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 идей было создание революционного, в плане внешнего вида, приложения («отойти» от стандартного, привычному 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.

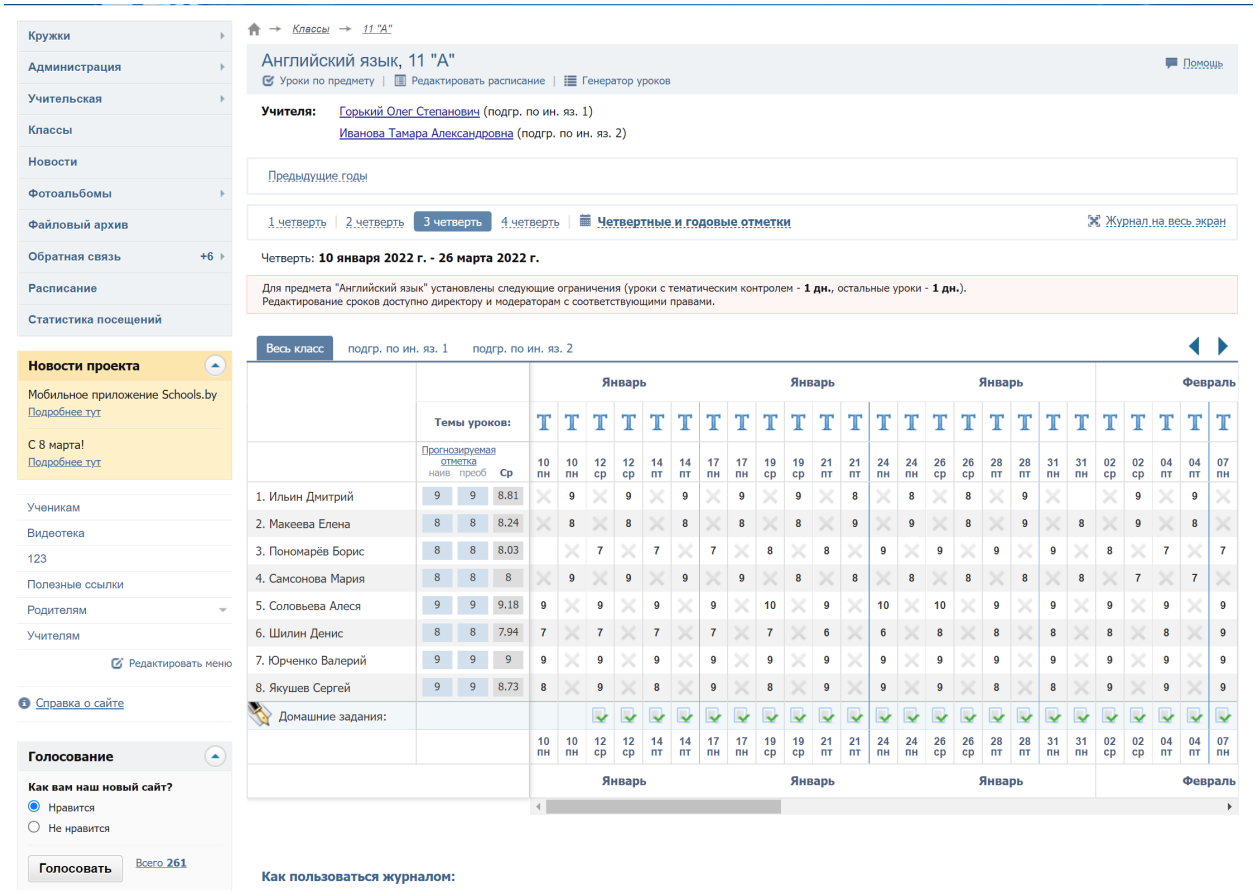

Рисунок 1.1- Образовательная платформа – «SCHOOLS.BY». Электронный журнал

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» заключаются 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:

Индивидуальные комментарии учителя к выставленной отметке, прикреплённые файлы к домашнему заданию (тест, лекция, конспект), учебная литература по теме урока, темы пройденных уроков, информация о планируемых контрольных работах и их темах, оповещение о пропусках по SMS и e-mail, расписание учителей-предметников, сервис «Архив успеваемости», сервис «Графики успеваемости» (графики текущих отметок), сервис «Рейтинги» (сервис позволяет ознакомиться с рейтингами по предметам в классе. Рейтинги формируются автоматически исходя из выставленных отметок на платформе), сервис «Наблюдение учащегося» (это позволяет наблюдать за успеваемостью учащегося), контроль расхода оплаты за питание, отсутствие рекламы в электронном дневнике и мобильном приложении.

