

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1 ОБЗОР ЛИТЕРАТУРЫ	9
1.1 Обзор существующих аналогов.....	9
1.2 Обзор технологий.....	12
1.2.1 C++	12
1.2.2 Oracle SQL Developer Data Modeler	13
1.2.3 PostgreSQL	14
1.2.4 Фреймворк QT Creator.....	15
1.2.5 Классы QSqlQuery, QDesktopServices	16
1.2.6 API Яндекс ID	16
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ.....	18
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ.....	22
3.1 Блок авторизации пользователей.....	22
3.1.1 Класс mainwindow	22
3.1.2 Класс Authorization	23
3.2 Блок реляционной базы данных	24
3.2.1 Таблица users	24
3.2.2 Таблица teacher	25
3.2.3 Таблица student	25
3.2.4 Таблица stud_parent	26
3.2.5 Таблица ids.....	26
3.2.6 Таблица subject	26
3.2.7 Таблица lesson_status	26
3.2.8 Таблица studying	27
3.2.9 Таблица condition.....	27
3.2.10 Таблица mark_id	27
3.2.11 Таблица studying_group	27
3.2.12 Таблица note.....	28
3.2.13 Таблица teacher_note.....	28
3.2.14 Таблица student_note	28
3.2.15 Таблица lesson.....	28
3.2.16 Таблица lesson_note	29
3.2.17 Таблица timetable	29
3.3 Блок меню взаимодействия с пользователем.....	29
3.4 Блок выбора учащегося для выставления отметки.....	30
3.5 Блок выставления отметки учащемуся.....	31
3.6 Блок выбора объема отображаемой информации об учащихся.....	33
3.7 Блок выбора конкретного учащегося и отображения информации о нем для роли «преподаватель»	34
3.8 Блок выбора группы учащихся и отображения информации о них	38
3.9 Блок отображения информации о конкретном учащемся для роли «родитель»	39

3.10	Блок отображения информации о конкретном учащемся для роли «учащийся»	41
4	РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ	43
4.1	Создание базы данных PostgreSQL	43
4.2	Создание пользователей базы данных PostgreSQL.....	44
4.3	Создание резервных копий базы данных PostgreSQL	46
4.4	Создание таблиц и связей базы данных PostgreSQL с использованием Oracle SQL Developer Data Modeler	47
4.5	Регистрация приложения в API Яндекс ID	49
4.6	Заголовочный файл connection.h.....	51
4.7	Класс mainwindow	53
4.8	Класс authorization	53
4.9	Класс diary_menu	54
4.10	Класс add_mark	55
4.11	Класс add_mark2	56
4.12	Класс check_class	59
4.13	Класс class_info	60
4.14	Класс check_student	62
4.15	Класс student_info	63
4.16	Класс parent_window	65
4.17	Класс student_window	65
5	ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ	67
7	ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ МНОГОПОЛЬЗОВАТЕЛЬСКОГО КЛИЕНТ- СЕРВЕРНОГО ПРИЛОЖЕНИЯ «ЭЛЕКТРОННЫЙ ЖУРНАЛ»	77
7.1	Характеристика многопользовательского клиент-серверного приложения «электронный журнал», разработанного по индивидуальному заказу	77
7.2	Расчет затрат на разработку клиент-серверного приложения «Электронный журнал»	78
7.3	Расчет результата от разработки клиент-серверного приложения «Электронный журнал»	80
7.4	Расчет показателей экономической эффективности разработки клиент-серверного приложения «Электронный журнал».....	81
	ЗАКЛЮЧЕНИЕ	82

ВВЕДЕНИЕ

«Национальное образование Республики Беларусь традиционно является одной из высших ценностей белорусского народа», - гласит официальный сайт министерства образования Республики Беларусь. И с этим, как ни странно, нельзя не согласиться. Данные слова подтверждает и официальная статистика: уровень грамотности взрослого населения - 99,7%, охват базовым, общим средним и профессиональным образованием занятого населения - 98%. Важным фактором таких высоких значений также является тот факт, что каждый третий житель республики учится.

Но за счет чего достигаются такие высокие показатели? Один из факторов – преподаватели и методические указания. Каждый преподаватель того либо иного предмета имеет методические пособия, соответствующие нормам, требованиям министерства образования Республики Беларусь. Второй немаловажный фактор – система контроля знаний обучающихся. Для обеспечения возможности подведения итогов контроля знаний необходимы носители с отметками. Как раз для этих целей все учреждения образования имеют носители, с указанной там текущей аттестацией учащихся. В садах и школах это учебные журналы. В университете зачетные книжки.

С момента начала использования последних и до сегодняшнего дня лидирующее место занимают не цифровые носители, а бумажные. В моем понимании, бумажный носитель устарел и является менее надежным вариантом хранения аттестационной информации. Также, учитывая факт того, что сегодня прогресс движется в сторону виртуализации и информатизации, я считаю актуальным вариант создания данного программного продукта.

Целью данного дипломного проекта является разработка приложения, способного хранить данные о преподавателях, учащихся и их родителях, обрабатывать новые данные (например, добавлять отметки, комментарии к отметке) и отображать имеющиеся данные любому пользователю в пределах его «прав». Также хочется отметить, что основной идеей было создание революционного, в плане внешнего вида, приложения («отойти» от стандартного, привычного всем «с детства» облика)

В соответствии с поставленной целью были определены следующие задачи:

- выбор платформы для создания системы;
- разработка приложения с интуитивно понятным интерфейсом;
- создание приложения с ограничением прав доступа;
- реализация возможности сбора любых данных о пользователях в дальнейшем;
- авторизация через сторонние сервисы, соответствующие стандартам безопасности данных.

Приложение будет разработано для персонального компьютера, и предоставлять следующие возможности:

- авторизация через сторонний сервис;
- функционал, ограниченный в зависимости от роли после авторизации: от просмотра отметок, до возможности их выставления;
- взаимодействие с базой данных приложения для хранения, обновления, отображения, резервного копирования данных;

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор существующих аналогов

На этапе проектирования системы были изучены существующие аналоги. Самым популярным, на сегодняшний день, является образовательная платформа – «SCHOOLS.BY» [1] (рисунок 1.1). Сайт учреждения образования и система электронного учёта успеваемости взаимосвязаны и интегрированы между собой.

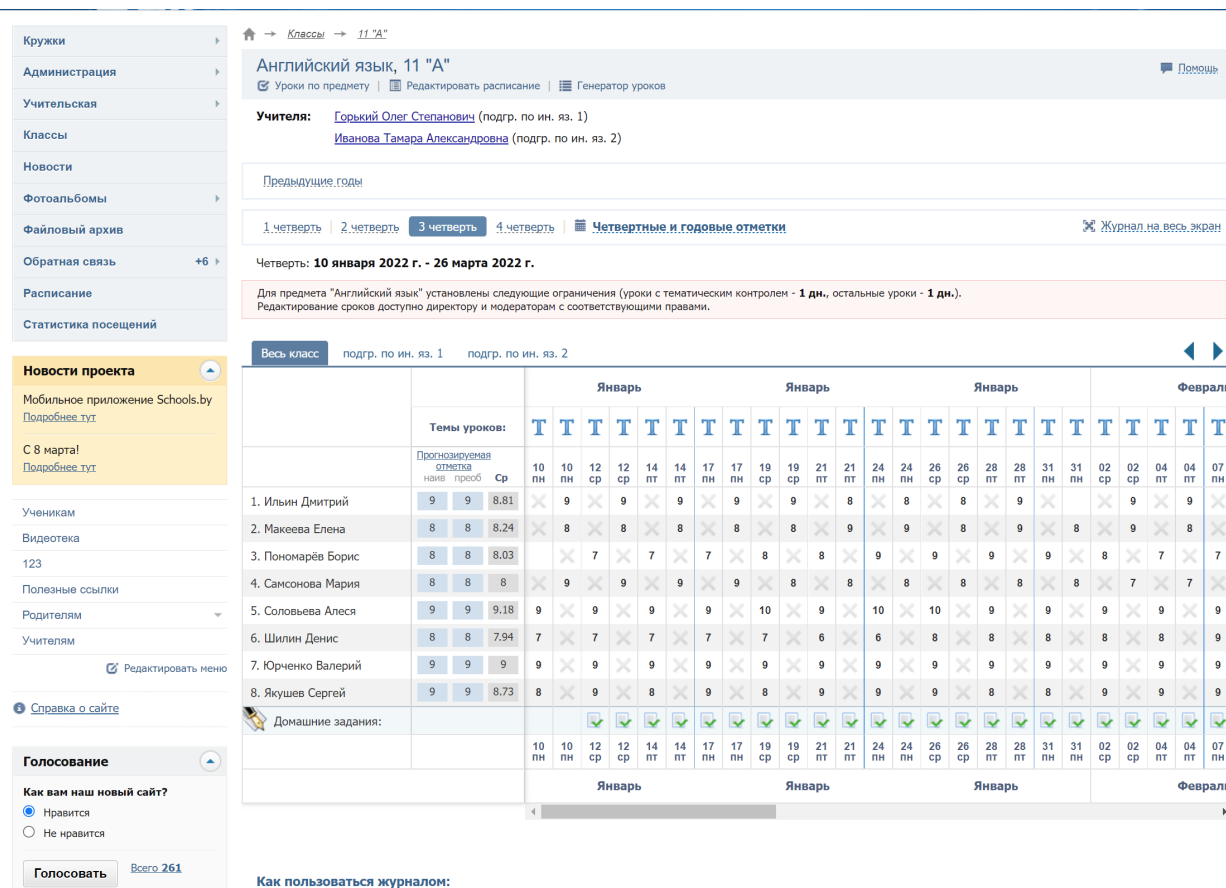


Рисунок 1.1- Образовательная платформа – «SCHOOLS.BY». Электронный журнал

Данная платформа предназначена для обработки и предоставления в электронном виде информации об успеваемости учащихся: отметки и пропуски, комментарии к ним; пометки к урокам; домашние задания; замечания; расписание четвертей, занятий и др., а также смежной информации, доступной через Интернет.

Платформа содержит в себе несколько сервисов: «электронный журнал», «электронный дневник». Суть «электронного журнала» заключается в том, что для каждого класса по каждому предмету создается журнал. Доступ к нему имеют учителя-предметники, классный руководитель,

директор. Родителям и учащимся журнал недоступен. На основании данных, внесенных учителями в журналы, для каждого учащегося формируется его «электронный дневник». В последнем отражено всё, что учителя внесли в журнал (отметки, пропуски, комментарии и т.д.), а также поведение и замечания за каждую неделю. Родители учащегося имеют доступ к данным только своего ребенка.

У данной платформы есть 2 варианта использования для родителей: «базовый» и «расширенный». «Базовый» является полностью бесплатным и включает в себя такие функции как: круглосуточный доступ к информации об успеваемости ребёнка, актуальное расписание занятий и домашнее задание, возможность коммуникации с учителями и администрацией. «Расширенный» вариант является платным, но включает следующие возможности: SMS-оповещения о пропусках учебных занятий, комплексный анализ успеваемости (сервисы «Успеваемость», «Смотритель» и «Аналитика»), информирование о предстоящей контрольной/самостоятельной работе, предварительный прогноз четвертной и годовой отметок, рейтинг успеваемости (достижений) ребёнка, контроль лицевого счёта ребёнка.

Исходя из всего вышесказанного, можно заметить, что данная платформа обладает обширным функционалом и полностью справляется со своими задачами, однако также можно отметить и несколько недостатков данного решения:

- все данные хранятся в сети интернет;
- устаревший дизайн (подобен на классический бумажный вариант);
- при выставлении отметки преподаватель видит все предыдущие значения, что может сказаться на объективности, при выставлении новой;
- сложный пользовательский интерфейс;
- большая стоимость расширенного пакета функций (порядка 70 белорусских рублей за год).

Еще один, не менее популярный, аналог - платформа электронных сервисов для образования «Знай.бай» [2] (рисунок 1.2). Данная платформа позиционирует себя как объединение всех сервисов, необходимых учителям, родителям и школьникам: электронные журналы и дневники, электронные учебные пособия, система школьных платежей (питание, учебники, кружки, попечительские взносы, прочие услуги учреждения образования).

Точно также как и «SCHOOLS.BY», последние интегрируются с Системой электронных дневников и журналов. Имеются также два пакета услуг: «стандартный» и «премиум». «Стандартный» включает в себя следующий функционал: актуальное расписание уроков с учетом замен, Домашнее задание, внесенное учителем, ссылки на дополнительный материал (видео, аудио и другие ресурсы) к домашнему заданию, информация о пропусках и отметках, сообщения от классного руководителя, возможность отправлять выполненное домашнее задание, круглосуточная техническая поддержка. «Премиум» пакет содержит более обширный функционал:

Электронный журнал

Рисунок 1.2 - Платформа электронных сервисов для образования
«Знай.бай»

Сделаем небольшой вывод. Оба аналога хранят абсолютно все данные в сети интернет, что может негативно сказаться на безопасности личных данных. Также не стоит забывать про риски потери соединения с сетью интернет, а также необходимость в стабильном подключении к последнему. Аналоги не придумывают новый взгляд на классические инструменты

(бумажные журналы и дневники), а лишь копируют их. Интерфейс имеет множество кнопок и вкладок, что не всегда понятно простому пользователю. Зачастую появляются сообщения об отказе в доступе к ресурсу, т.к. внешний вид приложений шаблонный. Стоимость продукта для конечного потребителя слишком высока, что отталкивает от приобретения последнего.

1.2 Обзор технологий

1.2.1 C++

C++ - один из старейших и наиболее эффективных языков программирования, который до сих пор продолжает доминировать в сфере программирования и написания приложений для персональных компьютеров. Это можно увидеть по рейтингу [3] (рисунок 1.3).

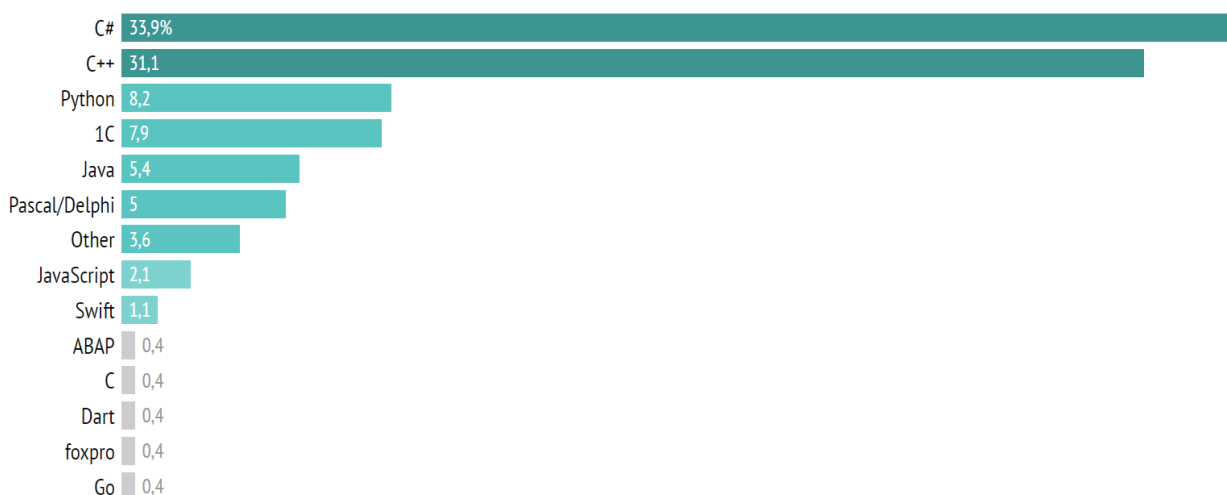


Рисунок 1.3 – Рейтинг языков программирования для написания приложений для персональных компьютеров

Не стоит забывать и тот факт, что язык программирования C++ до сих пор используется во всех сферах деятельности программирования: от высоконагруженных систем до программирования микроконтроллеров. На C++ без проблем можно написать такое программное обеспечение как web-сервер, игры, компоненты программ и так далее. Преимуществами данного языка можно назвать такие факторы, что он является:

- портативный. C++ предлагает такие возможности как переносимость или независимость от платформы, которая позволяет пользователю легко запускать одну и ту же программу в разных операционных системах или интерфейсах (например, Windows OS и Linux);
- объектно-ориентированный. Язык включает в себя такие понятия, как классы, наследование, полиморфизм, абстракция данных и инкапсуляция,

которые обеспечивают повторное использование кода и делают программу еще более надежной.

- общество пользователей. Язык хоть и является старым, однако на нем программируют до сих пор, что дает нам возможность найти в сети интернет ответы на почти любой интересующий нас вопрос.

Среди недостатков можно выделить такие минусы как:

- обычно используется для приложений, зависящих от платформы (мобильные приложения или приложения для персональных компьютеров);
- нет встроенной поддержки потоков.

1.2.2 Oracle SQL Developer Data Modeler

Oracle SQL Developer Data Modeler [4] – графический инструмент для создания, редактирования, просмотра логических, реляционных, физических моделей. Он может подключаться к любой поддерживаемой базе данных Oracle и не зависит от выбранной платформы. Благодаря данному инструменту можно визуализировать базы данных (рисунок 1.4), а также генерировать, из визуализированных баз данных, файлы формата DDL (Data Definition Language) (язык описания данных) (рисунок 1.5).

