# **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.

## **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.

## **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.

## **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;

## **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.

## **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).

## **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».

## **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.

## **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».

## **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».

## **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.

## **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».

## **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.

## **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.

## **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.

## **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».

## **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.

## **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.

## **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.

## **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.

## **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.

## **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.

## **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.

## **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.

## **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.

## **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;

## **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. Сheck\_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);

## **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).

## **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.

## **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 перемененной 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.