что значительно упрощает разработку.

«Яндекс» бесплатно предоставляет доступ ко множеству своих сервисов (например, «Яндекс.Дисплей», что позволяет размещать рекламные баннеры в приложении, «Яндекс ID», используемая в данном дипломном проекте для доступа к логину, имени и фамилии, полу пользователя, а также доступа к адресу электронной почты), что позволяет рассчитывать на масштабирование в дальнейшем с использованием данного сервиса.

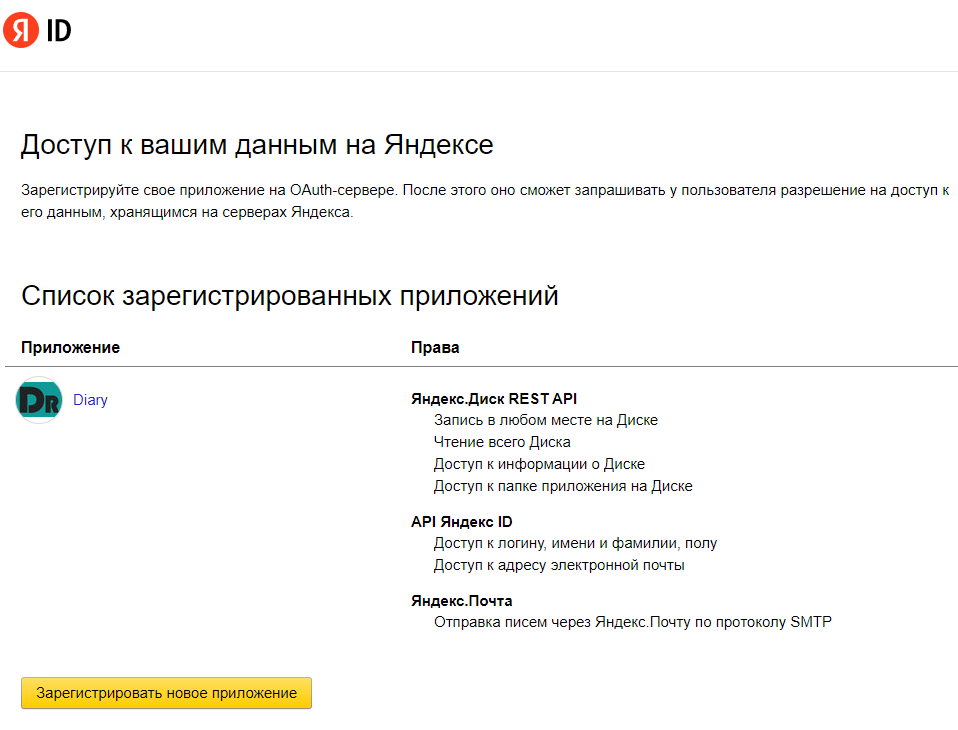


Рисунок 4.8 – Зарегистрированные приложения в API Яндекс ID

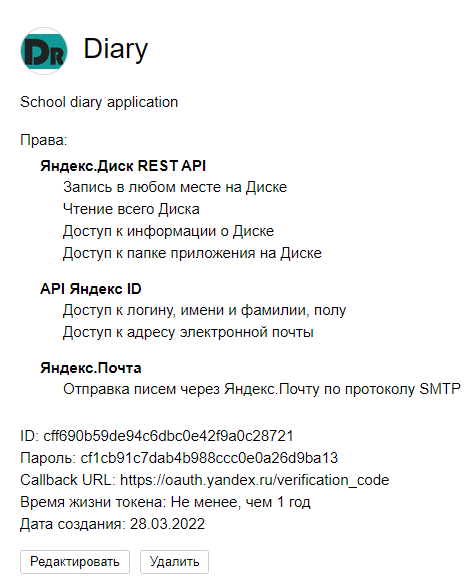


Рисунок 4.9 – Приложение в API Яндекс ID

После успешного создания приложения пользователь увидит нечто похожее на то, что изображено на рисунке 4.9. Самый интересующие здесь момент – это ID (идентификационный номер приложения).