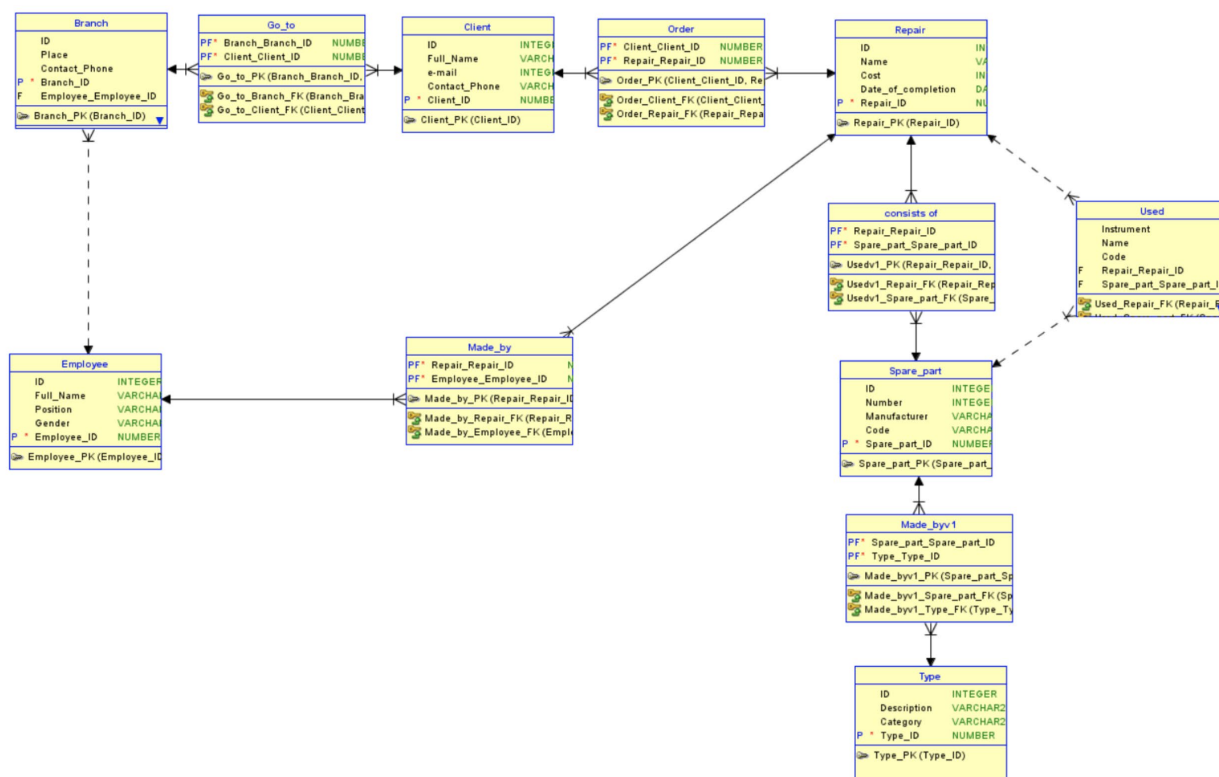


Рисунок 1.4 – Пример реляционной диаграммы, полученной в Oracle SQL Developer Data Modeler

```
CREATE TABLE branch (
    id                INTEGER,
    place             VARCHAR2(4000),
    contact_phone     INTEGER,
    branch_id         NUMBER NOT NULL,
    employee_employee_id NUMBER
);

ALTER TABLE branch ADD CONSTRAINT branch_pk PRIMARY KEY ( branch_id );
```

Рисунок 1.5 – Пример содержимого DLL файла, полученного в Oracle SQL Developer Data Modeler

1.2.3 PostgreSQL

PostgreSQL [5] - система объектно-реляционных баз данных с открытым исходным кодом, которая использует и расширяет язык SQL в сочетании со многими функциями, позволяющими безопасно хранить и масштабировать самые сложные рабочие нагрузки данных. У PostgreSQL есть прямой аналог/конкурент в лице MySQL (система управления реляционными базами данных), однако при выборе системы мой выбор пал на PostgreSQL, т.к. его поддерживает большее количество языков программирования, например, MySQL не поддерживается языком JavaScript, что может сказаться на дальнейшем масштабировании проекта (например, при усовершенствовании версии для персональных компьютеров к веб-версии приложения). Также можно отметить тот факт, что PostgreSQL «чувствителен» к регистру символов в запросе, в то время как MySQL не различает его вовсе. Если строки в запросах не будут в точности совпадать с полями в базе данных, то запрос не будет выполнен. В остальном данные системы подобны друг другу.

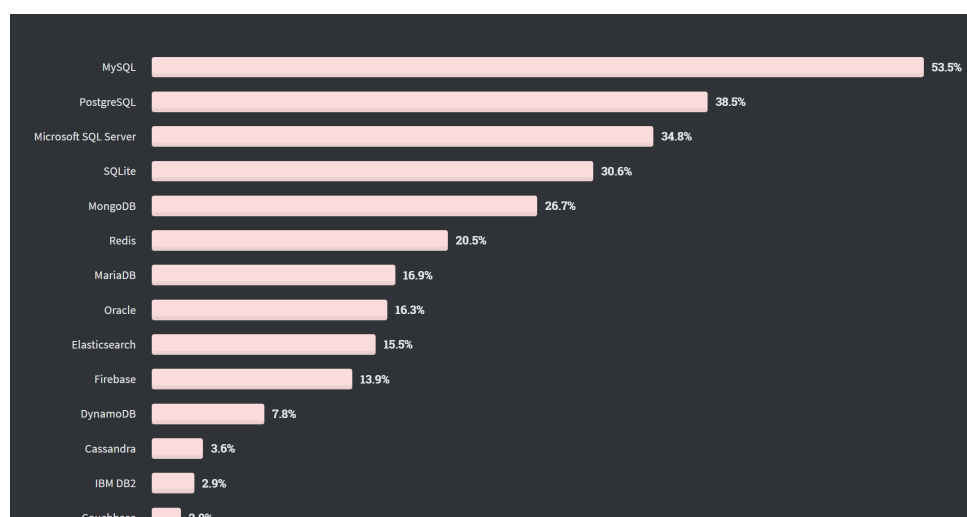


Рисунок 1.6 – Выбор систем баз данных, среди профессиональных разработчиков в 2020 году

Каждый год Stack Overflow [6] (система вопросов и ответов о программировании) опрашивает профессиональных разработчиков для получения статистики. Проанализировав ее, можно увидеть факт роста числа пользователей PostgreSQL в прошлом (2021-м) году по сравнению с позапрошлым (2020-м) (рисунки 1.6 - 1.7), что явно говорит о доверии к этой системе, среди разработчиков.

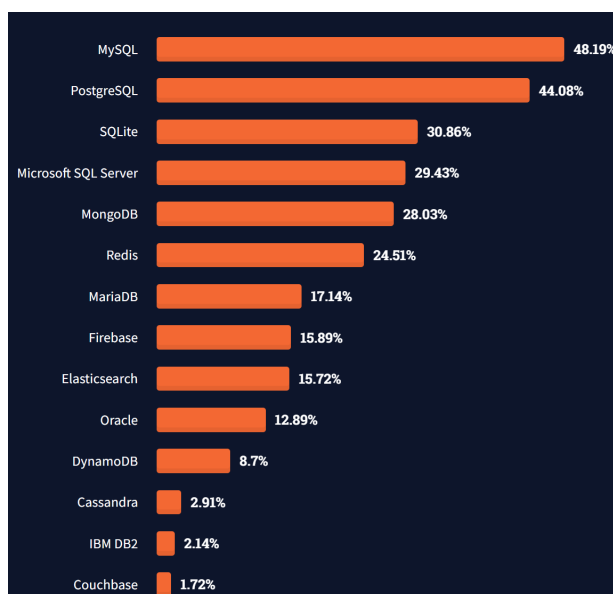


Рисунок 1.7 – Выбор систем баз данных, среди профессиональных разработчиков в 2021 году

Как небольшой итог всего вышесказанного можно выделить основные преимущества PostgreSQL:

- надежность (полное соответствие принципам ACID - атомарность, непротиворечивость, изолированность, сохранность данных);
- производительность (основывается на использовании индексов, интеллектуальном планировщике запросов, тонкой системы блокировок, системе управления буферами памяти и кэширования, превосходной масштабируемости при конкурентной работе);
- поддержка SQL;
- богатый набор типов данных;
- простота использования;
- безопасность данных.

1.2.4 Фреймворк QT Creator

Qt Creator [7] - это кроссплатформенный фреймворк, написанный на C++, предназначенный для разработки приложений для настольных и мобильных устройств, а также встроенных систем. Qt Creator сочетает в себе

кроссплатформенное приложение для разработки программного обеспечения, среду графического пользовательского интерфейса (GUI) и набор инструментов для разработки приложений с использованием стандарта C++.

Главнейшими преимуществами данного фреймворка, по сравнению с другими является то, что:

- разработка приложений позволяет импортировать приложение на несколько платформ с помощью перекомпиляции проекта;
- кроссплатформенная разработка программного обеспечения позволяет легко создавать интуитивно понятные интерфейсы для всех пользователей, независимо от используемой операционной системы.

Аналогом данного фреймворка является Visual Studio [8] (продукт компании Microsoft, включающий интегрированную среду разработки программного обеспечения) в котором есть возможность создания Windows Forms. По сравнению с QT Creator Visual Studio предоставляет возможность написания программного обеспечения на многих популярных языках (например, C#, C++ и другие), имеет встроенный отладчик, которого, к слову, нет в QT Creator, однако графический интерфейс пользователя менее продвинут и предоставляет меньше возможностей.

1.2.5 Классы QSqlQuery, QDesktopServices

Класс QSqlQuery [9] является встроенным в QT Creator. Его функционал заключается в инкапсуляции функциональности, связанной с созданием, навигацией и извлечением данных из SQL запросов к базе данных. Некоторые из предоставляемых методов:

- `next()`. Извлекает следующую запись в результат;
- `previous()`. Извлекает предыдущую запись в результат;
- `first()`. Извлекает первую запись в результат;
- `last()`. Извлекает последнюю запись в результат;
- `seek()`. Извлекает запись под определенным индексом;

Класс QDesktopServices [10] также встроен в QT Creator, но предоставляет методы для доступа к общим службам рабочего стола, таких как открытие веб-страницы. Метод `openUrl()` является основным. Его функционал заключается в открытии указанных URL-адресов в соответствующем веб-браузере для среды рабочего стола пользователя и возвращается в значение «true» случае успеха и «false» в случае неудачи.

1.2.6 API Яндекс ID

API (Application Programming Interface — «программный интерфейс приложения») - описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой. С помощью API Яндекс ID [11] можно подключить механизм авторизации пользователей к приложению

с минимальными затратами собственных ресурсов. Данный интерфейс использует протокол OAuth, который позволяет предоставить третьей стороне ограниченный доступ к защищённым ресурсам пользователя без необходимости передавать ей логин и пароль. Таким образом приложение получает не личные данные пользователя, а лишь OAuth-токен. Среди возможных аналогов для сторонней авторизации «Google Cloud Platform» [12] предоставляет «Google+ API», «Gmail API», однако при выборе и детальном сравнении платформ между собой были отмечены следующие недостатки решения от «Google Cloud Platform»:

- авторизация возможно только посредством аккаунтов в системе «Google», в то время как «Яндекс» предоставляет возможность авторизации через другие сервисы (например, тот же самый «Google») (рисунок 1.8);
- «Google Cloud Platform» является платным решением. На использование API установлены ограничения, которые позволяют авторизовать бесплатно лишь небольшое количество пользователей, в то время как «Яндекс ID» не имеет ограничений в использовании;
- Все API от «Яндекс» являются бесплатными, в то время как часть «Google API» являются платными, а другая часть условно бесплатными.

а)

The screenshot shows the Google login interface. At the top is the Google logo. Below it is the word 'Вход' (Login) and the text 'Переход в Google Cloud Platform'. There is a text input field labeled 'Телефон или адрес эл. почты' (Phone or email address). Below the field is a link 'Забыли адрес электронной почты?' (Forgot email address?). Further down, there is a note 'Работаете на чужом компьютере? Включите гостевой режим. Подробнее' (Working on someone else's computer? Enable guest mode. Read more). At the bottom left is a link 'Создать аккаунт' (Create account), and at the bottom right is a blue button labeled 'Далее' (Next).

б)

The screenshot shows the Yandex ID login interface. At the top is the Yandex logo and the text 'Войдите с Яндекс ID' (Log in with Yandex ID). Below it is the text 'Введите ваш ID' (Enter your ID) and a text input field labeled 'Телефон или почта' (Phone or email). There is a link 'Не помню' (I don't remember). Below the field is a yellow button labeled 'Войти' (Log in). Further down is a button labeled 'Создать ID' (Create ID). At the bottom, there is a row of social media icons: VK, Facebook, Google, and QR code. A dropdown menu is open, showing options: Mail.ru, Одноклассники (Odnoklassniki), and Twitter.

Рисунок 1.8 – Авторизация в системах: а – «Google Cloud Platform»; б – «Яндекс ID»

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

В результате изучения требований к разрабатываемой системе перейдем к разбиению системы на функциональные блоки (модули). Данный подход позволяет создавать более гибкую архитектуру приложения, что позволяет модернизировать существующие блоки и создавать новые без внесения значительных изменений в общую схему работы всей системы целиком.

В разрабатываемом настольном приложении выделяются следующие общие блоки:

- блок авторизации пользователей;
- блок реляционной базы данных.

Далее, в зависимости от роли, полученной после взаимодействием с блоком авторизации, выделяют разные блоки. При получении роли «преподаватель» выделяют следующие блоки для взаимодействия с ними:

- блок меню для взаимодействия с пользователем;
- блок выбора учащегося для выставления отметки;
- блок выставления отметки учащемуся;
- блок выбора объема отображаемой информации об учащихся;
- блок выбора конкретного учащегося и отображения информации о нем для роли «преподаватель»;
- блок выбора группы учащихся и отображения информации них.

При получении роли «родитель», в дополнение к общим блокам, выделяют следующий блок для взаимодействия с ними:

- блок отображения информации о конкретном учащемся для роли «родитель».

При получении роли «учащийся», в дополнение к общим блокам, выделяют следующий блок для взаимодействия с ними:

- блок отображения информации о конкретном учащемся для роли «учащийся».

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.004 С1.

Каждый из представленных выше модулей необходим для выполнения определенной заранее задачи, однако ничего не мешает масштабировать их в дальнейшем. Чтобы система могла функционировать полноценно, каждый модуль взаимодействует друг с другом передавая идентификационные данные пользователя (имя пользователя).

Рассмотрим детальнее функциональные блоки приложения.

Блок реляционной базы данных включает данные, используемые настольным приложением во время его использования. Для реализации данного блока использовалась база данных PostgreSQL. Она, как и любая другая база данных, позволяет хранить и обрабатывать информацию в структурированном виде. Структуризация происходит благодаря таблицам и связям между ними. В данном проекте используются два вида связей:

– многие ко многим. Пример из проекта – «У учащегося может быть несколько преподавателей. У преподавателя может быть несколько учащихся». Для реализации данной связи используется «посредник» (дополнительная таблица) между двумя рассматриваемыми таблицами. Он хранит два внешних ключа, первый из которых ссылается на первую таблицу, а второй - на вторую;

– один к одному. Пример из проекта – «Один учащийся может принадлежать лишь одной учебной группе. Однако одна учебная группа может включать в себя несколько учеников». При реализации данного вида связи также необходим «посредник». Он хранит лишь один ключ, который ссылается на таблицу со множественной связью.

Также можно отметить факт того, что PostgreSQL управляет доступом при помощи так ролей. Роли могут быть членами других ролей, что позволяет им наследовать параметры привилегий, определённых ранее ролей. Что будет использовано при разработке приложения и описано позже.

Блок авторизации пользователей является модулем, необходимым для обеспечения безопасности при включении приложения, разграничения прав пользователей, защиты от несанкционированного доступа в личным данным. Главная возможность данного блока – авторизация пользователя, без передачи личных данных, кроме индивидуального номера в приложении. Этот блок взаимодействует со следующими блоками:

- реляционной базы данных;
- взаимодействия с пользователем;
- отображения информации о конкретном учащемся для роли «учащийся»;
- отображения информации о конкретном учащемся для роли «родитель».

В дальнейшем, при масштабировании, планируется добавление варианта регистрации новых пользователей в приложении, используя описанный ранее блок.

Блок меню для взаимодействия с пользователем доступен пользователям, авторизовавшимся с ролью «преподаватель». Основная его задача – предоставить интуитивно понятный выбор дальнейшего действия посредством интерфейса (выставление, просмотр отметок; просмотр контактной информации). Взаимодействует данный блок со следующими блоками:

- блок авторизации;
- блок выбора объема отображаемой информации об учащемся;
- блок выбора учащегося для выставления отметки;
- блок выставления отметки учащемуся.

Блок выбора учащегося для выставления отметки также доступен пользователям с ролью «преподаватель». Он необходим для конкретизации учащегося, которому в дальнейшем будет выставлена отметка и, по желанию,

комментарий к ней. Данный блок напрямую взаимодействует с блоками реляционной базы данных, для понимания существует студент, которому намереваются выставить отметку, и блоком авторизации, для понимания кто именно хочет выставить отметку. Взаимодействует описанный выше блок со следующими блоками:

- реляционной базы данных;
- выставления отметки учащемуся;
- меню взаимодействия с пользователем.

