**Отчет по проектной эксплуатационной практике**

**Про библиотеки и модули, которые использовал**

Взаимодействие клиента с сервером осуществляется с использованием HTTP-запросов последнему при открытии окон, нажатиях на кнопки, выборе данных (группы, даты занятия, время занятия) в предусмотренных для этого виджетах, а также, внесении, удалении и изменении каких-либо данных, в процессе работы клиента.

Для отправки HTTP-запросов со стороны клиента использовались встроенные средства QT. В частности, модуль Qt Network и его классы QNetworkAccsessManager, QNetworkRequest, QNetworkReply.

Данный модуль предоставляет возможности для выполнения различных сетевых операций, в том числе и реализации HTTP-запросов.

API доступа к сети построен вокруг объекта QNetworkAccessManager, который содержит общую конфигурацию и настройки для отправляемых им запросов. Он содержит конфигурацию прокси и кэша, а также сигналы, связанные с такими проблемами, и сигналы ответа, которые можно использовать для отслеживания хода выполнения сетевой операции.

После создания объекта QNetworkAccessManager приложение может использовать его для отправки запросов по сети. Предоставляется группа стандартных функций, которые принимают запрос и дополнительные данные, и каждая из них возвращает объект QNetworkReply. Возвращенный объект используется для получения любых данных, возвращаемых в ответ на соответствующий запрос.

QNetworkRequest, в свою очередь, содержит запрос, который нужно отправить с помощью QNetworkAccessManager. Содержит информацию, необходимую для отправки запроса по сети, например, URL-адрес и некоторую вспомогательную информацию, которую можно использовать для изменения запроса.

Для формирования тела HTTP-запроса в JSON формате использовались следующие классы: QJsonDocument, QJsonObject, QJsonArray, QJsonValue – из модуля Qt Core.

Класс QJsonDocument предоставляет способ чтения и записи документов JSON. Этот класс, упаковывает полный документ JSON и может читать и записывать этот документ как из текстового представления в кодировке UTF-8, так и из собственного двоичного формата Qt.

Парсер работает очень быстро и эффективно и преобразует JSON в двоичное представление, используемое Qt.

К документу можно отправить запрос о том, содержит ли он массив или объект. Массив или объект, содержащийся в документе, можно получить, а затем прочитать или изменить.

При помощи QJsonArray, QJsonObject, QJsonValue можно инкапсулировать массив, объект и значение JSON, соответственно.

Графическая составляющая клиента создана в Qt Creator, по большей части с использованием Qt Designer для того, чтобы не загромождать программный код, прописывая, например, положение элементов в окне или другие составляющие пользовательского интерфейса.

Алгоритм работы с HTTP-запросами на стороне клиента можно разбить на следующие части: получение данных с сервера для их обработки и последующего вывода пользователю в указанном формате (в таблицах, выпадающих списках, виджетах даты и времени и т.д.); а также, обработку данных, введенных пользователем, для подготовки их к отправке и, непосредственно, отправку этих данных на сервер.

**Алгоритмы**

MainWindow

**Конструктор класса**

1. Начало

2. Инициируем объекты для открытия других окон, которые доступны из главного окна, а также новый объект загрузки, для загрузки доступных групп.

3. Вызываем процедуру открытия главного окна, когда на одном из второстепенных окон нажата соответствующая кнопка.

4. Открываем главное окно.

5. Конец

**Процедура разрушения (закрытия) главного окна**

1. Начало

2. Удаляем все созданные во время работы программы временные файлы.

3. Закрываем окно.

4. Конец

**Процедура открытия окон для просмотра (отметки) посещаемости, а также просмотра списка студентов**

1. Начало

2. Открываем новое второстепенное окно.

3. Закрываем главное окно.

4. Читаем список групп

5. Конец

CheckAttendance

**Конструктор класса**

1. Начало

2. Инициализируем новый объект загрузки и таблицу.

3. Передаем номер группы, для которой нужно загрузить данные.

4. Получаем файл с информацией о студентах и их посещаемости.

5. Конец

**Процедура чтения групп студентов**

1. Начало

2. Открытие файла со списком групп.

3. Парсинг содержимого файла.

4. В файле содержится массив? Если нет, то переход к п.7.

5. Взятие очередного элемента массива group. Если пройден последний элемент массива, то переход к п.6.

6. Добавление элемента массива в конец выпадающего списка (combobox).

7. Конец

**Процедура чтения информации о студентах**

1. Начало

2. Открытие файла со списком студентов.

3. Париснг содержимого файла.

4. Открытие файла с посещаемостью студентов.

5. Парсинг содержимого файла.

6. В файле с посещаемостью содержится массив? Если нет, то переход к п.14.

7. В файле со студентами содержится массив? Если нет, то переход к п.14.

8. Взятие очередного элемента массива student. Если пройден последний элемент массива, то переход к п.14.

9. Получаем этот элемент, как объект.

10. Взятие очередного элемента массива timerecord. Если пройден последний элемент массива, то переход к п.8.