Позже, данный идентификационный номер будет использоваться при авторизации пользователей, при составлении ссылки, по которой производится перенаправление:

https://oauth.yandex.ru/authorize?response\_type=token&client\_id=cff690b59de94c6dbc0e42f9a0c28721

Как можно заметить, данный ID указывается после знака «=».

## **Заголовочный файл connection.h**

В приложении предусмотрено подключение к базе данных PostgreSQL. Для осуществления данного действия создан метод createConnection(QString role, QString password), который осуществляет подключение к базе данных. Передаваемые данному методу параметры представляют собой имя пользователя базы данных (role) и пароль для данного пользователя (password).

Для подключения к базе данных используется метод addDatabase, класса QSqlqDatabase, являющегося дополнительным модулем, который необходимо до устанавливать в фреймворк QTCreator, путем добавления DLL (Dynamic Link Library — «библиотека динамической компоновки») файлов в исходную папку QT. При наличии всех файлов остается лишь подключиться к базе данных, использую имя пользователя и пароль. Также стоит отметить, что если соединение с базой данных уже существует и осуществляется попытка нового подключения к ней, то прежнее соединение будет разорвано. Это будет использоваться в дальнейшем, при изменении роли пользователи после авторизации.

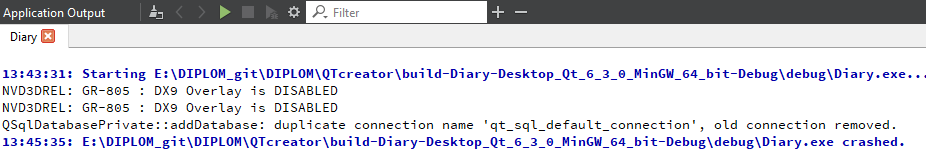


Рисунок 4.10 – Пример логирования приложения при изменении роли пользователя базы данных

На рисунке 4.10 как раз-таки и предоставлено сообщение об прерывании «старого» соединения к базе данных (предпоследнее). Сообщения «NVD3DREL: GR-805 : DX9 Overlay is DISABLED» говорят об ошибке драйвера Nvidia в связи с более актуальной версией, что не влияет на работоспособность приложения.

SqlDatabase db = QSqlDatabase::addDatabase("QPSQL");

db.setDatabaseName("DiaryDB");

db.setUserName(role);

db.setPassword(password);

if (!db.open()){

QMessageBox::warning(0,"Ошибка БД", "Something with DB Wrong");

return false;

}

else{

return true;

}

}

## **Класс mainwindow**

В данном классе интересным будет лишь обращение к методу createConnection("auth\_user", "pgsecret"), описанному ранее, где auth\_user это имя пользователя с правами только на просмотр содержимого базы данных, а pgsecret пароль для данного пользователя. В случае ошибки подключения к базе данных в логах приложения появится ошибка с кодом 2, что соответствует фразе «При соединении с сервером произошла ошибка» из официальной документации Microsoft [14].

## **Класс authorization**

Данный класс, как говорилось ранее, отвечает за авторизацию пользователей. Основная логика возлагается на метода on\_accept\_button\_clicked(). Тут из строки ввода приложение получает токен приложения в API Яндекс ID. Далее, для более безопасного хранения данных, данный токен шифруется. В моем случае производится преобразования введенного значения к формату UTF-8, после чего с помощью метода QCryptographicHash() преобразованное ранее значение шифруется по алгоритму SHA-256. Данный алгоритм считается безопасным алгоритмом хеширования, который генерирует выходное хеш-значение в 256 бит. SHA-256 является одной из криптографических хеш-функций, а также называется односторонней функцией, в которой сгенерированное хеш-значение не может быть обращено назад (теоретически).

setToken(ui -> token\_lineEdit -> text());

setToken\_toUtf8(getToken().toUtf8());

setToken(QCryptographicHash::hash(getToken\_toUtf8(), QCryptographicHash::Sha256).toHex());

Можно отметить и тот факт, что данные в базе данных PostgreSQL также шифруются по алгоритму SHA-265. Таким образом мы дважды шифруем введенное значение, что гарантирует невозможность расшифровки данных, при несанкционированном доступе к базе данных.