Блок выставления отметки учащемуся является логическим продолжением блока выбора учащегося для выставления отметки. Он позволяет завершить процесс выставления отметки учащемуся путем ввода значения в соответствующее поле. В данном блоке также производится проверка: имеет ли возможность ранее авторизованный «преподаватель» выставить отметки указанному в прошлом блоке учащемуся. Производится это в данном блоке сугубо из соображений безопасности и производительности: при выставлении отметок придется делать обращение к этим переменным и будет не целесообразно получать их новыми запросами к базе данных или передавать как параметры одного блока другому. Этот блок взаимодействует с такими блоками как:

- реляционной базы данных;
- выбора учащегося для выставления отметки;
- меню для взаимодействия с пользователем.

Блок выбора объема отображаемой информации об учащихся доступен сугубо пользователям с ролью «преподаватель». Используется в целях упрощения интерфейса для конечного пользователя, с целью понимания последним какую информацию необходимо получить: о конкретном студенте или их группе. Взаимодействие данного блока ограничивается связью со следующими блоками:

- реляционной базы данных;
- меню взаимодействия с пользователем;
- выбора группы учащихся и отображения информации о них
- выбора конкретного учащегося и отображения информации о нем

для роли «преподаватель».

Блок выбора конкретного учащегося и отображения информации о нем для роли «преподаватель» является продолжением блока выбора объема отображаемой информации и также доступен лишь «преподавателям». Он предназначен для выбора конкретного обучающегося и дальнейшей визуализации основной информации о нем:

- отметок;
- комментариев;
- контактных данных «родителей» выбранного учащегося;

Этот блок связан с такими блоками как:

- реляционной базы данных;
- блок выбора объема отображаемой информации об учащихся.

Блок выбора группы учащихся и отображения информации них также рассматривается как продолжение блока выбора объема отображаемой информации и доступен для пользователей с ролью «преподаватель». Он используется для:

- отображение краткой информации об обучающихся во всех группах (при условии, что конкретная группа учащихся (класс) не указана);
- определенной группы (класса), при конкретном вводе.

Вводимая информация проверяется на наличие запрашиваемых данных в базе данных. Взаимодействие этого блока происходит со следующими блоками:

- реляционной базы данных;
- выбора объема отображаемой информации об учащихся;
- меню взаимодействия с пользователем.

Блок отображения информации о конкретном учащемся для роли «родитель» является единственным «дополнительным» блоком для пользователей с ролью «родитель». Он отвечает сугубо за корректность отображения информации о их «ребенке»:

- классе, в котором обучается ребенок;
- комментариях к учащемуся от преподавателей;
- контактной информации об учреждении.

Блок отображения информации о конкретном учащемся для роли «учащийся» точно как и предыдущий является уникальным: доступен пользователям лишь с ролью «учащийся». Его функционал ограничен отображением сугубо отметок об авторизованном «ученике».

Последние два блока взаимодействуют лишь с блоком реляционной базы данных.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В этом пункте будет описано, на какие логические блоки делится приложение, из каких классов состоят эти логические блоки, какие методы включают в себя эти модули и какую функциональность выполняют эти методы.

Программа состоит из множества классов, каждый из которых выполняет важную для функциональности блоков работу. Приложение имеет интуитивно понятный и логичный интерфейс. При запуске, пользователь видит окно авторизации, через которое происходит дальнейшая идентификация пользователей. Так как это окно является первым, то в фреймворке QT Creator его принято называть `mainwindow`. Оно построено на базе стандартного для фреймворка `QMainWindow` классе, который и будет являться родительским классом.

Важной особенностью выбранного фреймворка является механизм сигналов и слотов. В Qt введена техника, альтернативная функциям обратного вызова: используются сигналы и слоты. Сигнал испускается, когда происходит определенное событие. Виджеты Qt имеют множество предопределенных сигналов, и всегда можно создать их подклассы, чтобы добавить свои сигналы. Слот — это функция, вызываемая в ответ на определенный сигнал.

Далее будут последовательно описаны все блоки, упомянутые в разделе системного проектирования, классы, требующиеся для работы различных библиотек, а также интерфейсы, реализованные в ходе их разработки.

3.1 Блок авторизации пользователей

Блок авторизации пользователей состоит из нескольких классов, таких как: `MainWindow` и `Authorization`. Данные классы необходимы для первого подключения к блоку базы данных, авторизации пользователей и отображения информационных сообщений.

3.1.1 Класс `mainwindow`

Рассмотрим класс `mainwindow`. Этот класс является базовым для приложений на QT Creator. Используется в приложении для «приветствия» пользователя, для отображения информации, в виде дальнейшей инструкции, для новых пользователя, первого подключения к блоку базы данных и перенаправляет пользователя к классу `Authorization`.

Описанный класс состоит лишь из одного поля:

– `message: QMessageBox` – Оно необходимо для формирования информационного сообщения, которое будет выводиться пользователю с помощью встроенного в фреймворк метода `setText()`;

Методы, реализуемые классом `mainwindow`:

- метод `private UI::MainWindow` – создает экземпляры виджетов.

Компилятор в свою очередь получает информацию из формы `.UI` и генерирует код создания виджета в сгенерированных исходных файлах;

Слоты, реализуемые классом `mainwindow`:

- слот `void on_authorize_button_clicked()` – приватный.

Реализуется после получения сигнала «`clicked`» («нажат») от клиента. После вызова данного слота происходит отображение сообщений, подключение к базе данных, переход к окну авторизации.

3.1.2 Класс `Authorization`

Следующий по очереди класс `Authorization`. Данный класс используется в приложении для авторизации пользователя. Он перенаправляет пользователя на сторонний сервис: Яндекс ID, в котором заранее зарегистрировано приложение и после удачной авторизации сервис возвращает токен пользователя в приложении сервиса, по которому пользователь уже авторизуется в приложении «Электронный журнал».

Описанный класс состоит из следующих полей:

- `query: QSqlQuery` – используется для выполнения SQL запросов к базе данных. Применительно к разработанному приложению используется совместно с методами `exec()` и `next()`;

- `question_to_db: QString` – строка, необходимая для формирования запросов к SQL базе данных. Применяется в качестве передаваемого параметра для метода `exec()`;

- `user_id: QString` – строка, хранящая переменную с личным номером пользователя в базе данных. Используется для формирования запроса к базе данных, как часть конструкции `question_to_db`;

- `full_name: QString` – строка, содержащая полное имя авторизованного пользователя, необходимое для его отображения в интерфейсе приложения в дальнейшем, в качестве передаваемого параметра. Формируется в результате присваивания возвращаемого значения от запроса к базе данных;

- `role_num: int` – номер роли в базе данных. Необходим для понимания роли авторизованного пользователя, с целью предоставления определенных прав при использовании базы данных и отображения необходимых окон разработанного приложения;

- `token: QString` – приватное поле, хранящее изначально незашифрованный личный номер пользователя в приложении стороннего сервиса, однако после манипуляций, которые будут описаны в следующем разделе, хранящее значение, зашифрованное по алгоритму SHA-256;

- `token_toUtf8: QByteArray` – также приватное поле, хранящее личный номер пользователя в приложении стороннего сервиса, но в отличии

от обычного token-a, оно зашифровано по стандарту UTF-8.

Методы, реализуемые классом `mainwindow`:

- метод `private UI::MainWindow` – создает экземпляры виджетов. Компилятор в свою очередь получает информацию из формы `.UI` и генерирует код создания виджета в сгенерированных исходных файлах;

- метод `public getToken()` – получает значение token-a. Используется при формировании запросов к базе данных, видоизменения значения переменной `token_toUtf8`;

- метод `public setToken()` – устанавливает определенное значение token-a;

- метод `public getToken_toUtf8()` – получает значение `token_toUtf8`. Используется при шифровании значения token-a;

- метод `public setToken_toUtf8()` – устанавливает определенное значение `token_toUtf8`.

Слоты, реализуемые классом `mainwindow`:

- слот `void on_accept_button_clicked()` – приватный. Реализуется после получения сигнала «clicked» («нажат») от клиента. После его реализации происходит выполнение всей логики данного окна: шифрование значения token-a, передача его к базе данных, получение от базы данных ответного значения с личным номером пользователя в разработанном приложении и номером его роли. После чего происходит «перенаправление» к оставшимся окнам согласно блокам приложения;

3.2 Блок реляционной базы данных

Как говорилось ранее, в качестве базы данных использовалась PostgreSQL. Ее администрирование производится с помощью бесплатно предоставляемого средства `pgAdmin`. База данных состоит из некоторого множества таблиц, с заполненными полями. Ниже будут описаны все таблицы используемой базы данных с содержащимися в них полями.

3.2.1 Таблица `users`

Таблица, используемая для авторизации пользователя, к которой имеют доступ все пользователи при включении приложения. Состоит из трех полей:

- `user_id: integer` – хранит личный номер пользователя приложения. Выдается вручную администратором базы данных с использованием специальной логики, которая будет описана в следующем разделе. Является первичным ключом, что не позволяет создавать одинаковых записей (строк) в таблице;

- `role_id: integer` – хранит значение, соответствующее номеру роли. Выбранный тип данных `integer` используется сугубо для уменьшения итогового размера базы данных;

– token: character varying – хранит зашифрованное значение личного номера пользователя в стороннем сервисе (Яндекс ID).

3.2.2 Таблица teacher

Таблица, используемая для хранения конкретной информации о пользователях с ролью «учитель» и связи с другими таблицами, благодаря наличию вторичных ключей внутри. Состоит из следующих полей:

– full_name_t: character varying – хранит значение полных фамилии и имени «учителя»;

– gender: character varying – хранит значение пола пользователя. Заполняется одной буквой «М» или «W» (мужской или женский). Используется сугубо для сбора статистики в дальнейшем;

– id: integer – хранит значение идентификационного номера пользователя. Является первичным ключом;

– rank: character varying – строка со значением «звания» преподавателя. Создано для сбора статистик в дальнейшем. Предполагаемые «звания» - учитель, директор, заместитель директора и т.д. и т.п.

– phone_number: character varying – хранит значение личного или рабочего номера телефона преподавателя;

– subject_id: integer – поле, необходимое для понимания, какой предмет преподает «учитель», чтобы избежать выставления отметок по «чужим» предметам (учитель математики не может выставять отметки по лингвистическому предмету). Является внешним ключом, что обеспечивает однозначную логическую связь, между таблицами одной БД;

– birth_date_t: date – еще одно поле для масштабирования приложения или сбора какой-либо информации о педагогах. Хранит информацию о дате рождения «преподавателя».

3.2.3 Таблица student

Таблица, используемая для хранения конкретной информации о пользователях с ролью «учащийся» и связи с другими таблицами, благодаря наличию первичных и внешних ключей внутри. Состоит из следующих полей:

– full_name_st: character varying – хранит значение полных фамилии и имени «учащегося»;

– gender: character varying – хранит значение пола пользователя. Заполняется одной буквой «М» или «W» (мужской или женский). Используется сугубо для сбора статистики в дальнейшем;

– id: integer – хранит значение идентификационного номера пользователя. Является первичным ключом;

– studying_group_id: integer – указывает на идентификационный

номер класса. Является внешним ключом.

- `stud_parent_id: integer` – хранит идентификационный номер родителя «учащегося». Является внешним ключом;
- `birth_date_st: date` – поле, хранящее полную дату рождения «учащегося». Необходимо сугубо для масштабирования приложения или сбора какой-либо информации об учащихся.

3.2.4 Таблица `stud_parent`

Таблица, используемая для хранения полной информации о пользователях с ролью «родитель» и связи с другими таблицами, благодаря наличию первичных ключей внутри. Состоит из трех полей:

- `full_name_parent: character varying` – хранит значение полных фамилии и имени «родителя»;
- `id: integer` – хранит значение идентификационного номера пользователя. Является первичным ключом;
- `phone_numb_parent: character varying` – хранит номер родителя, отображаемый «преподавателю» при попытке получения последним полной информации об определенном «учащемся».

3.2.5 Таблица `ids`

Таблица, используемая для связи таблиц `teacher` и `student`. Состоит из двух полей, являющихся одновременно первичными и внешними ключами:

- `student_id: integer` – хранит уникальное значение идентификационного номера «учащегося»;
- `teacher_id: integer` – хранит уникальное значение идентификационного номера «учителя».

3.2.6 Таблица `subject`

Таблица, используемая для хранения названий предметов и сопоставления этим именам уникальных идентификаторов. Состоит всего из двух полей:

- `sub_name: character varying` – полное название изучаемого предмета;
- `id: integer` – первичный ключ. Сопоставляется определенному предмету.

3.2.7 Таблица `lesson_status`

Таблица, используемая для хранения, так называемого, статуса урока.

Поля, внутри данной таблицы следующие:

- `mark: integer` – значение выставленной отметки «учащегося»;
 - `id: integer` – первичный ключ. Сопоставляется определенному уроку;
 - `visited: character varying` – значение статуса посещения урока.
- Используется для дальнейшего масштабирования приложения. По умолчанию, при выставлении отметки устанавливается как «yes».

3.2.8 Таблица `studying`

Таблица, используемая для связи таблиц `subject` и `student`. Состоит из двух полей, являющихся одновременно первичными и внешними ключами:

- `student_id: integer` – хранит уникальное значение идентификационного номера «учащегося»;
- `subject_id: integer` – хранит уникальное значение идентификационного номера предмета.

3.2.9 Таблица `condition`

Таблица, используемая для связи таблиц `lesson_status` и `subject`. Состоит из двух полей, являющихся одновременно первичными и внешними ключами:

- `lesson_status_id: integer` – хранит уникальное значение идентификационного номера статуса урока;
- `subject_id: integer` – хранит уникальное значение идентификационного номера урока.

3.2.10 Таблица `mark_id`

Таблица, используемая для связи таблиц `lesson_status` и `student`. Состоит из двух полей, являющихся одновременно первичными и внешними ключами:

- `student_id: integer` – хранит уникальное значение идентификационного номера «учащегося»;
- `lesson_status_id: integer` – хранит уникальное значение идентификационного номера статуса урока.

3.2.11 Таблица `studying_group`

Таблица, используемая для хранения значения номера класса и его профиля, что необходима для простоты понимания пользователей (чтобы вместо уникального номера, к примеру, «112211», пользователь видел, к

примеру, значение 11 «А»). Поля, внутри данной таблицы следующие:

- `id: integer` – первичный ключ. Уникально сопоставляется определенной комбинации номера и профиля;
- `num: integer` – значение номера класс (классические значения от «1» до «11»);
- `profile: character varying` – буквенное обозначение класса. Классические значения от «А» до «Е».

3.2.12 Таблица `note`

Таблица, используемая для хранения комментариев. Представлена с двумя полями:

- `note: character varying` – строка с комментарием;
- `id: integer` – первичный ключ. Сопоставляется комментарию, для отсутствия путаницы.

Так как комментарии выставляются определенным «преподавателем» определенному «учащемуся», то появляются две связующие таблицы: `teacher_note` и `student_note`.

3.2.13 Таблица `teacher_note`

Таблица, используемая для связи таблиц `teacher` и `notes`. Состоит из двух полей, являющихся одновременно первичными и внешними ключами:

- `teacher_id: integer` – хранит уникальное значение идентификационного номера «преподавателя»;
- `note_id: integer` – хранит уникальное значение идентификационного номера комментария.

3.2.14 Таблица `student_note`

Таблица, используемая для связи таблиц `teacher` и `notes`. Состоит из двух полей, являющихся одновременно первичными и внешними ключами:

- `student_id: integer` – хранит уникальное значение идентификационного номера «учащегося»;
- `note_id: integer` – хранит уникальное значение идентификационного номера комментария.

3.2.15 Таблица `lesson`

Для возможности дальнейшего масштабирования проекта заранее создана таблица `lesson`, необходимая для составления расписаний занятий. Поля, внутри данной таблицы следующие:

- `id: integer` – первичный ключ. Уникальный номер проведенного занятия;
- `lesson_date: date` – дата проведенного или проводимого занятия;
- `type: character varying` – тип проводимого занятия: лабораторная работа, контрольная работа, обычное занятие.

3.2.16 Таблица `lesson_note`

Таблица, используемая для связи таблиц `lesson` и `notes`. Состоит из двух полей, являющихся одновременно первичными и внешними ключами:

- `lesson_id: integer` – хранит уникальное значение идентификационного номера урока;
- `note_id: integer` – хранит уникальное значение идентификационного номера комментария.

3.2.17 Таблица `timetable`

Таблица, используемая для связи таблиц `lesson` и `notes`. Состоит из двух полей, являющихся одновременно первичными и внешними ключами:

- `lesson_id: integer` – хранит уникальное значение идентификационного номера урока;
- `studying_group_id: integer` – хранит уникальное значение идентификационного номера учебного класса.

3.3 Блок меню взаимодействия с пользователем

Описанный блок представлен в программе классом `diary_menu` состоит из следующих полей:

- `login: QString` – используется для отображения на пользовательском интерфейсе в качестве идентификационных данных авторизованного пользователя. В последующем выступает как передаваемый параметр для других окон.

Методы, реализуемые классом `diary_menu`:

- метод `private UI::diary_menu` – создает экземпляры виджетов. Компилятор в свою очередь получает информацию из формы `.UI` и генерирует код создания виджета в сгенерированных исходных файлах;

- метод `public getLogin()` – получает значение `login-a`. Используется как раз таки для передачи значения переменной в другие окна приложения;

- метод `public setLogin()` – устанавливает определенное значение `login-a`. Первоначально устанавливается значение, переданное предыдущим окном, однако спокойно может быть изменено разработчиком.