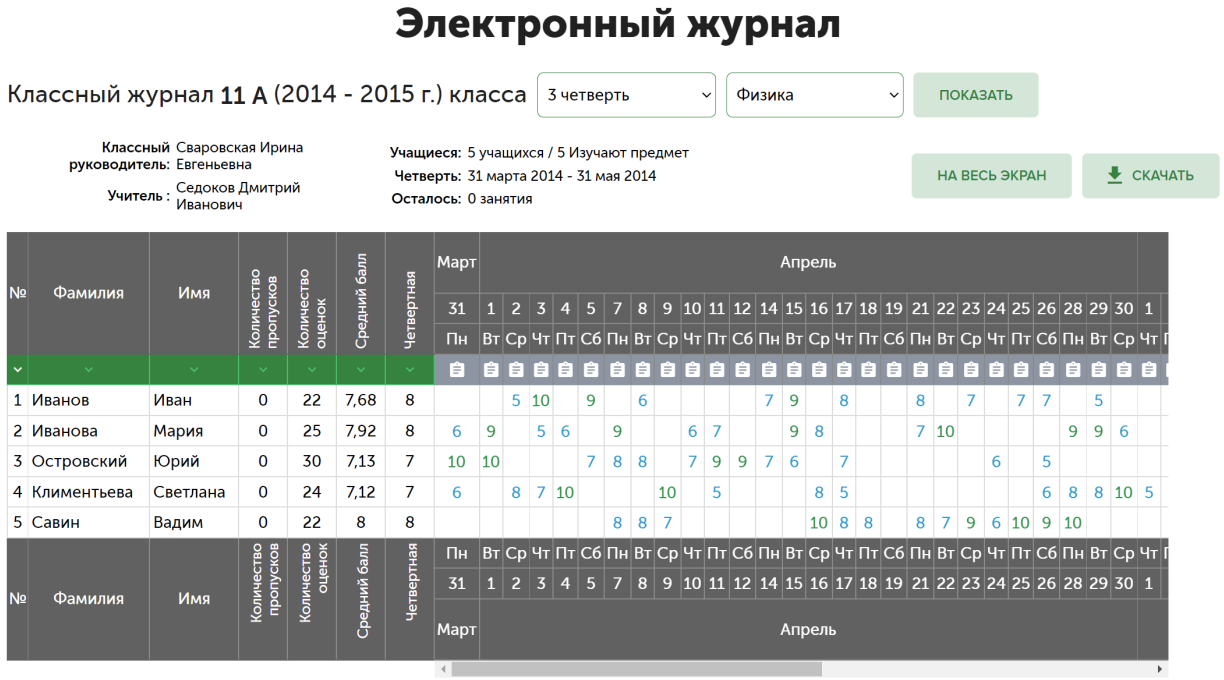

Рисунок 1.2 - Платформа электронных сервисов для образования «Знай.бай»

Также как и в прошлом аналоге можно отметить обширный функционал, однако данное решение не решает проблем предыдущего. Еще в глаза бросается стоимость «премиум пакета»: 0,35 рублей в день, что в пересчете на год будет 127,75 рублей, что в свою очередь много больше, описанного ранее аналога.

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).