В дальнейшем алгоритм взаимодействия с приложением основывается на запросах к базе данных. Первоначально осуществляется select-запрос к таблице users с целью соотнесения введенного токена и роли пользователя, его идентификационного номера.

question\_to\_db = "select user\_id, role\_id from users where token = '" + getToken() +"'";

query.exec(question\_to\_db);

if (!query.next()) {

QMessageBox::warning(this, "User name", "This user is not presented");

}

else {

close();

user\_id = query.value(0).toString();

role\_num = query.value(1).toInt();

}

В случае отсутствия пользователя в базе данных, то на экране будет отображено сообщение с соответствующим сообщением. Если же пользователь будет найден, то в поля user\_id и role\_num будут записаны номер роли и идентификационный номер пользователя.

Далее в зависимости от номера роли определяется дальнейшие действия: либо пользователь будет перенаправлен на другое окно, без изменения своих прав доступа к базе данных, либо, в случае с номером роли «1», пользователю будут изменены права на роль «teacher» с паролем «teacherDBpassword».

if(!createConnection("teacher","teacherDBpassword"))

{

exit(2);

}

Вне зависимости от роли пользователя перед выбором дальнейшего окна необходимо получить полные данные об авторизованном пользователе: фамилия и имя. Для этого осуществляются схожие между собой запросы к базе данных с отличием в названии переменных разных таблиц.

question\_to\_db = "select full\_name\_parent from stud\_parent where id = '" + user\_id + "'";

query.exec(question\_to\_db);

while (query.next()) {

full\_name = query.value(0).toString();

}

Параметр full\_name будет передан следующему окну в качестве параметра.

## **Класс diary\_menu**

Основная логика данного класса предназначена для осуществления интуитивной понятности интерфейса. Он перенаправляет на другие окна, представленные другими классами, однако также можно выделить момент вызова метода on\_contact\_info\_button\_clicked(), который предназначен для вывода контактной информации о разработчике.

close();

QMessageBox m;

m.setText("<a href=\"https://www.linkedin.com/in/albasandros\">To get support or contact with creator "

"you may press this hyperlink</a>");

QTimer::singleShot(10000, &m, SLOT(close()));

m.exec();

close();

В данном методе вызывается таймер на 10000 миллисекунд. При его вызове используется логика слотов и сигналов. Также прописана логика фиксации размера диалогового окна для предотвращения ошибок внешнего вида приложения, при изменении разрешения экрана.

setWindowFlags(Qt::Dialog);

setFixedSize(545, 445);

## **Класс add\_mark**

Вся логика данного класса реализуется посредством метода on\_next\_pushButton\_clicked(). Основная задача тут – проверка наличия искомого «учащегося» для выставления последнему отметки. Если «учащийся» в базе данных не представлен, что выводится соответствующее сообщение на экран. Проверка наличия осуществляется посредством select-запроса к базе данных PostgreSQL, с целью поиска идентификационного номера (id). Если «учащийся» будет найден в базе данных, то его полное имя будет передано в следующее окно с помощью метода getStudent\_fullname().

setStudent\_fullname(ui->student\_fullname->text());

question\_to\_db = "select id from student where full\_name\_st = '" + getStudent\_fullname() + "'";

query.exec(question\_to\_db);

if (!query.next()) {

QMessageBox::warning(this, "Student Full Name", "This student is not presented in DataBases");

}

else{

close();

add\_mark\_2 mark2\_window(this, getLogin(), getStudent\_fullname());

mark2\_window.setModal(true);

mark2\_window.exec();

}

## **Класс add\_mark2**

Этот класс является основным при выставлении отметки «учащемуся». В начале происходит заполнение полей уже имеющейся информацией автоматически. Выводится имя преподавателя в верхнем углу приложения, имя «учащегося», которому выставляется отметка. Эти данные уже известны, так как они передавались параметрами данному классу.

ui->label->setText("Welcome back, " + getLogin());

setStudent\_name(student\_name);

ui->lineEdit\_full\_name->setText(student\_name);

Следующий шаг – получение названия учебного предмета, по которому выставляется отметка. Это производится посредством select-запроса к базе данных с логикой, что преподаватель, выставляющий отметку, может ставить отметки только по своему предмету. Для этого ищется идентификационный номер учебного предмета, соответствующий преподавателю. После чего устанавливается соответствие идентификационного номера полному названию предмета. Далее данное название вводится автоматически в соответствующее поле на экране.