Слоты, реализуемые классом `diary_menu`:

– слот `void on_get_data_button_clicked()` – приватный.

Предназначен для отображения следующего окна, уточняющего какого вида информация об обучающихся необходима конечному пользователю. Реализуется после получения сигнала «clicked» («нажат») от клиента. После его реализации происходит перенаправление пользователя на окно уточнения объема получаемой информации об учащихся;

– слот `void on_exit_button_clicked()` – приватный. Реализуется после получения сигнала «clicked» («нажат») от клиента. После его реализации происходит завершение работы приложения, путем использования метода `quit()` стандартного класса фреймворка QT Creator `QApplication`;

– слот `void on_contact_info_button_clicked()` – приватный. Разработан для отображения контактной информации о разработчике и возможного получения дополнительной информации о приложении. Реализуется после получения сигнала «clicked» («нажат») от клиента. После его реализации происходит вызов метода `setText()` стандартного класса `QMessageBox`. Также подключается таймер `QTimer`, отсчитывающий время отображения контактной информации на экране, который пользователь может прервать досрочно, нажав на кнопку «ОК» на экране;

– слот `void on_start_lesson_button_clicked()` – приватный. Разработан для выставления отметок учащимся, после прохождения определенных блоков, которые будут описаны ниже. Реализуется после получения сигнала «clicked» («нажат») от клиента. После его реализации происходит перенаправление к другому диалоговому окну, представленному классом `add_mark`.

3.4 Блок выбора учащегося для выставления отметки

Блок выбора учащегося для выставления отметки представлен в программе одним классом: `add_mark`. Данный класс состоит из следующих полей:

– `login: QString` – используется для отображения на пользовательском интерфейсе в качестве идентификационных данных авторизованного пользователя. В последующем выступает как передаваемый параметр для других окон;

– `student_fullname: QString` – используется для хранения полного имени «учащегося» и дальнейшими операциями над ним;

– `question_to_db: QString` – строка, используемая при формировании запросов к базе данных PostgreSQL. Применяется в качестве передаваемого параметра для метода `exec()`;

– `query: QSqlQuery` – используется для выполнения SQL запросов к базе данных. Применительно к разработанному приложению используется

совместно с методами `exec()` и `next()`.

Методы, реализуемые классом `add_mark`:

- метод `private UI::add_mark` – создает экземпляры виджетов. Компилятор в свою очередь получает информацию из формы `.UI` и генерирует код создания виджета в сгенерированных исходных файлах;

- метод `public getLogin()` – получает значение `login-a`. Используется как раз таки для передачи значения переменной в другие окна приложения;

- метод `public setLogin()` – устанавливает определенное значение `login-a`. Первоначально устанавливается значение, полученное от предыдущего окна;

- метод `public getStudent_fullname()` – получает значение `student_fullname-a`. Используется для передачи значения переменной в другие окна приложения, формирования запросов к SQL базе данных, для получения информации о наличии студента с такими фамилией и именем.

- метод `public setStudent_fullname()` – устанавливает значение переменной `student_fullname` в соответствии с введенными пользователем в «окно ввода» данными.

Слоты, реализуемые классом `diary_menu`:

- слот `void on_back_pushButton_clicked()` – приватный. Предназначен для возврата к блоку меню взаимодействия с пользователем и соответственно к окну, представленным классом `diary_menu`. Реализуется после получения сигнала «`clicked`» («нажат») от клиента. После его реализации происходит перенаправление пользователя к окну меню приложения;

- слот `void on_next_pushButton_clicked()` – приватный. Реализуется после получения сигнала «`clicked`» («нажат») от клиента. После его реализации происходит проверка правильности введенных данных: существует ли пользователь, которому выставляется отметка. Если такой пользователь действительно существует, то происходит перенаправление к следующему окну, представленному классом `add_mark2`. Если же запрашиваемый пользователь не предоставлен в базе данных, то пользователь увидит предупредительное сообщение, с помощью метода `warning()` стандартного для фреймворка QT Creator класса `QMessageBox`, и сможет повторить ввод.

3.5 Блок выставления отметки учащемуся

Данный блок описывается классом `add_mark2`, который в свою очередь состоит из следующих полей:

- `login: QString` – используется для отображения на пользовательском интерфейсе в качестве идентификационных данных

авторизованного пользователя. В последующем выступает как передаваемый параметр для других окон;

- `student_name: QString` – используется для хранения полного имени «учащегося» и дальнейшими операциями над ним;

- `question_to_db: QString` – строка, используемая при формировании запросов к базе данных PostgreSQL. Применяется в качестве передаваемого параметра для метода `exec()`;

- `query: QSqlQuery` – используется для выполнения SQL запросов к базе данных. Применительно к разработанному приложению используется совместно с методами `exec()` и `next()`;

- `subject: QString` – используется для хранения названия учебного предмета и отображения последнего пользователю в понятном виде в форме выставления отметки;

- `subject_id: QString` – используется для хранения идентификационного номера учебного предмета, упрощения формирования запросов к базе данных;

- `student_id: QString` – используется для хранения уникального идентификационного номера «студента», возможности осуществить запросы к базе данных;

- `mark: QString` – выполняет функции хранения значения. Использовано для проверки принадлежности адекватности значения (отметки по учебным предметам во всех образовательных учреждениях на территории Республики Беларусь выставляются в пределах от 0 до 10). Также с помощью этой переменной в дальнейшем формируются `SELECT` и `INSERT` запросы к SQL базе данных;

- `lessonID: QString` – переменная используемая при `INSERT`-запросе к базе данных для выставления отметки «учащемуся» с учетом индивидуального номера каждого занятия;

- `commentID: QString` – переменная используемая при `INSERT`-запросе к базе данных для выставления комментария «учащемуся» с учетом индивидуального номера каждого комментария;

- `lessonStatus: QString` – переменная используемая при формировании `INSERT`-запроса к базе данных, являющаяся обязательной при выставлении отметки «учащемуся». По умолчанию, значение переменной – «yes», чтобы пользователю не было необходимости вводить данное значение каждый раз;

- `comment: QString` – переменная, хранящая значение, введенное пользователем в поле комментариев к отметке. В дальнейшем используется при формировании запросов к базе данных PostgreSQL.

Методы, реализуемые классом `add_mark2`:

- метод `private UI::add_mark2` – создает экземпляры виджетов. Компилятор в свою очередь получает информацию из формы `.UI` и генерирует

код создания виджета в сгенерированных исходных файлах;

- метод `public getLogin()` – получает значение `login-a`. Используется как раз таки для передачи значения переменной в другие окна приложения;

- метод `public setLogin()` – устанавливает определенное значение `login-a`. Первоначально устанавливается значение, полученное от предыдущего окна;

- метод `public getStudent_name()` – получает значение `student_name-a`. Используется для передачи значения переменной в другие окна приложения, формирования запросов к SQL базе данных, для получения информации о наличии студента с такими фамилией и именем.

- метод `public setStudent_name()` – устанавливает значение переменной `student_name` в соответствии с введенными пользователем в «окно ввода» данными.

Слоты, реализуемые классом `diary_menu`:

- слот `void on_pushButton_cancel_clicked()` – приватный. Предназначен для возврата к окну, представленным классом `add_mark` в случае, если, например, был введен не тот пользователь. Реализуется после получения сигнала «`clicked`» («нажат») от клиента. После его реализации происходит перенаправление пользователя к окну выбора пользователя для выставления отметки;

- слот `void on_pushButton_evaluate_clicked()` – приватный. Реализуется после получения сигнала «`clicked`» («нажат») от клиента. После его реализации выполняется основная логика данного окна: проверяется, принадлежит ли введенная отметка рамкам от «0» до «10». В противном случае пользователь увидит предупреждающее сообщение с помощью метода `warning()` класса `QMessageBox`. Далее будет выведено окно проверки ложного ввода: не ошибся ли пользователь с вводом – с помощью метода `question()` класса `QMessageBox`. После подтверждения пользователем своего ввода будут производиться `SELECT` и `INSERT` запросы для получения всех идентификационных номеров:

- вводимого комментария;
- учащегося, которому выставляется отметка;
- урока, на котором выставляется отметка.

3.6 Блок выбора объема отображаемой информации об учащихся

Описанный блок выполняет функцию визуализации и представлен в программе классом `get_data`, который в свою очередь состоит из следующих полей:

- `login: QString` – элемент используемый для отображения на интерфейсе пользователя в качестве короткой информации авторизованного пользователя: фамилия и имя. В последующем выступает как передаваемый

параметр для других окон.

Методы, реализуемые классом `diary_menu`:

- метод `private UI::get_data` – создает экземпляры виджетов. Компилятор в свою очередь получает информацию из формы `.UI` и генерирует код создания виджета в сгенерированных исходных файлах;

- метод `public getLogin()` – получает значение `login-a`. Используется как раз таки для передачи значения переменной в другие окна приложения;

- метод `public setLogin()` – устанавливает определенное значение `login-a`. Первоначально устанавливается значение, переданное предыдущим окном, однако спокойно может быть изменено разработчиком.

Слоты, реализуемые классом `diary_menu`:

- слот `void on_get_back_button_clicked()` – приватный. Предназначен для возврата к блоку меню взаимодействия с пользователем в случае, например, ошибочного выбора (нажатия) в предыдущем блоке. Реализуется после получения сигнала «clicked» («нажат») от клиента. После его реализации происходит перенаправление пользователя к предыдущему окну с передачей в него идентификационных данных в виде фамилии и имени (`login`);

- слот `void on_students_info_button_clicked()` – приватный. Реализуется после получения сигнала «clicked» («нажат») от клиента. После его реализации происходит перенаправление пользователя на окно уточнения информации, предоставленное классом `check_student`: информацию о каком именно студенте необходимо получить;

- слот `void on_classes_info_button_clicked()` – приватный. Реализуется после получения сигнала «clicked» («нажат») от клиента. После его реализации происходит перенаправление пользователя на окно уточнения, предоставленное классом `check_class`, объема получаемой информации об учащихся;

3.7 Блок выбора конкретного учащегося и отображения информации о нем для роли «преподаватель»

Данный блок предназначен для оформления запроса пользователем об отображении полной информации о конкретном выбранном учащемся. Представлен данный блок в программе двумя классами: `check_student` и `student_info`. Разберем каждый из классов по-отдельности. `Check_student` состоит из следующих полей:

- `full_name_st: QString` – элемент используемый для хранения полного имени (фамилии и имени) учащегося, информацию о котором и хочет получить «преподаватель».

- `login: QString` – элемент используемый для отображения на интерфейсе пользователя в качестве короткой информации авторизованного

пользователя: фамилия и имя. В последующем выступает как передаваемый параметр для других окон;

- `query: QSqlQuery` – используется для выполнения SQL запросов к базе данных. Применительно к разработанному приложению используется совместно с методами `exec()` и `next()`;

- `question_to_db: QString` – строка, используемая при формировании `SELECT` запросов к базе данных PostgreSQL. Применяется в качестве передаваемого параметра для метода `exec()`;

Методы, реализуемые классом `diary_menu`:

- метод `private UI::check_student` – создает экземпляры виджетов. Компилятор в свою очередь получает информацию из формы `.UI` и генерирует код создания виджета в сгенерированных исходных файлах;

- метод `public getLogin()` – получает значение `login-a`. Используется как раз таки для передачи значения переменной в другие окна приложения;

- метод `public setLogin()` – устанавливает определенное значение `login-a`. Первоначально устанавливается значение, переданное предыдущим окном, однако спокойно может быть изменено разработчиком;

- метод `public getFull_name_st()` – получает значение `full_name_st`. Используется для передачи значения переменной в другие окна приложения, формирования `SELECT`-запроса для базы данных PostgreSQL;

- метод `public setFull_name_st()` – устанавливает определенное значение `full_name_st`. Первоначально устанавливается значение, переданное предыдущим окном, однако без проблем в случае необходимости может быть изменено разработчиком.

Слоты, реализуемые классом `check_student`:

- слот `void on_cancel_button_clicked()` – приватный. Предназначен для возврата к блоку выбора объема отображаемой информации об учащихся в случае, например, ошибочного ввода в предыдущем блоке. Реализуется после получения сигнала «`clicked`» («нажат») от клиента. После его реализации происходит перенаправление пользователя к предыдущему окну с передачей в него идентификационных данных в виде фамилии и имени (`login`);

- слот `void on_find_button_clicked()` – приватный. Реализуется после получения сигнала «`clicked`» («нажат») от клиента. После его реализации происходит проверка наличия искомого «студента» в базе данных с помощью `SELECT`-запроса в базу данных. В случае удачного запроса произойдет перенаправление пользователя на окно визуализации информации о студенте, предоставленное классом `student_info`. В случае отсутствия информации о введенном учащемся пользователь будет об этом осведомлен с помощью соответствующего сообщения, написанного с использованием

метода `warning()` стандартного класса `QMessageBox`.

В свою очередь класс `student_info` состоит из следующих полей:

- `query: QSqlQuery` – используется для выполнения SQL запросов к базе данных. Применительно к разработанному приложению используется совместно с методами `exec()` и `next()`;

- `query_getnotes_full: QSqlQuery` – используется для выполнения нового SQL запроса во время выполнения предыдущего. В программе этот запрос реализован во время «получения» отметок «учащегося» и комментариев к этим отметкам;

- `question_to_db: QString` – строка, используемая при формировании `SELECT` запросов к базе данных PostgreSQL. Применяется в качестве передаваемого параметра для метода `exec()`;

- `full_name: QString` – элемент используемый для хранения полного имени (фамилии и имени) учащегося, информацию о котором и хочет получить «преподаватель»;

- `studying_group_id: QString` – в данной переменной хранится уникальный идентификационный номер группы, который позже, с помощью `SELECT`-запроса к базе данных, будет преобразовываться в вид, понятный конечному пользователю;

- `birth_date: QString` – элемент используемый для хранения даты рождения студента, являющийся по своей сути, лишь дополнительной информацией на экране приложения для «преподавателя»;

- `student_id: QString` – переменная, хранящая идентификационный номер «студента» выбранного для детального отображения информации. Используется при составлении запросов к базе данных PostgreSQL;

- `stud_parent_id: QString` – переменная, хранящая идентификационный номер «родителя» выбранного для детального отображения «учащегося». Используется при составлении запросов к базе данных PostgreSQL, а потом отображается как контактная информация;

- `gr_num: QString` – строка, хранящая информацию о численном значении ступени образования (например, «10»-й класс). Применяется для понятного пользователю отображения информации о численном значении номера класса;

- `gr_prof: QString` – строка, хранящая информацию о буквенном обозначении класса, в котором обучается выбранный пользователь. Применяется для понятного пользователю отображения информации о численном значении номера класса;

- `note_id: QString` – строка, содержащая идентификационный номер комментария. Сохраняется именно в формат `QString`, поскольку используется только при составлении запроса в базу данных;

- `marks: QString` – строка, в которой содержатся отметки выбранного «учащегося». Принимает значения, полученные от запросов в базу данных

(для получения всех и средней отметок) и моментально выводятся на экран (в соответствующие поля на экране) с помощью использования метода `append()` класса `QTextEdit`.

- `subject: QString` – элемент, хранящий в себе значение, соответствующее названию учебного предмета, по которому поставлена та либо иная отметка. Используется для отображения на экране с помощью использования метода `append()` класса `QTextEdit`. Значение переменной обновляется с помощью присваивания результата `SELECT`-запроса к базе данных.

- `full_name_parent: QString` – переменная, хранящее в себе полное имя (фамилия и имя) «родителя» «учащегося» выбранного для детального отображения информации о последнем. Выводится на экран (в соответствующее поле на экране) с помощью использования метода `insert()` класса `QLineEdit`;

- `phone_num_parent: QString` – строка, содержащая номер телефона «родителя» конкретного «учащегося». Выводится на экран (в соответствующее поле на экране) с помощью использования метода `insert()` класса `QLineEdit`.

Методы, реализуемые классом `student_info`:

- метод `private UI::check_student` – создает экземпляры виджетов. Компилятор в свою очередь получает информацию из формы `.UI` и генерирует код создания виджета в сгенерированных исходных файлах;

- метод `public getLogin()` – получает значение `login-a`. Используется как раз таки для передачи значения переменной в другие окна приложения;

- метод `public setLogin()` – устанавливает определенное значение `login-a`. Первоначально устанавливается значение, переданное предыдущим окном, однако спокойно может быть изменено разработчиком;

- метод `public getFull_name_st()` – получает значение `full_name_st`. Используется для передачи значения переменной в другие окна приложения, формирования `SELECT`-запроса для базы данных PostgreSQL;

- метод `public setFull_name_st()` – устанавливает определенное значение `full_name_st`. Первоначально устанавливается значение, переданное предыдущим окном, однако без проблем в случае необходимости может быть изменено разработчиком.

