Aby skomunikować się z serwerem należy wykorzystać obiekt klasy Server istniejący w klasie App.

Pokaże na przykładzie rejestrowania użytkownika jak to zrobić.

1. Załóżmy, że mam przycisk w Menu, którego wciśnięcie zarejestruje mi nowego użytkownika.
2. Aby to zrobić musze wysłać dane do serwera. Wykorzystamy do tego typ komunikatu POST, nazwę komendy oraz dane w postaci JSONa.
3. Komendy są ukryte pod makrem COMMAND_TYPE_(….) w pliku nagłówkowym „commandType.hpp”. W naszym przypadku będzie to makro COMMAND_TYPE_CLIENT_REGISTER .
4. JSON potrzebny do dodania użytkownika tworzymy na podstawie jego parametrów. Najpierw tworzę obiekt Json do przechowywania danych o użytkowniku.

```
QJsonObject objUser;
objUser.insert(USER_PARAMETERS_USER_ID, QString("0"));
objUser.insert(USER_PARAMETERS_USER_NAME, QString("admin2"));
objUser.insert(USER_PARAMETERS_USER_PASSWORD, QString("admin123"));
objUser.insert(USER_PARAMETERS_USER_PESEL, QString("9510050454"));
objUser.insert(USER_PARAMETERS_USER_FIRST_NAME, QString("Tomek"));
objUser.insert(USER_PARAMETERS_USER_SURNAME, QString("Tomkowski456"));
```

   Makra po postacią USER_PARAMETERS_(….) znajdują się w pliku nagłówkowym „userParametersEnum.hpp” i opisują one nazwy kluczy parametrów dla użytkownika. Nie trzeba podawać wszystkich danych. Np. W przypadku usuwania lub pobierania informacji o użytkowniku wystarczy podać tylko USER_PARAMETERS_USER_ID. Można również wysyłać puste obiekty.

5. Po utworzeniu obiektu Json o użytkowniku należy go dodać do obiektu JSON, który będzie wysyłany. Dlaczego? Bo serwer w tym wypadku oczekuję na klucz o nazwie „user”, w którym ma dane o użytkowniku.

```
QJsonObject obj;
obj.insert(USER_JSON_KEY_TEXT, objUser);
```

6. Ponieważ program nie wyśle obiektu JSON, tylko tekst, więc należy użyć klasy, która przetwarza obiekty JSON na tekst i odwrotnie (co ważne).

```
QJsonDocument jsonDoc(obj);
```

7. Następnie musimy użyć metody klasy Server pod nazwą setLastRequest. Są dwie wersje tej metody: 1 – podajemy argumenty („jakas_komenda”, GET), gdzie GET – jest to wartość enuma MessageType, (Post nie zadziała dla tej metody), 2 – („jakas_komenda”, POST , (jakisObjektQJsonDoc)).

```
parent->getParent()->getServer().setLastRequest(QString(COMMAND_TYPE_CLIENT_REGISTER_TEXT), POST, jsonDoc);
while(!parent->getParent()->getServer().isServerReplied());
```

   Następnie oczekujemy na odpowiedz serwera. Aby pobrać dane wykorzystujemy komendę:

```
QJsonDocument jsonDoc = QJsonDocument::fromJson(parent->getParent()->getParent()->getParent()->getServer()->getLastReplay()->readAll());
QJsonObject jsonMainObj = jsonDoc.object();
```

UWAGA !! Serwer zawsze zwraca obiekt pod kluczem „error” dostępny w pliku nagłówkowym „returnErrorType.hpp” . Pasuje go sprawdzać, czy nie wystąpiły jakieś błędy.

Aby sprawdzić czy dany klucz istnieje wykorzystujemy metodę w warunku

```
if(jsonMainObj.value(RETURN_ERROR_JSON_VARIABLE_TEXT) == QJsonValue::Undefined)
```

Jeśli warunek jest spełniony to tego klucza NIE MA.

Jak chce wartość tego błedu do enuma to piszecie:

```
ReturnErrorType ret =  static_cast<ReturnErrorType>(actualSocket->requestData.value(USER_JSON_KEY_TEXT).toInt());
```

Jeśli będzie potrzebny static_cast, bo nie sprawdzałem.