question\_to\_db = "select subject\_id from teacher where full\_name\_t = '" + getLogin() + "'";

query.exec(question\_to\_db);

if (!query.next()){

QMessageBox::warning(this, "Subject Error", "You not teacher for this subject");

}else{

sub\_id = query.value(0).toString();

}

question\_to\_db = "select sub\_name from subject where id = '" + sub\_id + "'";

query.exec(question\_to\_db);

if (!query.next()){

QMessageBox::warning(this, "Subject Error", "Please, choose one of presented subjects");

}else{

subject = query.value(0).toString();

}

ui->lineEdit\_subjectName->setText(subject);

Далее «преподаватель» вводит отметки и комментарии в соответствующих полях диалогового экрана. Вводимую отметку необходимо перевести к значению text() для возможности осуществления запросов к базе данных в дальнейшем.

mark = ui -> lineEdit\_mark -> text();

Следующий шаг – проверка: изучает ли «учащийся» предмет, по которому ему выставляется отметка. Если данный предмет не изучается учащимся, то выводится соответствующее сообщение на экран.

question\_to\_db = "select \* from studying where subject\_id = '" + sub\_id + "' and student\_id = '" + student\_id + "';";

query.exec(question\_to\_db);

query.next();

QString student\_id\_check = query.value(0).toString();

if (student\_id\_check != student\_id){

QMessageBox::warning(this, "Subject error", "This student not studying this subject");

}

В противном случае продолжается выполнение программы. Происходит поиск значения последнего id предмета, для получения номера данного урока.

question\_to\_db = "select id from lesson\_status ORDER BY id DESC LIMIT 1";

query.exec(question\_to\_db);

query.next();

После вводятся значения отметки, идентификационного номера урока, студента, предмета.

question\_to\_db = "INSERT INTO lesson\_status VALUES ('" + mark + "', '" + lessonID + "', '" + lessonStatus + "')";

query.exec(question\_to\_db);

question\_to\_db = "INSERT INTO mark\_id VALUES ('" + student\_id + "', '" + lessonID + "')";

query.exec(question\_to\_db);

question\_to\_db = "INSERT INTO condition VALUES ('" + lessonID + "', '" + sub\_id + "')";

query.exec(question\_to\_db);

Если пользователь ввел комментарии, то производится запрос в базу данных для поиска последнего номера заметок. Этот номер переводится из формата string в формат int, изменяется на «+1», для получения номера новой заметки и трансформируется обратно к string, для облегчения запросов к базе данных. В конечном итоге данные значения вводятся с помощью insert-запросов к базе данных.

question\_to\_db = "select id from notes ORDER BY id DESC LIMIT 1";

query.exec(question\_to\_db);

query.next();

commentID = query.value(0).toString();

commentID\_int = commentID.toInt();

commentID\_int= commentID\_int + 1;

commentID = QString::number(commentID\_int);

question\_to\_db = "INSERT INTO notes VALUES ('" + comment + "', '" + commentID + "')";

query.exec(question\_to\_db);

question\_to\_db = "INSERT INTO student\_note VALUES ('" + student\_id + "', '" + commentID + "')";

query.exec(question\_to\_db);

## **Класс check\_class**

Как описывалось ранее данный класс используется для поиска информации о группе «учащихся», а именно классе целиком. Для этого пользователю необходимо ввести полноценный идентификатор класса (например, 10 «А») в специально отведенные для этого строки на диалоговом окне. Логика класса реализована в методе on\_find\_button\_clicked(). Для предупреждения проблем в будущем запросе к базе данных: в регистрах ввода буквенного обозначения класса (для предыдущего примера – «А» и «а») предусмотрен перевод данного значения к «верхнему» регистру.

setClass\_letter(ui->class\_letter\_label->text().toUpper());

Следующим шагом является проверка ввода. Предполагается, что если пользователь не ввел никаких данных и нажал кнопку подтверждения, то он хочет получить всю имеющуюся в базе данных информацию.