Слоты, реализуемые классом `student_info`:

- слот `void on_cancel_button_clicked()` – приватный. Является единственным слотом, реализуемым в классе `student_info`. Предназначен для возврата к предыдущему блоку: выбора объема отображаемой информации об учащихся, представленному классом `check_student`. Реализуется после получения сигнала «`clicked`» («нажат») от клиента. После

его реализации происходит перенаправление пользователя к предыдущему окну с передачей в него идентификационных данных в виде фамилии и имени (login);

3.8 Блок выбора группы учащихся и отображения информации о них

Этот блок выполняет функцию визуализации искомой информации и представлен в программе двумя классами: `check_class` и `class_info`. Разберем каждый из классов по-отдельности. Класс `check_class` состоит из следующих полей:

- `query: QSqlQuery` – используется для выполнения SQL запросов к базе данных. Применительно к разработанному приложению используется совместно с методами `exec()` и `next()`;
- `question_to_db: QString` – строка, необходимая для формирования запросов к SQL базе данных. Применяется в качестве передаваемого параметра для метода `exec()`;
- `login: QString` – элемент используемый для отображения на интерфейсе пользователя в качестве короткой информации авторизованного пользователя: фамилия и имя. В последующем выступает как передаваемый параметр для других окон;
- `class_letter: QString` – строка, хранящая информацию о буквенном обозначении класса, в котором обучается выбранный пользователь. Применяется для понятного пользователю отображения информации о численном значении номера класса;
- `class_num: int` – переменная, хранящая информацию о целочисленном значении ступени образования (например, «10»-й класс). Применяется для понятного пользователю отображения информации о значении номера класса.

Методы, реализуемые классом `check_class`:

- метод `private UI::check_class` – создает экземпляры виджетов. Компилятор в свою очередь получает информацию из формы `.UI` и генерирует код создания виджета в сгенерированных исходных файлах;
- метод `public getLogin()` – получает значение `login-a`. Используется как раз таки для передачи значения переменной в другие окна приложения;
- метод `public setLogin()` – устанавливает определенное значение `login-a`. Первоначально устанавливается значение, переданное предыдущим окном, однако спокойно может быть изменено разработчиком;
- метод `public getClass_letter()` – получает значение буквенного обозначения учебного класса. Используется при составлении SQL-запросов к базе данных PostgreSQL, выборе одного из параметров для дальнейшей обработки приложением;

– метод `public setClass_letter()` – устанавливает буквенное значение `class_letter` согласно введенному пользователем в поле `class_letter_label`, позже измененное из строчного состояния до прописного;

– метод `public getClass_num()` – получает значение целочисленного обозначения учебного класса. Используется при составлении SQL-запросов к базе данных PostgreSQL, выборе одного из параметров для дальнейшей обработки приложением;

– метод `public setClass_num()` – устанавливает целочисленное значение `class_letter` согласно введенному пользователем в поле `class_num_label`.

Слоты, реализуемые классом `check_class`:

– слот `void on_cancel_button_clicked()` – приватный. Предназначен для возврата к блоку выбора объема отображаемой информации, представленного в программе классом `get_data`, в случае, например, ошибочного выбора (нажатия) в предыдущем блоке. Реализуется после получения сигнала «`clicked`» («нажат») от клиента. После его реализации происходит перенаправление пользователя к предыдущему окну с передачей в него идентификационных данных в виде фамилии и имени (`login`);

– слот `void on_find_button_clicked()` – приватный. Предназначен для реализации основной логики класса `check_class`. Интуитивно понятный интерфейс пользователя, представленный данным классом, позволяет ввести данные о каком именно классе пользователю необходимо получить информацию. При отсутствии ввода, будет выведена информация о всех существующих классах. Реализуется после получения сигнала «`clicked`» («нажат») от клиента. После его реализации происходит перенаправление пользователя к следующему окну с передачей в него идентификационных данных: фамилия и имя (`login`), номер класса (`class_num`) и буквенное обозначение класса (`class_letter`).

3.9 Блок отображения информации о конкретном учащемся для роли «родитель»

Этот блок выполняет функцию визуализации искомой информации, доступной только пользователям с ролью «родитель», и представлен в программе одним классом: `parent_window`. Данный класс состоит из следующих полей:

– `query: QSqlQuery` – используется для выполнения SQL запросов к базе данных. Применительно к разработанному приложению используется совместно с методами `exec()` и `next()`;

– `query_getnotes_full: QSqlQuery` – используется для выполнения нового SQL запроса во время выполнения предыдущего. В программе этот

запрос реализован во время «получения» отметок «учащегося» и комментариев к этим отметкам (параллельно);

- `question_to_db: QString` – строка, необходимая для формирования запросов к SQL базе данных. Применяется в качестве передаваемого параметра для метода `exec()`;

- `full_name_parent: QString` – элемент, содержащий полные фамилию и имя авторизованного пользователя. Необходим для формирования запросов к базе данных, с целью последующего поиска «учащегося», который связан с авторизованным пользователем;

- `parent_id: QString` – строка, содержащая идентификационный номер «родителя». Необходим данный элемент при формировании запросов, по поиску информации об «учащемся». Изменяется значения, после запроса к базе данных по соответствию `parent_id` к `full_name_parent`;

- `full_name_st: QString` – строка, хранящая в себе полное имя «учащегося». Используется для получения отображения имени «учащегося» в понятной, для конечного пользователя, форме;

- `studying_group_id: QString` – элемент, содержащий идентификационный номер учебной группы. Используется при составлении SQL-запроса к базе данных, для получения в дальнейшем понятного пользователю номера класса;

- `student_id: QString` – строка, содержащая идентификационный номер «учащегося», сопоставленный с его полным именем. Используется при написании всех запросов к базе данных;

- `gr_num: QString` – переменная, содержащая в себе информацию о числовом значении ступени образования (например, «10»-й класс). Применяется для понятного пользователю отображения информации о значении номера класса;

- `gr_prof: QString` – строка, хранящая информацию о буквенном обозначении класса, в котором обучается соответствующий «учащийся». Применяется для понятного пользователю отображения информации о численном значении номера класса;

- `note_id: QString` – идентификационный номер, соответствующий заметке о выбранном «учащемся». Данная переменная обновляется во время цикличного запроса в базе данных;

- `marks: QString` – элемент содержащий значение отметки, выставленной соответствующему «учащемуся». Также используется для хранения средних отметок по предметам. Отображается данная переменная на экран посредством метода `append()` класса `QTextEdit` в соответствующие поля на экране пользователя;

- `subject: QString` – строка, хранящая полное название предмета. Используется одновременно с переменной `marks` при отображении отметок «учащегося». Изменяется значение данной переменной циклично, вместе с

переменной `marks` (так как они существуют в одном цикле одного запроса к базе данных).

Методы, реализуемые классом `parent_window`:

- метод `private UI:: parent_window` – создает экземпляры виджетов. Компилятор в свою очередь получает информацию из формы `.UI` и генерирует код создания виджета в сгенерированных исходных файлах;

- метод `public getFull_name_parent()` – получает значение `full_name_parent-a` (имя авторизованного пользователя). Используется при составлении SQL-запросов внутри класса;

- метод `public setFull_name_parent ()` – устанавливает определенное значение `full_name_parent-a`. Данное значение не изменяется, но существует в программе, так как язык C++ поддерживает инкапсуляцию и создание одновременно `get()` и `set()` методов является «хорошим тоном»

Слоты, реализуемые классом `parent_window`:

- слот `void on_pushButton_cancel_clicked()` – приватный. Предназначен для интуитивно понятного закрытия приложения конечным пользователем. Реализуется после получения сигнала «clicked» («нажат») от клиента. После его реализации происходит вызов метода `quit()` класса `QApplication` который «уничтожает» приложение на уровне ядра;

- слот `void on_pushButton_info_clicked ()` – приватный. Создан для отображения контактной информации, об учреждении образования, с помощью которой конечный пользователь может получить дополнительную информацию. Реализуется после получения сигнала «clicked» («нажат») от клиента. После его реализации происходит вызов метода `setText()` стандартного для фреймворка класс `QMessageBox`. Передаваемым параметром для метода `setText()` как раз таки и будет контактная информация. Через промежуток времени, отсчитанный таймером, встроенным в фреймворк `QT Creator`, `QTimer`, пользователь вернется к блоку, представленному классом `parent_window`.

3.10 Блок отображения информации о конкретном учащемся для роли «учащийся»

Данный блок является самым ограниченным по своему функционалу, так как роли «учащийся» предоставляется меньше всего возможностей. Представлен данный блок в программе одним классом: `student_window`. Данный класс состоит из следующих полей:

- `query: QSqlQuery` – используется для выполнения SQL запросов к базе данных. Применительно к разработанному приложению используется совместно с методами `exec()` и `next()`;

- `question_to_db: QString` – строка, необходимая для формирования запросов к SQL базе данных. Применяется в качестве передаваемого

параметра для встроенного метода `exec()`;

- `marks: QSqlQuery` – используется для хранения значения отметки, полученного в результате SQL-запроса к базе данных PostgreSQL. Также, значение данной переменной выводится на экран с помощью использования метода `append()` встроенного класса `QTextEdit` в соответствующие поля на экране пользователя;

- `subject: QSqlQuery` – хранит название учебного предмета. Значение данной переменной изменяется после осуществления SQL-запроса к базе данных, который возвращает значения `marks` и `subject`. Отображается на экране также благодаря методу `append()` встроенного класса `QTextEdit` в то же поле, в котором отображается и значение переменной `marks`;

- `student_id: QSqlQuery` – строка, содержащая идентификационный номер «учащегося», сопоставленный с его полным именем. Используется при реализации запросов к базе данных, для получения конкретной информации об учащемся;

- `full_name_st: QSqlQuery` – переменная используемая для хранения полного имени (фамилии и имени) учащегося, информацию о котором и хочет получить пользователь.

Методы, реализуемые классом `student_window`:

- метод `private UI:: student_window` – создает экземпляры виджетов. Компилятор в свою очередь получает информацию из формы `.UI` и генерирует код создания виджета в сгенерированных исходных файлах;

- метод `public getFull_name_st()` – получает значение `full_name_st`. Используется при составлении SQL-запросов внутри класса, для получения информации о конкретном «учащемся»;

- метод `public setFull_name_st()` – устанавливает определенное значение `full_name_st`. Данный метод в программе не используется, но создается, так как в классическом представлении инкапсуляции необходимо создавать его вместе с `getter`-ами;

Слоты, реализуемые классом `student_window`:

- слот `void on_pushButton_clicked()` – приватный. Является единственным слотом, реализованным в данном классе. Предназначен для интуитивно понятного закрытия приложения конечным пользователем. Реализуется после получения сигнала «`clicked`» («нажат») от клиента. После его реализации происходит вызов метода `quit()` класса `QApplication` который «уничтожает» приложение на уровне ядра.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Листинг программы представлен в Приложении А.

4.1 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.

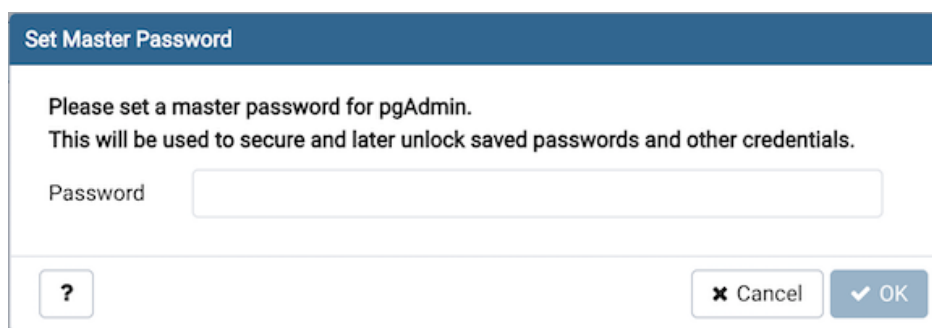


Рисунок 4.1 - Окно «Set Master Password» в pgAdmin 4

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

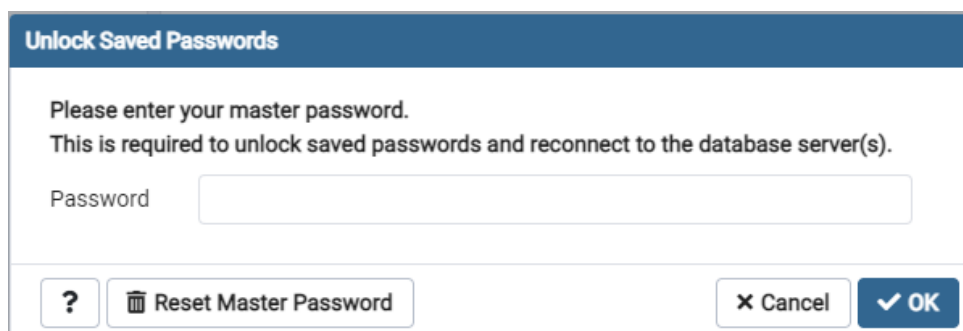


Рисунок 4.2 - Окно «Unlock Saved Passwords» в pgAdmin 4

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 создания новой базы данных.

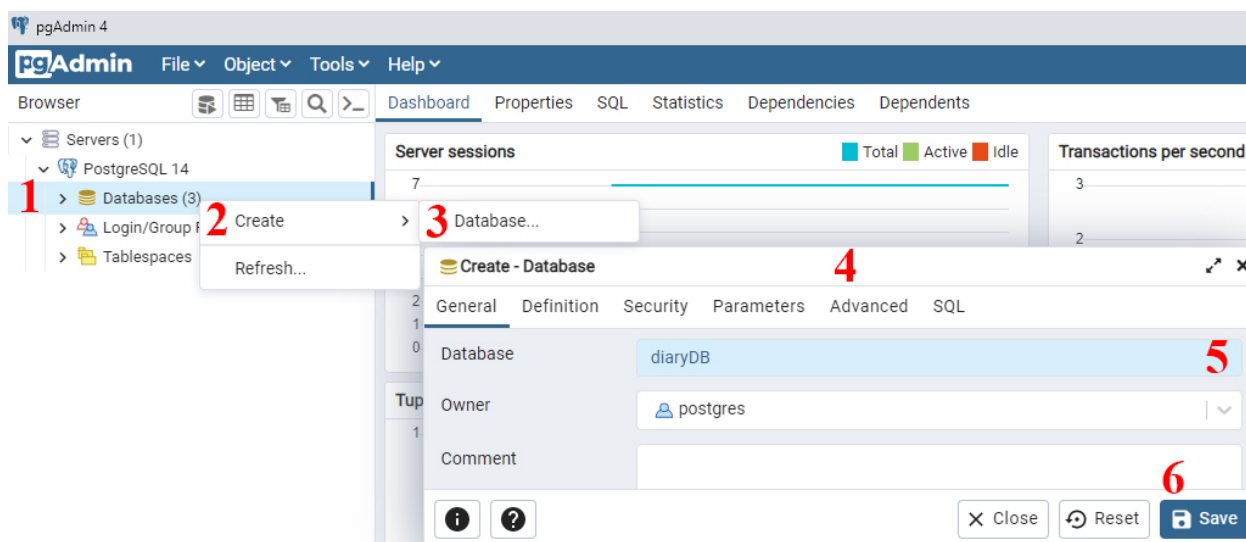


Рисунок 4.3 - Создание новой базы данных посредством pgAdmin 4

4.2 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.

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 username, passwd from pg_shadow;
```

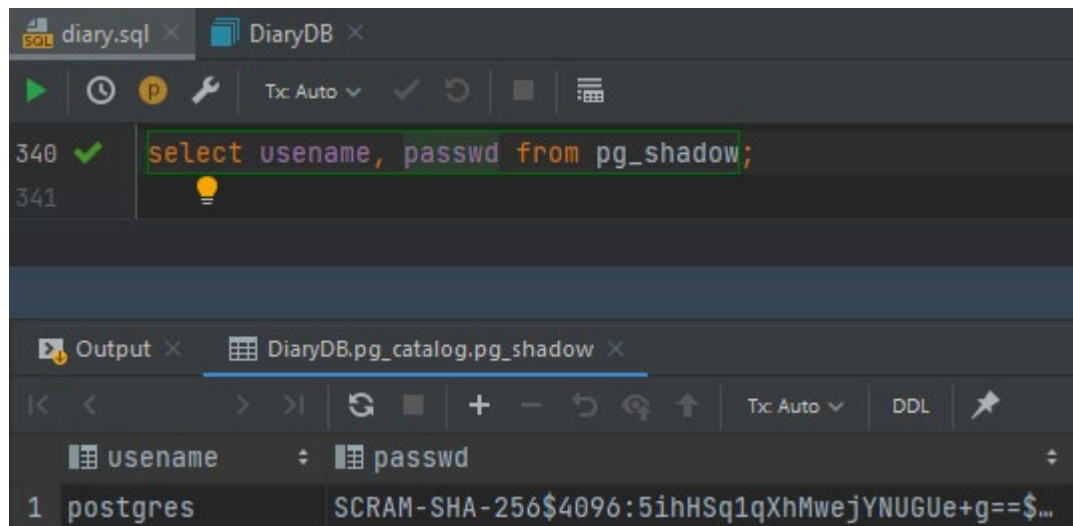



Рисунок 4.4 - Первоначальные пользователи базы данных PostgreSQL

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;
```

Также создается пользователь teacher с правом на редактирование записей.

```
CREATE USER teacher WITH ENCRYPTED PASSWORD
'teacherDBpassword';
GRANT USAGE ON SCHEMA public to teacher;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON
TABLES TO teacher;
```

После выполнения данных запросов можно проверить их выполнение уже описанным ранее запросом (рисунок 4.5).