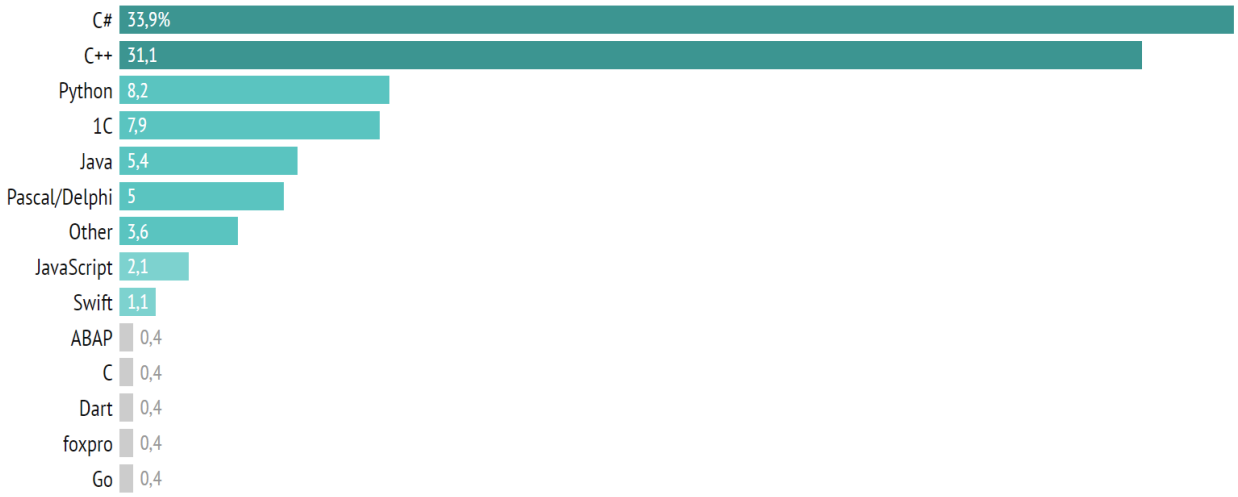

Рисунок 1.3 – Рейтинг языков программирования для написания приложений для персональных компьютеров

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).