if(getClass\_letter() == "" && getClass\_num() == 0){

close();

class\_info class\_window(this, getLogin(), getClass\_letter(), getClass\_num());

class\_window.setModal(true);

class\_window.exec();

}

Если же данные были введены, то первоначально производится проверка введенных данных. Пользователь обязан ввести численное и буквенное обозначение номера класс, а также численное значения номера класса обязано быть в пределах от «1» до «11».

if(getClass\_letter() == ""){

QMessageBox::warning(this, "Checking class info", "You need to input class letter");

}

else if(getClass\_num() == 0){

QMessageBox::warning(this, "Checking class info", "You need to input class number");

}

else if (getClass\_num() < 0 || getClass\_num() > 11){

QMessageBox::warning(this, "Checking class info", "Check class number. It must be in range 0..11");

}

Если проверки проведены успешно, то осуществляется запрос к базе данных с целью получения идентификационного номера (id) учебного класса. В случае не обнаружения данного номера пользователю выводится на экран соответствующее сообщение. Если id был найден, то в следующий класс передается полное имя авторизованного пользователя, номер и буква искомого класса.

question\_to\_db = "select id from studying\_group where num = '" + QString::number(getClass\_num()) + "' and profile = '" + getClass\_letter() +"'";

query.exec(question\_to\_db);

if (!query.next()){

QMessageBox::warning(this, "Checking class info", "This class is not presented in DataBase");

}else{

if (( (getClass\_letter() != "" && getClass\_num() > 0 && getClass\_num() < 12)) ){

close();

class\_info class\_window(this, getLogin(), getClass\_letter(), getClass\_num());

class\_window.setModal(true);

class\_window.exec();

}

## **Класс class\_info**

Данный класс является логическим завершением над поиском и отображением информации об «учащихся». Первым делом формируется сообщение, выводимое в специальное окно в зависимости от круга, среди которого выделяли «учащихся». Если искалась информация обо всех «классах», то выводится сообщение вида «Информация обо всех учащихся». Если информация уточнялась по конкретному классу, то выведенное сообщение, на примере 10 «А» класса, будет: «Вся информация про 10 «А» класс».

if (getClass\_num() == 0 && getClass\_letter() == ""){

infoText = "Info about all students";

}else{

infoText = "All info about " + QString::number(getClass\_num()) + " '" + getClass\_letter() + "'" + "class";

}

ui->textEdit->append(infoText);

Затем, в зависимости от круга поиска информации, составляется запрос к базе данных. Если требуется информация обо всех учащихся, то запрос принимает следующий вид:

if (getClass\_num() == 0 && getClass\_letter() == ""){

question\_to\_db="SELECT \* FROM student";

}

Если же информация уточняется по конкретному учебному классу, то для начала ищется идентификационный номер учебного класса, а после чего выделяются «учащиеся», обучающиеся в данной группе.

else{

get\_gr\_id\_request = "SELECT id from studying\_group WHERE num = '" + QString::number(getClass\_num()) + "' and profile = '" + getClass\_letter() +"'";

query\_getid.exec(get\_gr\_id\_request);

while (query\_getid.next()) {

gr\_id = query\_getid.value(0).toString();

}

question\_to\_db="SELECT \* FROM student WHERE studying\_group\_id = '" + gr\_id + "'";

}

Следом выполняются описанные выше запросы. Первыми извлекаются все данные об учащемся.

query.exec(question\_to\_db);

while (query.next()) {

full\_name = query.value(0).toString();

studying\_group\_id = query.value(3).toString();

birth\_date = query.value(4).toString();

stud\_parent\_id = query.value(5).toString();

Далее внутри данного запроса выполняется еще один запрос, для получения информации об родителях учащегося. Основной интересующей здесь информацией на данный момент будут полное имя родителя и номер телефона.

question\_to\_db\_par = "select full\_name\_parent, phone\_numb\_parent from stud\_parent where id = '" + stud\_parent\_id + "'";

query\_parents.exec(question\_to\_db\_par);

while (query\_parents.next()) {

full\_name\_parent=query\_parents.value(0).toString();

phone\_numb\_parent=query\_parents.value(1).toString();