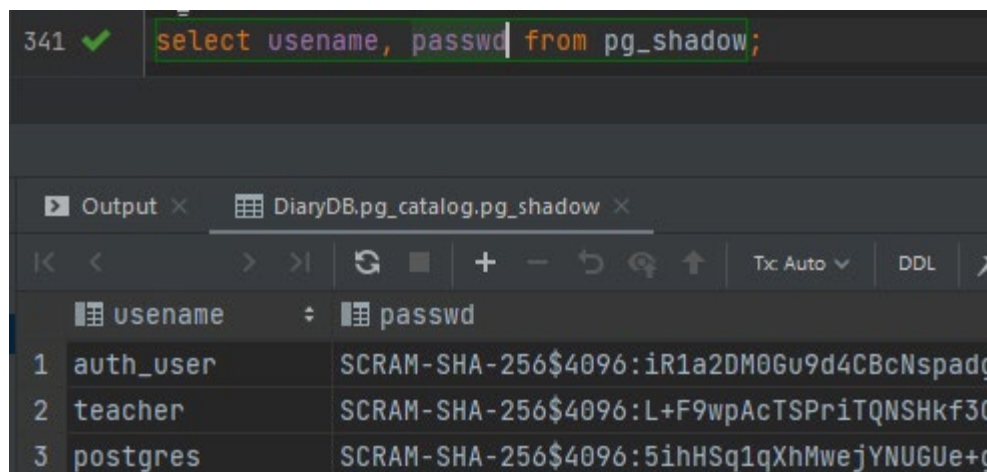


Рисунок 4.5 - Все пользователи базы данных PostgreSQL

4.3 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 обновлений в базу данных, что может привести к утрате актуальных данных.

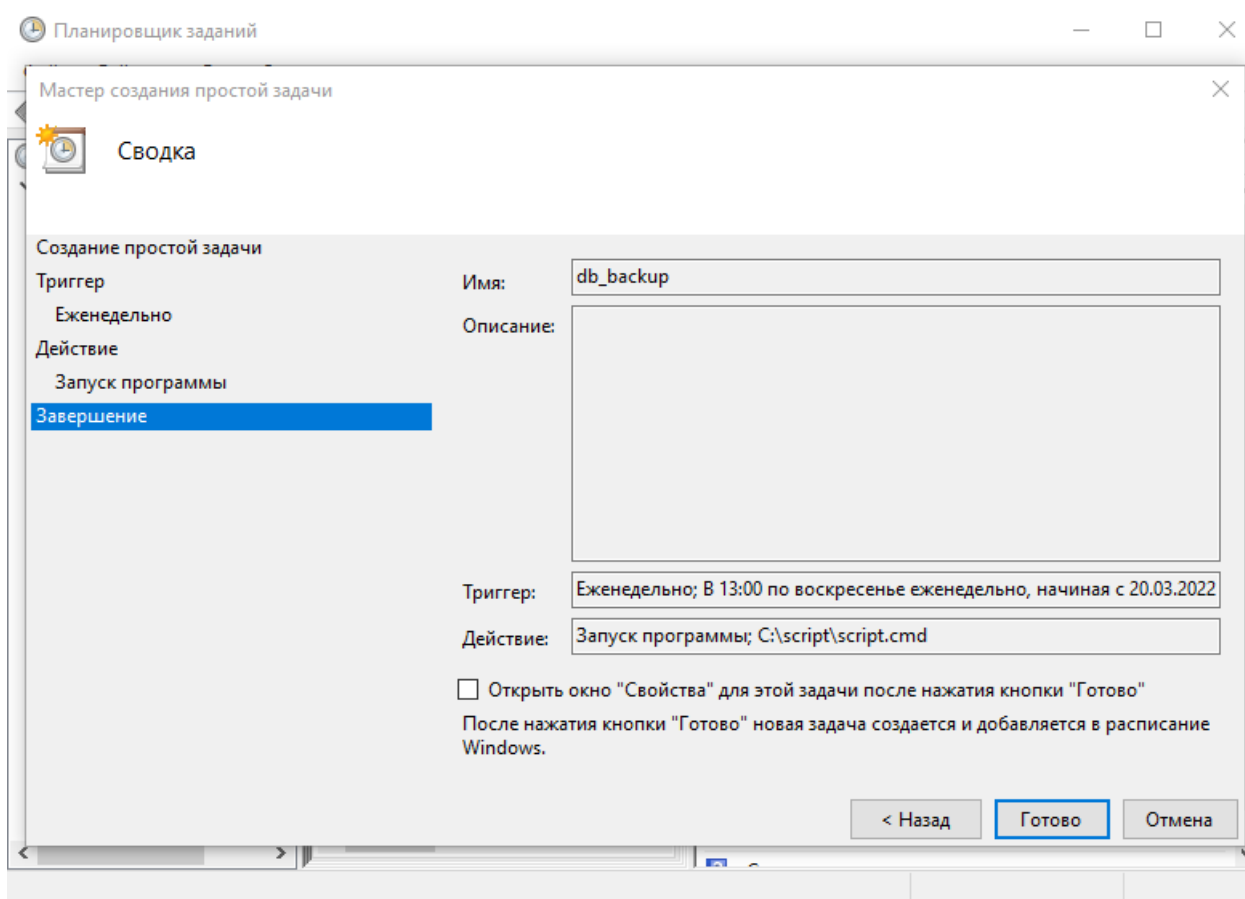


Рисунок 4.6 - Создание задачи по резервному копированию базы данных

4.4 Создание таблиц и связей базы данных PostgreSQL с использованием Oracle SQL Developer Data Modeler

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

формата «DLL» (Dynamic Link Library — «библиотека динамической компоновки»). 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
);

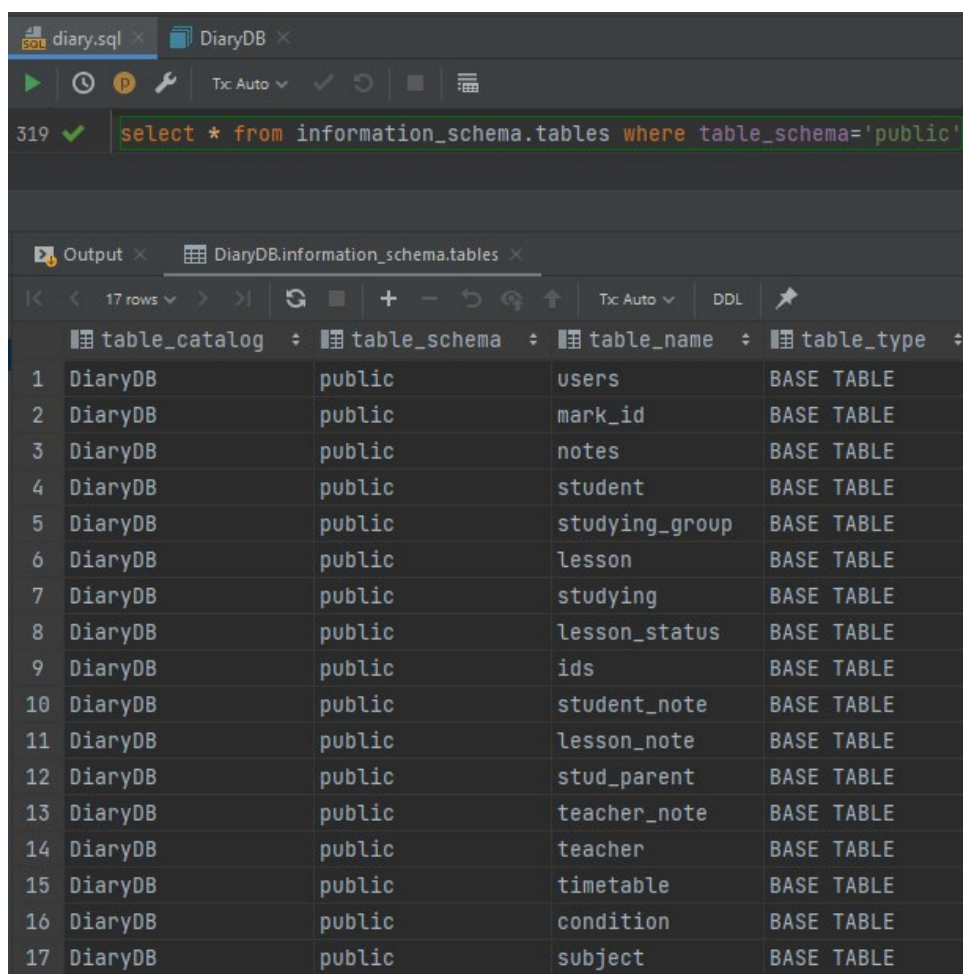
ALTER TABLE student ADD CONSTRAINT student_pk PRIMARY KEY (
id );
ALTER TABLE ids
    ADD CONSTRAINT ids_student_fk FOREIGN KEY ( student_id )
        REFERENCES student ( id );

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. Отображение оформлено с использованием консоли приложения DataGrip и выполнением вышеописанного запроса.



	table_catalog	table_schema	table_name	table_type
1	DiaryDB	public	users	BASE TABLE
2	DiaryDB	public	mark_id	BASE TABLE
3	DiaryDB	public	notes	BASE TABLE
4	DiaryDB	public	student	BASE TABLE
5	DiaryDB	public	studying_group	BASE TABLE
6	DiaryDB	public	lesson	BASE TABLE
7	DiaryDB	public	studying	BASE TABLE
8	DiaryDB	public	lesson_status	BASE TABLE
9	DiaryDB	public	ids	BASE TABLE
10	DiaryDB	public	student_note	BASE TABLE
11	DiaryDB	public	lesson_note	BASE TABLE
12	DiaryDB	public	stud_parent	BASE TABLE
13	DiaryDB	public	teacher_note	BASE TABLE
14	DiaryDB	public	teacher	BASE TABLE
15	DiaryDB	public	timetable	BASE TABLE
16	DiaryDB	public	condition	BASE TABLE
17	DiaryDB	public	subject	BASE TABLE

Рисунок 4.7 - Результат проверки создания таблиц в базе данных

4.5 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.



Доступ к вашим данным на Яндексе

Зарегистрируйте свое приложение на OAuth-сервере. После этого оно сможет запрашивать у пользователя разрешение на доступ к его данным, хранящимся на серверах Яндекса.

Список зарегистрированных приложений

Приложение	Права
 Diary	Яндекс.Диск REST API Запись в любом месте на Диске Чтение всего Диска Доступ к информации о Диске Доступ к папке приложения на Диске API Яндекс ID Доступ к логину, имени и фамилии, полу Доступ к адресу электронной почты Яндекс.Почта Отправка писем через Яндекс.Почту по протоколу SMTP

[Зарегистрировать новое приложение](#)

Рисунок 4.8 – Зарегистрированные приложения 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 приложения указывается после знака «=» (равно) в ссылке.

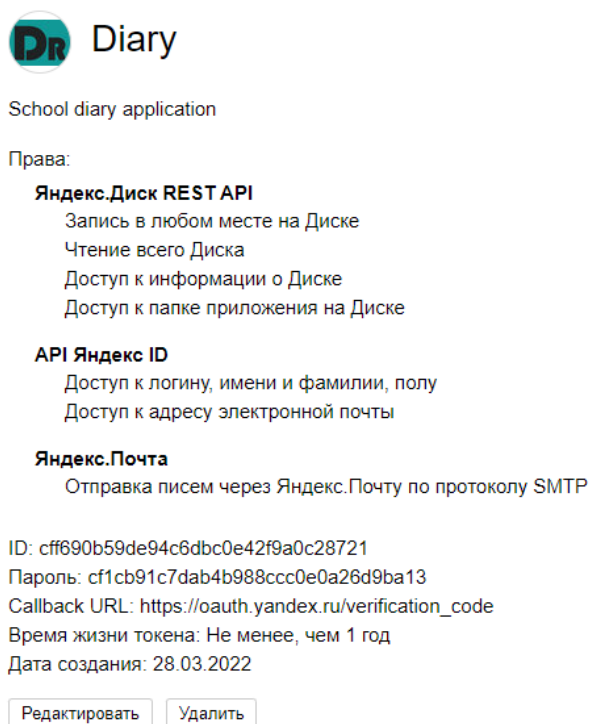


Рисунок 4.9 – Приложение в API Яндекс ID

4.6 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 файлов в исходную папку QT. При наличии всех файлов остается лишь подключиться к базе данных, используя 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 пользователя после авторизации.

На рисунке 4.10 как раз-таки и предоставлено сообщение об прерывании «старого» соединения к базе данных (предпоследнее).

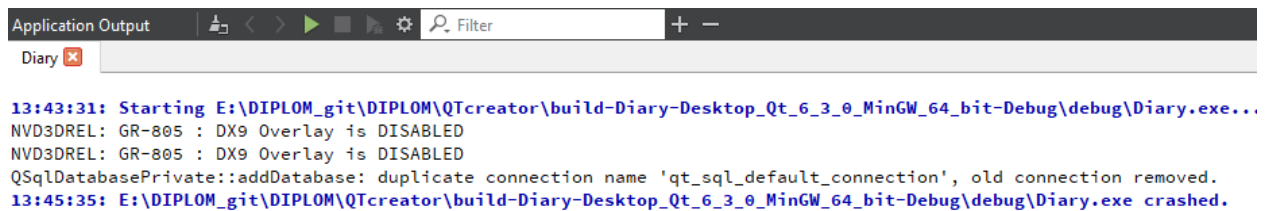


Рисунок 4.10 – Пример логирования приложения при изменении роли пользователя базы данных

Сообщения «NVD3DREL: GR-805 : DX9 Overlay is DISABLED» говорят об ошибке драйвера Nvidia, в связи с использованием более актуальной версии, что не влияет на работоспособность приложения.

```

    QSqlDatabase db = QSqlDatabase::addDatabase("QPSQL");
    db.setDatabaseName("DiaryDB");
    db.setUserName(role);
    db.setPassword(password);
    if (!db.open()) {
        QMessageBox::warning(0, "DataBase Error", "Something wrong
with DB");
        return false;
    }
    else{
        return true;
    }
}

```

Также в данном файле представлен метод `checkConnection()`. Он необходим для проверки подключения к базе данных и используется в программе во всех классах, где будут производиться любые действия с данными (чтение, запись). Его логика основана на осуществлении запроса к базе данных, который в случае успеха вернет значение «true» и в случае неудачи «false». Запрос ищет в таблице пользователей имя «postgres», которое в этой базе данных существует. Если оно не будет найдено, то значит база данных не подключена или имеет ошибки.

```

    QSqlQuery query;
    QString question_to_db = "null";

    question_to_db = "select username from pg_shadow where username
like 'postgres'";
    query.exec(question_to_db);
    if (!query.next()) {
        QMessageBox::warning(0, "DataBase Error", "Something
wrong with DB");
        return false;
    }else{

```



```
        return true;
    }
}
```

4.7 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].

4.8 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.

4.9 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. В методе `setFixedSize()` первый параметр указывает на ширину, а второй на высоту окна (размеры в пикселях).

```
setWindowFlags(Qt::Dialog);
setFixedSize(545, 445);
```

4.10 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();
    }

```

4.11 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();
```

Для проверки выставленной отметки (принадлежит ли она промежутку от «1» до «10») предусмотрена проверка, которая в свою очередь гарантирует проверку еще такого фактора, как ввод буквенного значения в данное поле. В случае обнаружения ошибки – на экране пользователя появится соответствующее сообщение.

```
int mark_toCheck = 0;
mark_toCheck = mark.toInt(0, 10);

if (mark_toCheck > 10 || mark_toCheck <= 0){
    QMessageBox::warning(this, "Mark error", "You can put
only integer marks in range: 1..10");
}
```

В случае прохождения данной проверки, перед дальнейшим вводом информации на экране пользователя появится окно с вопросом, уверен ли он в своих действиях. Если ответ пользователя – «да», то программа продолжит свое выполнение. Если же ответ – «нет», то все действия будут прерваны.

```
QMessageBox::StandardButton reply;
reply = QMessageBox::question(this, "Are you sure?", "Are you
sure, that you wanna do this?",

QMessageBox::Yes|QMessageBox::No);
if (reply == QMessageBox::Yes)
```

Первым шагом перед выставлением отметки является получение идентификационного номера учащегося(`student_id`). Это необходимо для составления запросов в дальнейшем.

```
question_to_db = "select id from student where full_name_st
= '" + getStudent_name() + "'";
query.exec(question_to_db);
query.next();
student_id = query.value(0).toString();
```

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);

```

4.12 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();
            }
        }

```

4.13 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_num_parent;
        }else{
            infoText = "\\n=====\\n"
            "\\nFull name: " + full_name + "\\nBirth data: " + birth_date +
            "\\nParent full name: " + full_name_parent + "\\nParent phone number:
            " + phone_num_parent;

        }
        ui->textEdit->append(infoText);
    
```

4.14 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 отсутствия «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!");
}
}

```

4.15 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_num_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_num_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
c.lesson_status_id) 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 round(avg(mark), 2), 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);

```

4.16 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();
}

```

4.17 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
c.lesson_status_id) 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()`.

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

В данном разделе приводится программа и методика испытаний, разработанного клиент-серверного приложения «Электронный журнал». При разработке данного приложения учитывались следующие минимальные системные требования:

Таблица 5.1 – Минимальные системные требования

Операционная система	Windows 7/10 64 bit
Объем ОЗУ	1 ГБ
Видеокарта	Nvidia 760
Разрешение экрана	1024 x 768
Свободного места на жестком диске	100 МБ

Оптимальными же требованиями к разработанному приложению будут считаться значения, соответствующие персональному компьютеру, на котором производилась разработка программного обеспечения.