11. Получаем элемент timerecord как объект.

12. Совпадает ли идентификатор студента в списке групп с идентификатором студента в списке посещаемости? Если нет, то переход к п.10.

13. Заполняем таблицу просмотра посещаемости по столбцам.

14. Изменяем внешний вид таблицы под содержимое.

15. Конец

**Процедура возврата к главному окну**

1. Начало

2. Закрываем активное окно.

3. Сигнализируем главному окну о необходимости его открытия.

4. Конец

**Процедура получения номера группы, выбранного пользователем в выпадающем списке**

1. Начало

2. Передаем номер группы, как строку, сигнализируя о начале загрузки данных для выбранной группы.

3. Конец

StudentList

**Конструктор класса**

1. Начало

2. Инициализация и открытие одного из окон, нового объекта загрузки и таблицы.

3. Передаем номер группы, для которой нужно загрузить данные.

4. Получаем файл с информацией о студентах.

5. Отправляем информацию о студенте с которым надо произвести манипуляции.

6. Конец

**Процедура чтения списка групп (рассмотрена ранее)**

**Процедура чтения информации о студентах**

1. Начало

2. Открытие файла со списком студентов.

3. Париснг содержимого файла.

4. В файле с посещаемостью содержится массив? Если нет, то переход к п.8.

5. Взятие очередного элемента массива student. Если пройден последний элемент массива, то переход к п.8.

6. Получаем этот элемент, как объект.

7. Заполняем таблицу просмотра посещаемости по столбцам.

8. Изменяем внешний вид таблицы под содержимое.

9. Конец

**Процедура удаления студента из списка**

1. Начало

2. Получаем значение очередной ячейки в строке, которая была выбрана для удаления. Если пройдена последняя ячейка, то переход к п.4.

3. Добавляем содержимое ячейки в массив строк.

4. Передаем полученный массив и сигнализируем о возможности его удалении.

5. Открываем окно с подтверждением удаления.

6. Конец

**Процедура изменения данных о студенте**

1. Начало

2. Получаем значение очередной ячейки в строке, которая была изменена. Если пройдена последняя ячейка, то переход к п.4.

3. Добавляем содержимое ячейки в массив строк.

4. Передаем полученный массив и сигнализируем о возможности его изменения.

5. Открываем окно с подтверждением изменения.

6. Конец

AddStudent

**Процедура добавления студента**

1. Начало

2. Открытие нового окна для добавления студента.

3. Конец

**Процедура отмены добавления нового студента**

1. Начало

2. Закрываем активное окно.

3. Сигнализируем об открытии окна списка студентов.

4. Конец

**Процедура добавления нового студента**

1. Начало

2. Получаем текст, введенный в поля внутри окна.

3. Значение, введенное в поле группы можно преобразовать в целое число? Если нет, то переход к п.9.

4. Формируем JSON файл, в который записываем действие с данными и сами данные.

5. Открываем файл на чтение и отправляем на сервер методом POST.

6. Закрываем активное окно.

7. Сигнализируем о добавлении нового студента.

8. Конец

**Процедура проверки правильности выполнения запроса**

1. Начало

2. Запрос прошел без ошибок? Если нет, то переход к п.4.

3. Выводим сообщение в консоль о том, что файл отправлен и удаляем объект ответа.

4. Выводим в консоль сообщение об ошибке.

5. Конец

AlterStudent

**Процедура отмены изменения информации о студенте (аналогична добавлению студентов)**

1. Начало

2. Закрываем активное окно.

3. Сигнализируем об открытии окна списка студентов.

4. Конец

**Процедура изменения информации о студенте**

1. Начало

2. Получаем измененные данные студента из массива строк.

3. Формируем JSON файл, в который записываем действие с данными и сами данные.

4. Открываем файл на чтение и отправляем файл на сервер методом POST.

5. Закрываем активное окно.

6. Сигнализируем о изменении данных студента.

7. Конец

**Процедура проверки правильности выполнения запроса** **(описана ранее)**

DeleteStud

**Процедура отмены удаления студента (аналогична добавлению студента)**

1. Начало

2. Закрываем активное окно.

3. Сигнализируем об открытии окна списка студентов.

4. Конец

**Процедура удаления студента**

1. Начало

2. Получаем данные студента, которого надо удалить, из массива строк.

3. Формируем JSON файл, в который записываем действие с данными и сами данные.

4. Открываем файл на чтение и отправляем файл на сервер методом POST.

5. Закрываем активное окно.

6. Сигнализируем о изменении данных студента.

7. Конец

**Процедура проверки правильности выполнения запроса (описана ранее)**

Download

**Конструктор класса**

1. Начало

2. Создаем объект для доступа к сети (для списка групп, студентов и таблицы посещаемости).

3. Вызываем процедуру для обработки полученных данных после того, как выполнение запроса будет завершено.

4. Конец

**Процедура получения данных о группах от сервера**

1. Начало

2. Указываем URL-адрес или локальный адрес, по которому расположены данные о группах.

3. Устанавливаем указанный адрес в объект запроса.