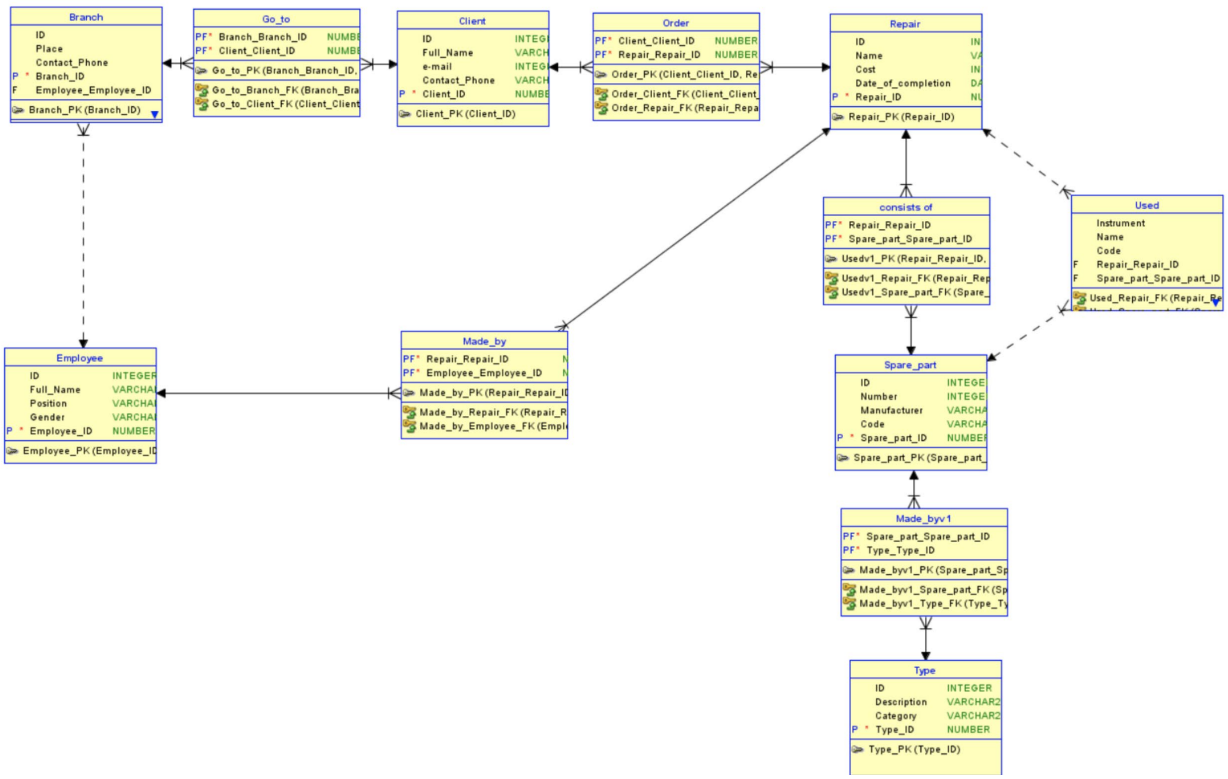

Рисунок 1.4 – Пример реляционной диаграммы, полученной в Oracle SQL Developer Data Modeler

```
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 системе 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.

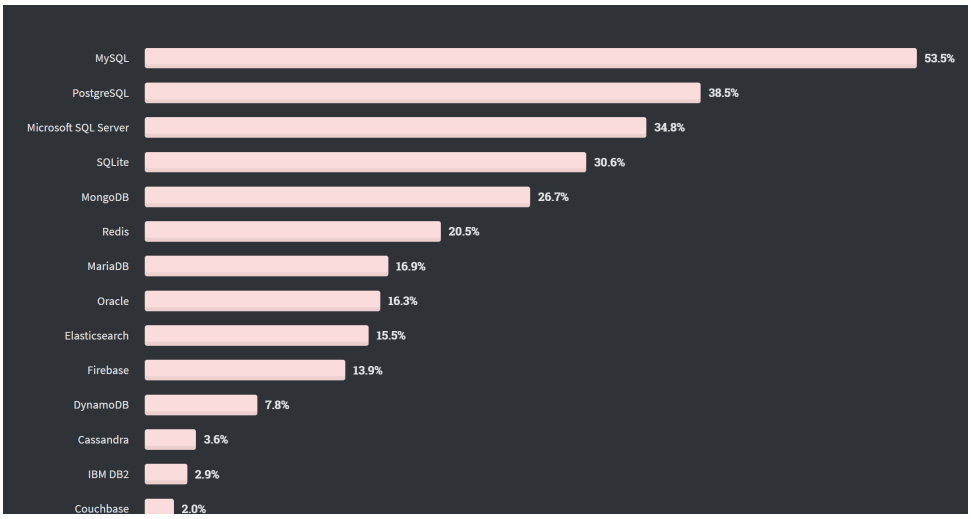

Рисунок 1.6 – Выбор систем баз данных, среди профессиональных разработчиков в 2020 году

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.

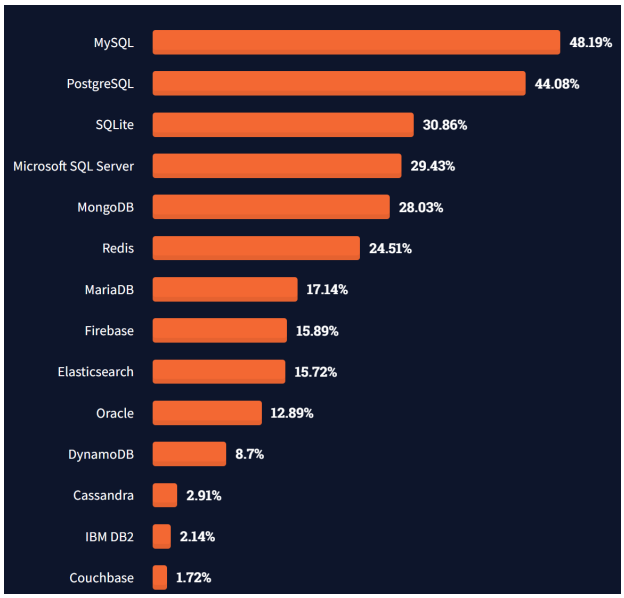

Рисунок 1.7 – Выбор систем баз данных, среди профессиональных разработчиков в 2021 году

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 заключаектся 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 позволят 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) б)

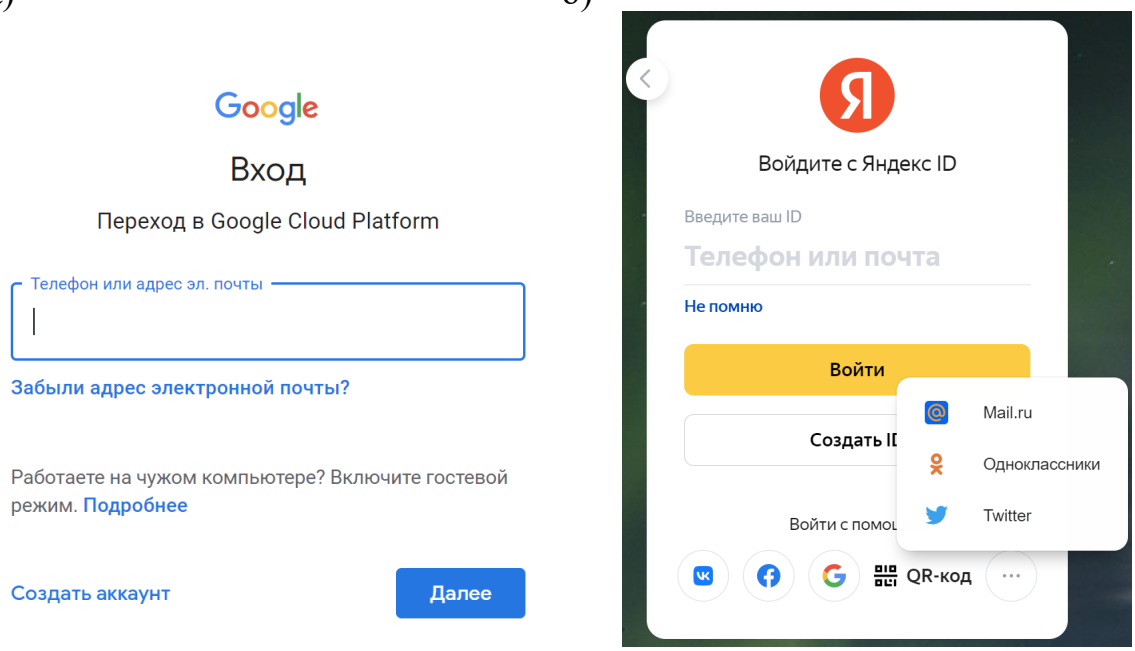

Рисунок 1.8 – Авторизация в системах: а – «Google Cloud Platform»; б – «Яндекс ID»

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» выставлять 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 фреймвора `QMainWindow` классе, который и будет является 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-а, 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-а. Используется при формировании запросов к базе данных, видоизменения значения переменной token_toUtf8;

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

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

– метод public setToken_toUtf8() – устанавливает определенное значение token_toUft8.

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-а, 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 с 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 «M» 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 выставлять 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 «M» 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 «учащемуся»;

– `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 определенный «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 и 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-а. Используется как раз таки для передачи значения переменной в другие окна приложения;

– метод public setLogin() – устанавливает определенное значение login-а. 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 «OK» 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-а. Используется как раз таки для передачи значения переменной в другие окна приложения;

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

– метод public getStudent_fullname() – получает значение student_fullname-а. Используется для передачи значения переменной в другие окна приложения, формирования запросов к SQL базе данный, 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`-а. Используется как раз таки для передачи значения переменной в другие окна приложения;

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

– метод `public getStudent_name()` – получает значение `student_name`-а. Используется для передачи значения переменной в другие окна приложения, формирования запросов к SQL базе данный, 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-а. Используется как раз таки для передачи значения переменной в другие окна приложения;

– метод public setLogin() – устанавливает определенное значение login-а. 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-а. Используется как раз таки для передачи значения переменной в другие окна приложения;

– метод public setLogin() – устанавливает определенное значение login-а. 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_numb_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-а. Используется как раз таки для передачи значения переменной в другие окна приложения;

– метод public setLogin() – устанавливает определенное значение login-а. 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 классам: 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, хранящее 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-а.

Используется как раз таки для передачи значения переменной в другие окна приложения;

– метод public setLogin() – устанавливает определенное значение login-а. 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-а (имя авторизованного пользователя). Используется при составлении SQL-запросов внутри класса;

– метод public setFull_name_parent () – устанавливает определенное значение full_name_parent-а. 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 переменнной `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 используются, 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.