Таблица 5.2 – Рекомендуемые системные требования

Операционная система	Windows 10 64 bit
Объем ОЗУ	8 ГБ
Видеокарта	Nvidia 1650
Разрешение экрана	3840 x 2160
Свободного места на жестком диске	1 ГБ

Проверка работоспособности приложения будет состоять из следующих шагов:

- проверка работоспособности API Яндекс ID;
- проверка сборки приложения на наличие ошибок;
- проверка качества отображаемого интерфейса;
- проверка подключения к базе данных и правильности ее работы;
- проверка целостности сохраняемых данных;
- проверка корректности вводимых данных;

5.1 Проверка работоспособности API Яндекс ID

Самой главной частью любого приложения с личными данными является безопасность. Для обеспечения безопасности необходимы специальные механизмы. В случае разработанного приложения использовался API Яндекс ID. Он предоставляет механизм, позволяющий авторизовываться в приложении через свои внутренние механизмы, использующие в свою очередь протокол авторизации OAuth 2.0, который является наиболее актуальным на сегодняшний день.

Таблица 5.3 – Тесты, для проверки работоспособности API Яндекс ID

Номер теста	Содержание теста	Ожидаемый результат	Тест пройден
1.	Авторизация в приложении с помощью аккаунта №1.	Пользователю необходимо ввести логин и пароль, либо авторизоваться с помощью других поддерживаемых сервисов. После чего произойдет выдача сервисом токена №1.	Да.
2.	Авторизация в приложении с помощью аккаунта №2.	Пользователю необходимо ввести логин и пароль, либо авторизоваться с помощью других поддерживаемых сервисов. После чего произойдет выдача сервисом токена №2, отличного от токена, выданного в тесте №1.	Да.
3.	Повторная авторизация с использованием аккаунта №2.	Пользователю будет предложена упрощенная схема авторизации, с учетом прошлого входа, где нужно подтвердить лишь личность.	Да.
4.	Попытка авторизации через несуществующий аккаунт.	Провал авторизации. Соответствующее сообщение на экране пользователя.	Да.

Для проверки работоспособности данного API используется несколько аккаунтов, зарегистрированных в Яндекс.

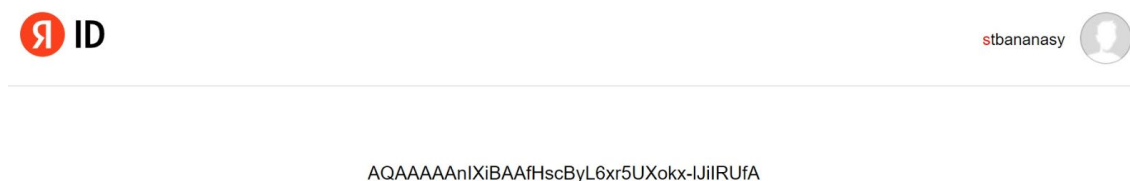


Рисунок 5.1 – Результат теста №1

Запомним результат выполнения теста №1. Это токен со следующим значением: «AQAAAAAnIXiBAAfHscByL6xr5UXokx-IJiRUfA»

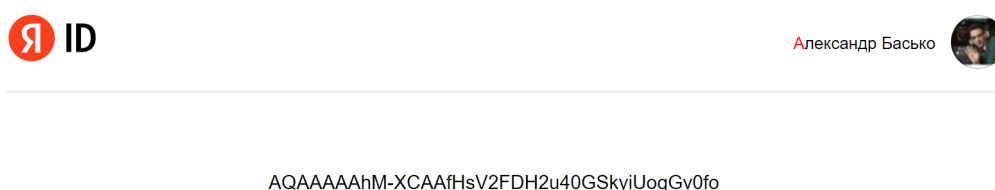


Рисунок 5.2 – Результат теста №2

Результатом выполнения теста №2 является токен со значением: «AQAAAAAhM-XCAAfHsV2FDH2u40GSkyiUogGv0fo». Сопоставив значения первого и второго токена, мы увидим явное несоответствие, что говорит о том, что для каждого авторизованного пользователя они отличны.

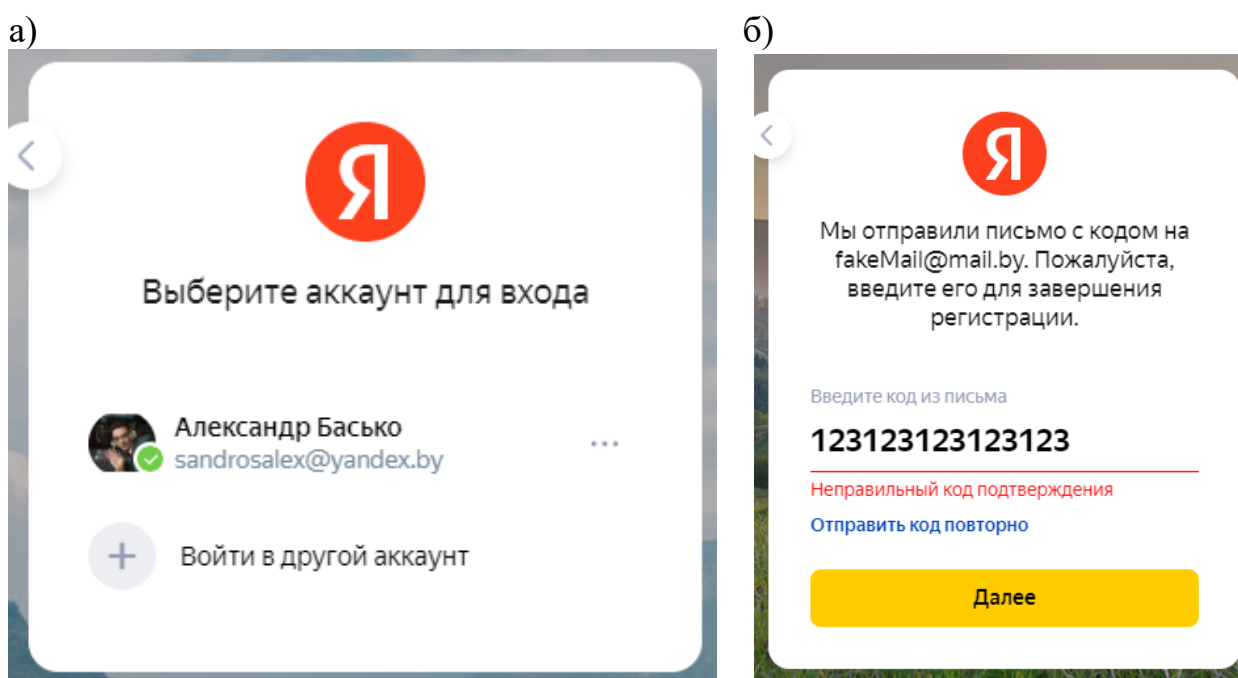


Рисунок 5.3 – Результаты выполнения тестов: а – №3; б – №4

Результаты тестов №3 и №4 видны на рисунке 5.3. В первом случае аккаунт уже известен приложению, и оно предлагает авторизовываться через него. Во втором случае API не предоставляет возможность авторизоваться несуществующему пользователю.

5.2 Проверка сборки приложения на наличие ошибок

Для следующего теста производится сборка разработанного приложения во фреймворке QT Creator.

Таблица 5.4 – Тесты, для проверки сборки приложения на наличие ошибок

Номер теста	Содержание теста	Ожидаемый результат	Тест пройден
6.	Сборка приложения в фреймворке QT Creator.	В консоли фреймворка будут отсутствовать ошибки, коды и т.п.	Да.

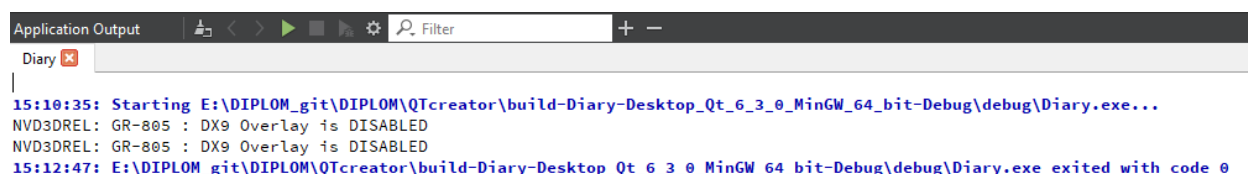


Рисунок 5.5 – Результат выполнения теста №6

В ходе выполнения теста было запущено приложение и выполнялись различные операции в нем на протяжении некоторого времени (2 минуты, исходя из таймера в логах QT Creator-a). В консоли появились сообщения говорящие об ошибке драйвера Nvidia в связи с использованием более актуальной версии видеокарты и драйвера к ней, что никак не влияет на работоспособность приложения. В связи с этим тест будет считаться пройденным.

5.3 Проверка качества отображаемого интерфейса

Данный тест проводится с целью проверки пользовательского интерфейса. Основными вопросами при его проведении являются вопросы:

- понятности иконок приложения, кнопок;
- читабельность основного текста, а также текстов информационных сообщений;
- шрифт, используемый в приложении.

Таблица 5.5 – Тесты, для проверки качества отображаемого интерфейса

Номер теста	Содержание теста	Ожидаемый результат	Тест пройден
7.	Проверка качества отображения интерфейса: отображение окон по центру экрана, удобность в чтении цветовой гаммы.	Все окна расположены по центру экрана, зафиксированы по размеру. Цветовая гамма читабельна (белый текст на темном фоне).	Да.

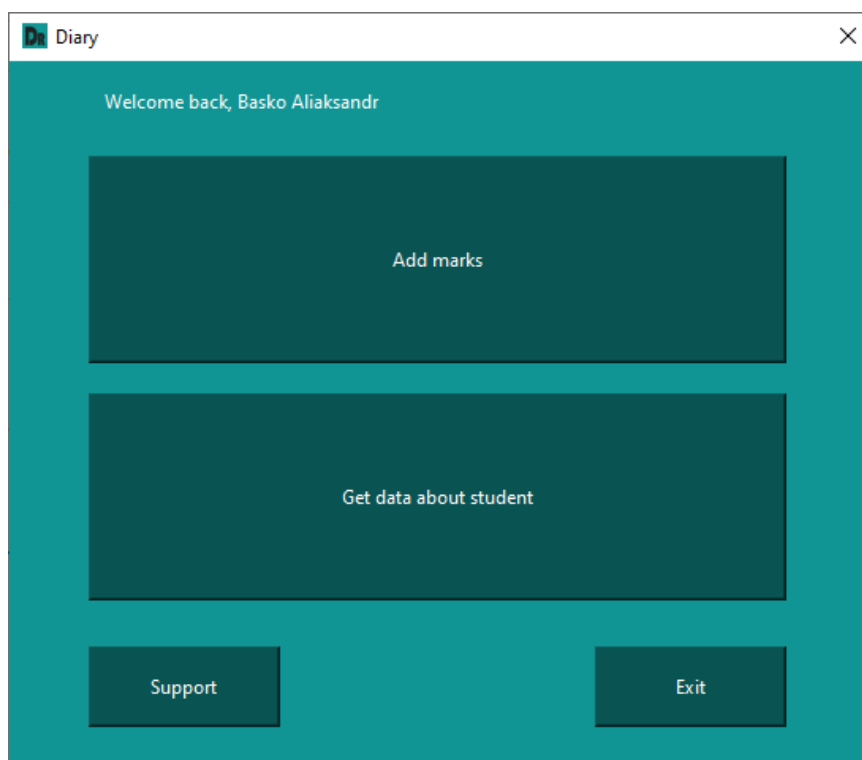


Рисунок 5.6 – Результат выполнения теста №7

5.4 Проверка подключения к базе данных и правильности ее работы

База данных – один из основных элементов разработанного приложения. Для функционирования проекта, в нем реализована проверка на наличие подключения приложения к базе данных, которая используется во всех функциях и методах, взаимодействующие в том или ином виде с базой данных (будь то чтение или запись в базу данных).

Таблица 5.6 – Тесты, для проверки подключения к базе данных и правильности ее работы

Номер теста	Содержание теста	Ожидаемый результат	Тест пройден
8.	Попытка авторизации в приложении со включенной базой данных.	Отсутствие каких-либо сторонних звуков, сообщений и т.п.	Да.
9.	Попытка авторизации в приложении с выключенной базой данных.	Звук системной ошибки Windows, сообщение об отсутствии подключения к базе данных.	Да.
10.	Попытка выставления отметки при включенной базе данных	Отсутствие каких-либо сторонних звуков, сообщений и т.п.	Да.
11.	Попытка выставления отметки при выключенной базе данных	Звук системной ошибки Windows, сообщение об отсутствии подключения к базе данных.	Да.
12.	Попытка просмотра информации о конкретном «ученике» при включенной базе данных	Отсутствие каких-либо сторонних звуков, сообщений и т.п.	Да.
13.	Попытка просмотра информации о конкретном «ученике» при выключенной базе данных	Звук системной ошибки Windows, сообщение об отсутствии подключения к базе данных.	Да.

Для отключения базы данных использовалась командная строка Windows, в которую вводилась команда «net stop», используется для остановки системных служб операционной системы Windows. Полная команда: «net stop postgresql-x64-14».

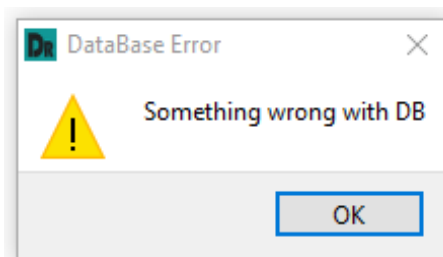


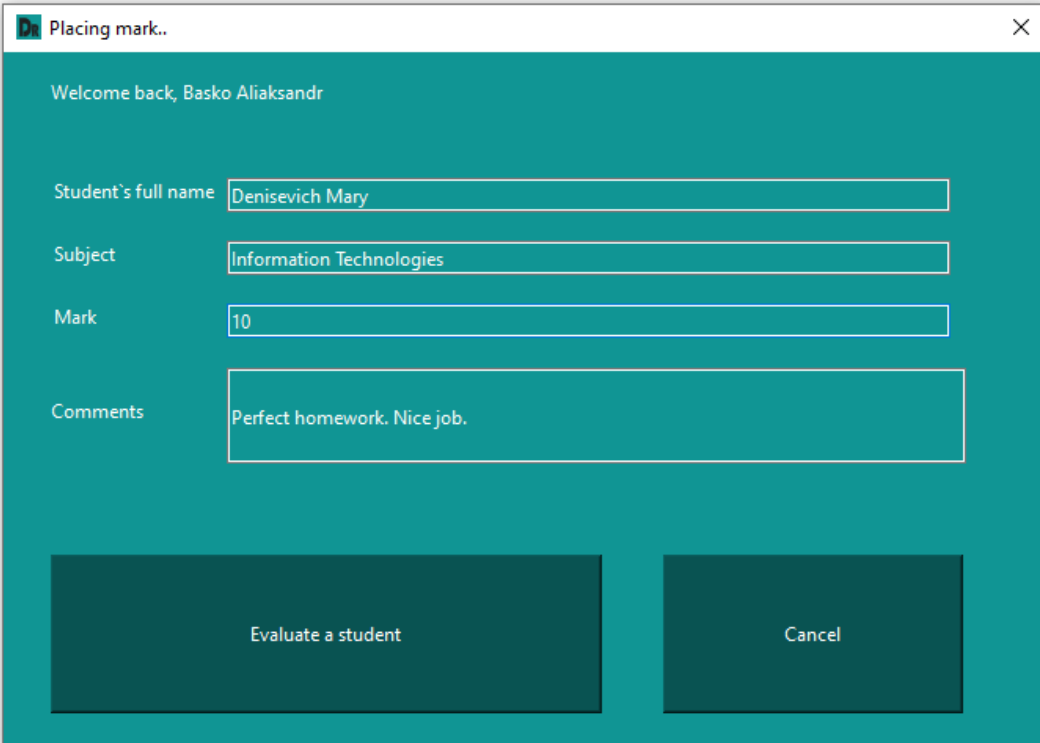
Рисунок 5.7 – Результат выполнения тестов №9, №11, №13

5.5 Проверка целостности сохраняемых данных

Таблица 5.7 – Тесты, для проверки целостности сохраняемых данных

Номер теста	Содержание теста	Ожидаемый результат	Тест пройден
14.	Добавление отметки «учащегося».	Отображение выставленной отметки в окне отображения информации об «учащемся»	Да.
15.	Добавление комментария к отметке «учащегося».	Отображение недавно добавленного «ученику» комментария в окне отображения информации об «учащемся».	Да.

Данные тесты производились для «учащегося» с именем «Denisevich Mary». Это связано с тем, что у данного «учащегося», на момент тестирования приложения, записан лишь один комментарий, что позволит больше визуализировать изменения в работе приложения.



Placing mark..

Welcome back, Basko Aliaksandr

Student's full name: Denisevich Mary

Subject: Information Technologies

Mark: 10

Comments: Perfect homework. Nice job.