}

Затем, в зависимости от объема искомой информации, создается еще один подзапрос, в случае если искалась информация обо всех «учащихся», для уточнения номера учебного класса. В последствие формируется особое сообщение с искомой информацией. Если информация ищется по конкретному классу, то подзапрос не создается, а сразу выводится вся информация на экран с иной формулировкой сообщения.

if(getClass\_num() == 0 && getClass\_letter() == ""){

question\_to\_db\_class = "select num, profile from studying\_group where id = '" + studying\_group\_id + "'";

query\_class\_num.exec(question\_to\_db\_class);

while (query\_class\_num.next()) {

gr\_num = query\_class\_num.value(0).toString();

gr\_prof= query\_class\_num.value(1).toString();

}

infoText = "\n=================================\n"

"\nFull name: " + full\_name +

"\nBirth data: " + birth\_date +

"\nGroup number: " + gr\_num + " '" + gr\_prof + "' " +

"\nParent full name: " + full\_name\_parent +

"\nParent phone number: " + phone\_numb\_parent;

}else{

infoText = "\n=================================\n"

"\nFull name: " + full\_name +

"\nBirth data: " + birth\_date +

"\nParent full name: " + full\_name\_parent +

"\nParent phone number: " + phone\_numb\_parent;

}

ui->textEdit->append(infoText);

## **Класс check\_student**

Описываемый класс представляет собой уточняющее окно, для более понятного внешнего вида приложения, однако имеет некоторые логические особенности. Основная логика возложена на метод on\_find\_button\_clicked(). Его задачей является проверка наличия искомого «учащегося» в базе данных. Для этого производится select-запрос к базе данных и в случае его успеха введенное в поле, на экране пользователя, значение отправляется в качестве одного из параметров (getFull\_name\_st). В случае отсутсвия «учащегося» с таким именем в базе данных, пользователь увидит на экране соответствующее сообщение.

setFull\_name\_st(ui->student\_full\_name->text());

if (getFull\_name\_st() != ""){

question\_to\_db = "select id from student where full\_name\_st = '" + getFull\_name\_st() + "'";

query.exec(question\_to\_db);

if (!query.next()) {

QMessageBox::warning(this, "Student Full Name", "This student is not presented in DataBases");

}else{

close();

student\_info student\_window(this, getLogin(), getFull\_name\_st());

student\_window.setModal(true);

student\_window.exec();

}

}

else{

QMessageBox::warning(this, "Checking student info", "You need to input full name of student!");

}

## **Класс student\_info**

Данный класс является логическим завершением блока поиска и отображения информации о конкретном учащемся. Первым делом устанавливается имя учащегося с помощью метода setFull\_name\_st(). Затем к базе данных формируется запрос с выделением всей информации об учащемся, доступной в базе данных, кроме гендера.

setFull\_name\_st(full\_name\_st);

question\_to\_db = "select \* from student WHERE full\_name\_st = '" + getFull\_name\_st() + "'";

query.exec(question\_to\_db);

while (query.next()) {

full\_name = query.value(0).toString();

//gender = query.value(1).toString();

student\_id = query.value(2).toString();

studying\_group\_id = query.value(3).toString();

birth\_date = query.value(4).toString();

stud\_parent\_id = query.value(5).toString();

}

Затем, для интуитивно понятного вида, конвертируется идентификационный номер учебного класса в привычный всем (например, id = «1» преобразуется к виду - 10 «А»).

question\_to\_db = "select num, profile from studying\_group where id = '" + studying\_group\_id + "'";

query.exec(question\_to\_db);

while (query.next()) {

gr\_num = query.value(0).toString();

gr\_prof= query.value(1).toString();

}

Следом осуществляется поиск информации о «родителях» «учащегося» с выделением такой информации как полное имя и номер телефона.

question\_to\_db = "select full\_name\_parent, phone\_numb\_parent from stud\_parent where id = '" + stud\_parent\_id + "'";

query.exec(question\_to\_db);

while (query.next()) {

full\_name\_parent = query.value(0).toString();

phone\_numb\_parent= query.value(1).toString();

}

Следующий запрос в базе данных – получение всех отметок учащегося по всем, изучаемым им, учебным предметам.