4. Отправляем запрос на получение данных по группам методом GET.

5. Сигнализируем об окончании выполнения запроса.

6. Конец

**Процедура получения данных о студентах и посещаемости (в целом аналогична предыдущему алгоритму)**

1. Начало

2. В выпадающем списке выбран пункт «Все группы»? Если нет, то переход к п.7.

3. Указываем URL-адрес или локальный адрес, по которому расположены данные обо всех студентах.

4. Устанавливаем указанный адрес в объект запроса.

5. Отправляем запрос на получение списка студентов методом GET.

6. Сигнализируем об окончании выполнения запроса.

7. Указываем URL-адрес или локальный адрес, по которому расположены данные о студентах конкретной группы.

8. Устанавливаем указанный адрес в объект запроса.

9. Отправляем запрос на получение списка студентов методом GET.

10. Сигнализируем об окончании выполнения запроса.

11. Указываем URL-адрес или локальный адрес, по которому расположены данные о посещаемости студентов.

12. Устанавливаем указанный адрес в объект запроса.

13. Отправляем запрос на получение списка студентов методом GET.

14. Сигнализируем об окончании выполнения запроса.

15. Конец

**Обработка результатов запроса на получение списка групп**

1. Начало

2. Выполнение запроса завершилось с ошибкой? Если нет, то переход к п.4.

3. Выводим в консоль сообщение об ошибке.

4. Создаем или открываем для записи файл и записываем в файл содержимое ответа на запрос.

5. Уведомляем об успешном заполнении файла в консоли.

6. Сигнализируем о том, что запись в файл завершена.

7. Конец

**Обработка результатов запроса на получение списка студентов (аналогично группам)**

1. Начало

2. Выполнение запроса завершилось с ошибкой? Если нет, то переход к п.4.

3. Выводим в консоль сообщение об ошибке.

4. Создаем или открываем для записи файл и записываем в файл содержимое ответа на запрос.

5. Уведомляем об успешном заполнении файла в консоли.

6. Сигнализируем о том, что запись в файл завершена.

7. Конец

**Обработка результатов запроса на получение данных о посещаемости (аналогично группам)**

1. Начало

2. Выполнение запроса завершилось с ошибкой? Если нет, то переход к п.4.

3. Выводим в консоль сообщение об ошибке.

4. Создаем или открываем для записи файл и записываем в файл содержимое ответа на запрос.

5. Уведомляем об успешном заполнении файла в консоли.

6. Сигнализируем о том, что запись в файл завершена.

7. Конец

MarkAttendance (все действия во многом схожи с CheckAttendance)

**Конструктор класса**

1. Начало

2. Инициализируем новый объект загрузки и таблицу.

3. Передаем номер группы, для которой нужно загрузить данные.

4. Получаем файл с информацией о студентах и их посещаемости.

5. Конец

**Процедура чтения групп студентов**

1. Начало

2. Открытие файла со списком групп.

3. Парсинг содержимого файла.

4. В файле содержится массив? Если нет, то переход к п.7.

5. Взятие очередного элемента массива group. Если пройден последний элемент массива, то переход к п.6.

6. Добавление элемента массива в конец выпадающего списка (combobox).

7. Конец

**Процедура чтения информации о студентах**

1. Начало

2. Получаем значение даты и времени занятия, указанных пользователем и форматируем их.

3. Открытие файла со списком студентов.

4. Париснг содержимого файла.

5. В файле со студентами содержится массив? Если нет, то переход к п.11.

6. Взятие очередного элемента массива student. Если пройден последний элемент массива, то переход к п.14.

7. Получаем этот элемент, как объект.

8. Инициализируем объект чекбокса.

9. Заполняем таблицу просмотра посещаемости по столбцам. Последним столбцом добавляем чекбоксы.

10. Записываем данные из таблицы в JSON файл.

11. Изменяем внешний вид таблицы под содержимое.

12. Создаем файл для записи значений в чекбоксах.

13. Конец

**Процедура возврата к главному окну**

1. Начало

2. Закрываем активное окно.

3. Сигнализируем главному окну о необходимости его открытия.

4. Конец

**Процедура получения номера группы, выбранного пользователем в выпадающем списке**

1. Начало

2. Передаем номер группы, как строку, сигнализируя о начале загрузки данных для выбранной группы.

3. Конец

**Процедура проверки правильности выполнения запроса (описана ранее)**

**Процедура отправки данных о посещаемости**

1. Начало

2. Открываем файл, полученный при заполнении таблицы посещаемости.

3. Парсим этот файл.

4. Открываем файл со значениями чекбоксов на чтение и считываем данные построчно.

5. Заполняем массив строк значениями каждой строки.

6. Формируем JSON файл, в который записываем действие на добавление данных и перезаписываем данные, которые были в файле с добавлением элемента present, в котором хранятся данные о посещаемости каждого студента.

7. Открываем файл на чтение и отправляем на сервер методом POST.

8. Закрываем активное окно.

9. Сигнализируем о добавлении нового студента.

10. Конец