Evaluate a student Cancel

Рисунок 5.7 – Процесс выполнения тестов: №14, №15

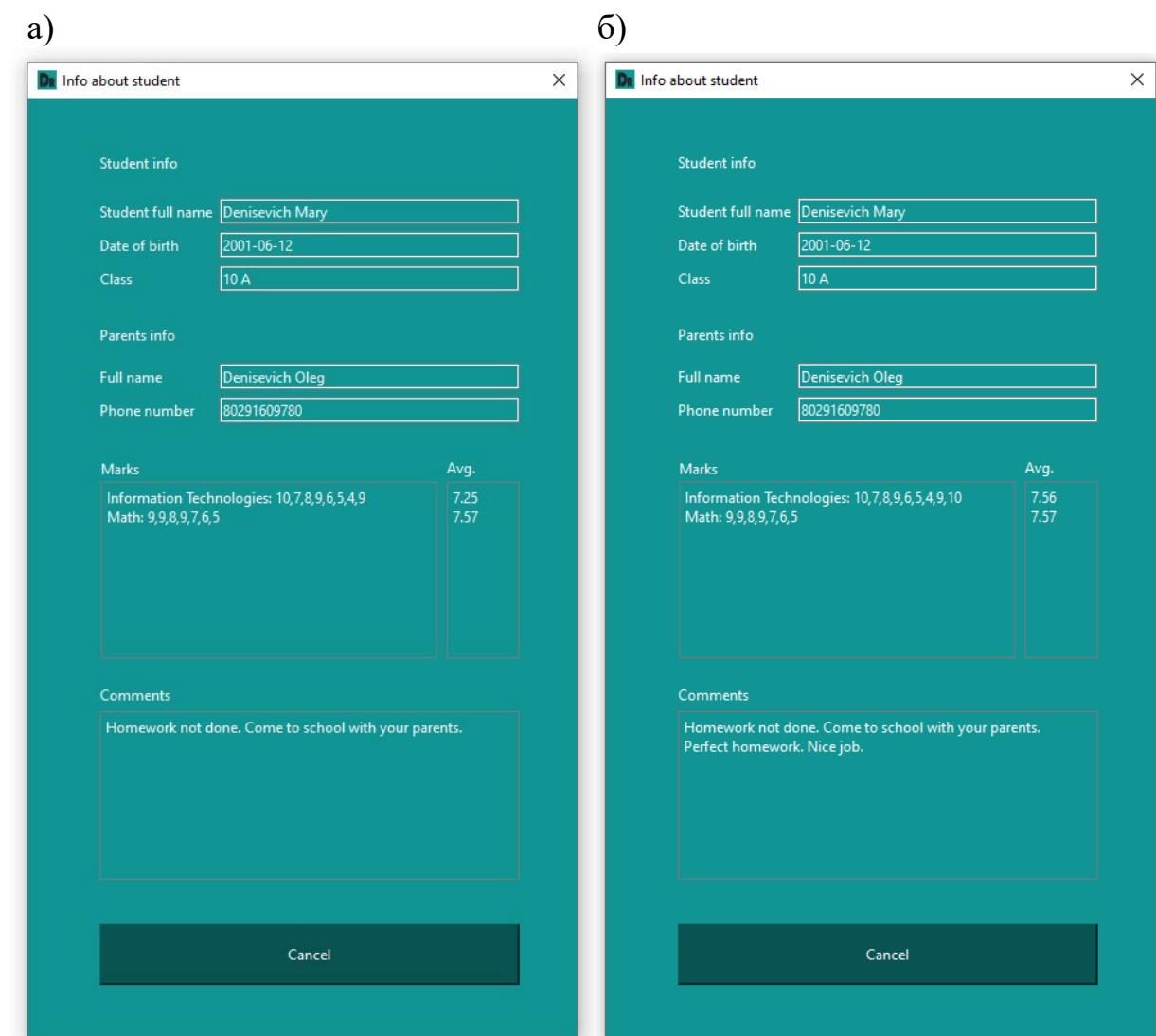


Рисунок 5.7 – Результаты выполнения тестов №14, №15:
а – до выполнения тестов; б – после выполнения тестов

5.6 Проверка корректности вводимых данных

Также важным является проверка корректности вводимых данных. Например, чтобы пользователь, авторизованный под ролью «учитель» не мог выставить отметки учащимся, которые не изучают данный предмет, а отметки были целочисленными и отвечали диапазону выставляемых отметок (от 1 до 10).

Таблица 5.8 – Тесты, для проверки корректности вводимых данных

Номер теста	Содержание теста	Ожидаемый результат	Тест пройден
1	2	3	4

Продолжение таблицы 5.8

1	2	3	4
16.	Корректный ввод токена	Отсутствие каких-либо сторонних звуков, сообщений и т.п.	Да.
17.	Корректный ввод полного имени «учащегося», для выставления отметки	Отсутствие каких-либо сторонних звуков, сообщений и т.п.	Да.
18.	Ввод токена, не предоставленного в базе данных (с ошибкой)	Звук системной ошибки Windows, сообщение об отсутствии таких данных в базе данных.	Да.
19.	При выставлении отметки «учащемуся» использование запрещенных символов в графе отметка («-5», «0», «15», «A»)	Звук системной ошибки Windows, сообщение об ошибке: «Отметка должна быть целочисленным значением из диапазона: «1-10»	Да.
20.	Ввод полного имени «учащегося» с ошибкой, при выставлении отметки	Звук системной ошибки Windows, сообщение об отсутствии «учащегося» с таким полным именем в базе данных.	Да.

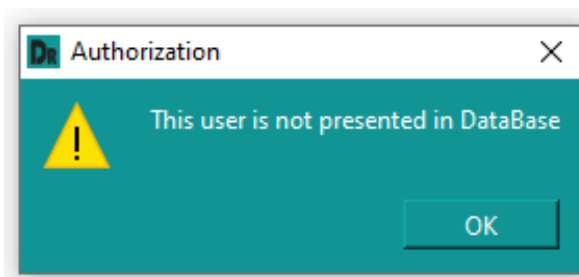
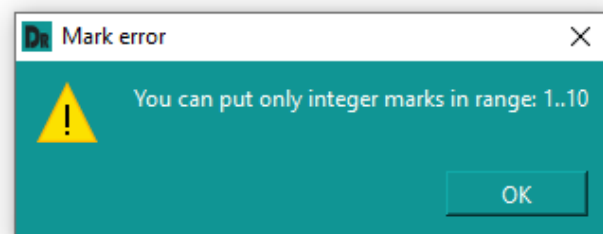


Рисунок 5.8 – Результат выполнения теста №18

Так как тесты №16 и №17 приводят к продолжению работы приложения, без ошибок (дополнительных сообщений), то они не демонстрируются.

a)



б)

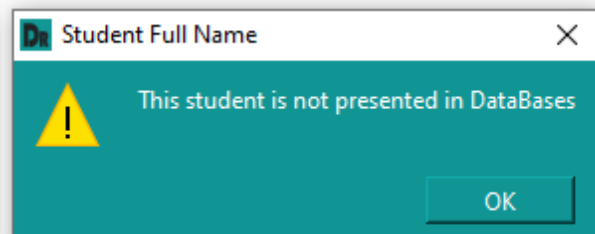


Рисунок 5.9 – Результаты выполнения тестов:
а – №19; б – №20

7 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ МНОГОПОЛЬЗОВАТЕЛЬСКОГО КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ «ЭЛЕКТРОННЫЙ ЖУРНАЛ»

7.1 Характеристика многопользовательского клиент-серверного приложения «электронный журнал», разработанного по индивидуальному заказу

Разрабатываемое в дипломном проекте приложение представляет собой набор модулей, соединенных воедино с интуитивно понятным интерфейсом, предоставляющим возможности классического школьного журнала в современной интерпретации. Данное приложение создается для государственного учреждения «Управление по образованию администрации Октябрьского района г. Минска», которое может распространить данное приложение в школах своего района. Также рассматривается вопрос о возможном масштабировании в пределах Республики Беларусь.

Проблема, которую будет решать проект – отсутствие, на сегодняшний день, в учреждениях общего среднего образования программных продуктов с интуитивно понятным, для конечного пользователя, с учетом любого возраста и уровня образования, интерфейсом, который будет предоставлять широкий спектр функционала (хранение данные о преподавателях, учащихся и их родителях, обработка новых данных (например, добавление отметок, комментариев к отметке) и отображение имеющихся данных любому пользователю в пределах его «прав» доступа) в сфере общего среднего образования. Весь этот функционал будет сосредоточен в одном приложении для персональных компьютеров, что очень актуально на сегодняшний день. Стоит отметить, что на сегодняшний день существует только два аналога данного программного средства: это решения от белорусских компаний под названиями «SCHOOLS.BY» и «Знай.бай» (основные отличия от описываемого программного продукта заключается в том, что аналоги являются веб-приложениями с большой конечной стоимостью для конечного потребителя (не менее 70 белорусских рублей в год), интуитивно понятны только постоянным пользователям, предоставляют полный функционал лишь на коммерческой основе (продажа лицензий)). Это исключает перенасыщение рынка схожими продуктами и оставляет вероятность получения на рынке ведущих позиций в дальнейшем.

Также стоит принять во внимание тот факт, что приложение, разработанное в данном проекте, может и будет подвергаться модификациям, масштабироваться. Разработанные и реализованные модули имеют большой шанс на применение в иных планах организации-разработчика, что приведет к значительному уменьшению трудоемкости работ в других проектах.

7.2 Расчет затрат на разработку клиент-серверного приложения «Электронный журнал»

Исходя из описания проекта можно выделить следующие роли в команде разработчиков:

– Инженер-программист. Человек, специализирующийся в таких технологиях как C+, QT Creator. Этот специалист будет заниматься разработкой основных алгоритмов проекта, внешнего вида и всех оставшихся функциональных составляющих.

– База данных приложения создается, модерируется и настраивается системным архитектором.

– Технический лидер – специалист, уровнем выше, чем инженер-программист, занимающийся решением схожих вопросов, однако на него возлагается вся ответственность перед руководством за сдачу проекта.

Прогнозируемый срок разработки данного программного продукта – два месяца. За этот срок разработчики должны выполнить в полной мере поставленные перед ними задачи и предоставить готовый продукт конечному потребителю.

Расчет затрат на основную заработную плату разработчиков описанного ранее программного продукта детально описан и предоставлен в таблице 7.1.

Таблица 7.1 – Расчет затрат на основную заработную плату

Исполнитель	Количество исполнителей, чел.	Тарифный оклад, руб.	Трудоемкость, мес.	Заработная плата по тарифу, руб.
1. Инженер-программист	1	2400,0	2	4800,0
2. Системный архитектор	1	2600,0	2	5200,0
3. Технический лидер	1	2850,0	2	5700,0
Всего	3	7850,0	2	15700,0
Премия, 33 %				5181,0
Всего основная заработная плата				20881,0

Общая сумма инвестиций (затрат) на разработку программного средства включает в себя основную заработную плату разработчиков, дополнительную заработную плату разработчиков, отчисления на социальные нужды и прочие расходы.

Дополнительная заработная плата разработчиков в размере 15% начисляется при успешном выполнении поставленных целей. Учитывая, что в этом проекте все задачи были выполнены в срок. Рассчитаем дополнительную заработную плату разработчиков по следующей формуле:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (7.1)$$

где $Н_д$ – норматив дополнительной заработной платы (15%).

Отчисления на социальные нужды рассчитываются по формуле:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100}, \quad (7.2)$$

где $Н_{соц}$ – ставка отчислений в ФСЗН (в соответствии с действующим законодательством по состоянию на 01.01.2022 г. – 34,6 %).

Прочие расходы рассчитываются по формуле:

$$Р_{пр} = \frac{З_о \cdot Н_{пр}}{100}, \quad (7.3)$$

где $Н_{пр}$ – норматив прочих расходов (30%).

Общая сумма инвестиций (затрат) на разработку рассчитывается по формуле, представленной ниже:

$$З_p = З_о + З_д + Р_{соц} + Р_{пр}, \quad (7.4)$$

Плановая прибыль, включаемая в цену программного средства, рассчитывается по формуле (7.5):

$$П_{пс} = \frac{З_p \cdot Р_{пс}}{100}, \quad (7.5)$$

где $Р_{пс}$ – рентабельность затрат на разработку программного средства (30%). Отпускная цена программного средства рассчитывается по формуле ниже:

$$Ц_{пс} = З_p + П_{пс}, \quad (7.6)$$

Формирование цены программного средства на основе затрат рассчитано и предоставлено в таблице 7.2.

Таблица 7.2 – Расчет цены программного средства на основе затрат

Наименование статьи затрат	Расчет	Значение, руб.
1	2	3
Основная заработная плата разработчиков	См. табл. 7.1	20881

Продолжение таблицы 7.2

1	2	3
Дополнительная заработная плата разработчиков	$\frac{20881 \cdot 15\%}{100}$	3132,15
Отчисления на социальные нужды	$\frac{(20881 + 3132,15) \cdot 34,6\%}{100}$	8308,55
Прочие расходы	$\frac{20881 \cdot 30\%}{100}$	6264,3
Общая сумма затрат на разработку	20881 + 3132,15 + 8308,55 + 6264,3	38586
Плановая прибыль, включаемая в цену программного средства (рентабельность затрат на разработку программного средства – 30%)	$\frac{38586 \cdot 30\%}{100}$	11575,8
Отпускная цена программного средства	38586 + 11575,8	50161,8

7.3 Расчет результата от разработки клиент-серверного приложения «Электронный журнал»

Для организации-разработчика экономическим эффектом будет считаться прирост чистой прибыли, полученной от разработки, а также реализации, программного продукта заказчику. Учитывая тот факт, что приложение для стационарных компьютеров будет реализовываться организацией-разработчиком согласно отпускной цены, полученной на базе затрат на разработку приложения (см. табл. 7.2), то экономический эффект, организацией-разработчиком, в виде прироста чистой прибыли от его разработки, определится по формуле:

$$\Delta\P_{\text{ч}} = \Pi_{\text{пс}} \left(1 - \frac{H_{\text{п}}}{100}\right), \quad (7.7)$$

где $\Pi_{\text{пс}}$ – прибыль, включаемая в цену программного средства, р;
 $H_{\text{п}}$ – ставка налога на прибыль согласно действующему законодательству, по состоянию на 01.01.2022 г. – 18 %.

Учитывая факт того, что организация-разработчик не является резидентом парка высоких технологий, то от уплаты налога на прибыль разработчик не освобожден. Прирост чистой прибыли, полученной от

разработки и реализации программного средства заказчику, рассчитаем по формуле (7.7):

$$\Delta\Pi_{\text{ч}} = 11575,8 * \left(1 - \frac{18\%}{100}\right) = 9492,16 \text{ руб} \quad (7.8)$$

7.4 Расчет показателей экономической эффективности разработки клиент-серверного приложения «Электронный журнал»

Рассчитаем экономическую эффективность от разработки клиент-серверного приложения «Электронный журнал», согласно персональному заказу, для организации-разработчика.

Для организации-разработчика программного средства оценка экономической эффективности разработки осуществляется с помощью расчета простой нормы прибыли (рентабельности затрат на разработку программного средства) по следующей формуле:

$$P_{\text{и}} = \frac{\Delta\Pi_{\text{ч}}}{Z_{\text{р}}} \cdot 100 \%, \quad (7.9)$$

где $\Delta\Pi_{\text{ч}}$ – прирост чистой прибыли, полученной от разработки программного средства организацией-разработчиком по индивидуальному заказу, р.

$Z_{\text{р}}$ – затраты на разработку программного средства организацией-разработчиком, р.

Рентабельность затрат на разработку программного средства рассчитаем по формуле (7.9):

$$P_{\text{и}} = \frac{9492,16}{38586} \cdot 100 \% \approx 24,6 \% \quad (7.10)$$

Результатом проведения экономического обоснования инвестиций в разработку и реализацию клиент-серверного приложения «Электронный журнал» стали полученные выше значения показателей эффективности:

1. Прирост чистой прибыли, полученной от разработки и реализации программного средства для разработчика, составляет порядка 9492,16 белорусских рублей;

2. Рентабельность затрат на разработку программного средства достигла отметки в 24,6%, что выше ставок по долгосрочным депозитам на момент разработки программного средства.

Как итог, можно сделать заключение, что разработка клиент-серверного приложения «Электронный журнал» является экономически эффективной для организации-разработчика, а реализация проекта с экономической точки зрения целесообразна.

ЗАКЛЮЧЕНИЕ

В период преддипломной практики было разработано приложение на операционной системе Windows 10, с использованием таких технологий как:

- API Яндекс ID;
- Фреймворк QT Creator;
- Язык программирования C++;
- Реляционная база данных PostgreSQL.

По результатам разработки приложения для настольных персональных компьютеров и написания документации к нем можно судить, что поставленные, на этапе проектирования, задачи выполнены в полном объеме и проект может считаться завершенным.

Также не стоит отвергать тот факт, что в проекте заложены идеи, которые способствуют возможному дальнейшему масштабированию, без серьезных усилий со стороны разработчиков:

- база данных базируется на языке SQL, что предоставляет возможность в дальнейшем изменить базу данных на любую другую, описанную на языке SQL;
- язык C++ является объектно-ориентированным, что позволит в будущем дополнять проект новыми модулями без изменения основного кода приложения;
- фреймворк не является устаревшим, а все также актуален на сегодняшний день, что гарантирует отсутствие препятствий для дальнейшей разработки приложения.

Также за период преддипломной практики было проведено экономическое обоснование разработки и реализации дипломного проекта. Его результатом стало полное понимание того, что разработанное приложение является рентабельным. Согласно проведенным расчетам, рентабельность составила почти 25%, что может говорить об экономической целесообразности проекта.