question\_to\_db = "select string\_agg(mark::text, ',' order by mark::text desc) as mark , sub\_name from subject join condition c on subject.id = c.subject\_id join lesson\_status ls on c.lesson\_status\_id = ls.id join mark\_id mi on ls.id = mi.lesson\_status\_id join student s on mi.student\_id = s.id where mi.student\_id = '" + student\_id + "' group by sub\_name ";

query.exec(question\_to\_db);

while (query.next()) {

marks = query.value(0).toString();

subject = query.value(1).toString();

ui->textEdit\_marks->append(subject + ": " + marks);

}

Также создается запрос для получения средней отметки по каждому из предметов.

question\_to\_db = "select avg(mark), sub\_name from subject join condition c on subject.id = c.subject\_id join lesson\_status ls on c.lesson\_status\_id = ls.id join mark\_id mi on ls.id = mi.lesson\_status\_id join student s on mi.student\_id = s.id where mi.student\_id = '" + student\_id + "' group by sub\_name";

query.exec(question\_to\_db);

while(query.next()){

marks = query.value(0).toString();

ui->textEdit\_avg\_marks->append(marks);

}

Самым последним является запрос для получения заметок.

question\_to\_db = "select notes\_id from student\_note where student\_id = '" + student\_id + "'";

query.exec(question\_to\_db);

while (query.next()) {

note\_id = query.value(0).toString();

QString notesQuestion;

notesQuestion = "select note from notes where id = '" + note\_id + "'";

query\_getnotes\_full.exec(notesQuestion);

QString note;

while (query\_getnotes\_full.next()){

note = query\_getnotes\_full.value(0).toString();

ui->textEdit\_comments->append(note);

}

}

В завершении, на экран выводится все полученная ранее информация с помощью метода insert().

ui->lineEdit\_full\_name->insert(full\_name);

ui->lineEdit\_dateOfBirth->insert(birth\_date);

ui->lineEdit\_class\_num->insert(gr\_num + " " + gr\_prof);

ui->lineEdit\_parents\_fullname->insert(full\_name\_parent);

ui->lineEdit\_parents\_phonenum->insert(phone\_numb\_parent);

## **Класс parent\_window**

Для данного класса код с логикой имеет схожий вид, как для класса student\_info. Основное отличие заключается лишь в запросе, который ищет данные об «учащемся» через соответствие «учащегося» «родителю».

question\_to\_db = "select full\_name\_st, studying\_group\_id, id from student where stud\_parent\_id = '" + parent\_id + "'";

query.exec(question\_to\_db);

while (query.next()) {

full\_name\_st = query.value(0).toString();

studying\_group\_id = query.value(1).toString();

student\_id = query.value(2).toString();

}

## **Класс student\_window**

Этот класс является самым «бедным» по функционалу. В его задачи входит отображение отметок «учащегося»: всех и среднего значения. Вначале необходимо получить идентификационный номер учащегося по его полному имени. После чего используя этот номер (student\_id) выполняются два запроса к базе данных с целью выявления отметок.

question\_to\_db = "select string\_agg(mark::text, ',' order by mark::text asc) as mark , sub\_name from subject join condition c on subject.id = c.subject\_id join lesson\_status ls on c.lesson\_status\_id = ls.id join mark\_id mi on ls.id = mi.lesson\_status\_id join student s on mi.student\_id = s.id where mi.student\_id = '" + student\_id + "' group by sub\_name ";

query.exec(question\_to\_db);

while (query.next()) {

marks = query.value(0).toString();

subject = query.value(1).toString();

ui->textEdit\_marks->append(subject + ": " + marks);

}

question\_to\_db = "select avg(mark), sub\_name from subject join condition c on subject.id = c.subject\_id join lesson\_status ls on c.lesson\_status\_id = ls.id join mark\_id mi on ls.id = mi.lesson\_status\_id join student s on mi.student\_id = s.id where mi.student\_id = '" + student\_id + "' group by sub\_name";

query.exec(question\_to\_db);

while(query.next()){

marks = query.value(0).toString();

ui->textEdit\_avg\_marks->append(marks);

}

В последствие полученные значения отметок выводятся на экран с использованием метода append().