



M Ű E G Y E T E M 1 7 8 2

Frontend készítés Qt alapokon

Rendszertervezés laboratórium 2

Mérési útmutató

Készítette: Blázovics László

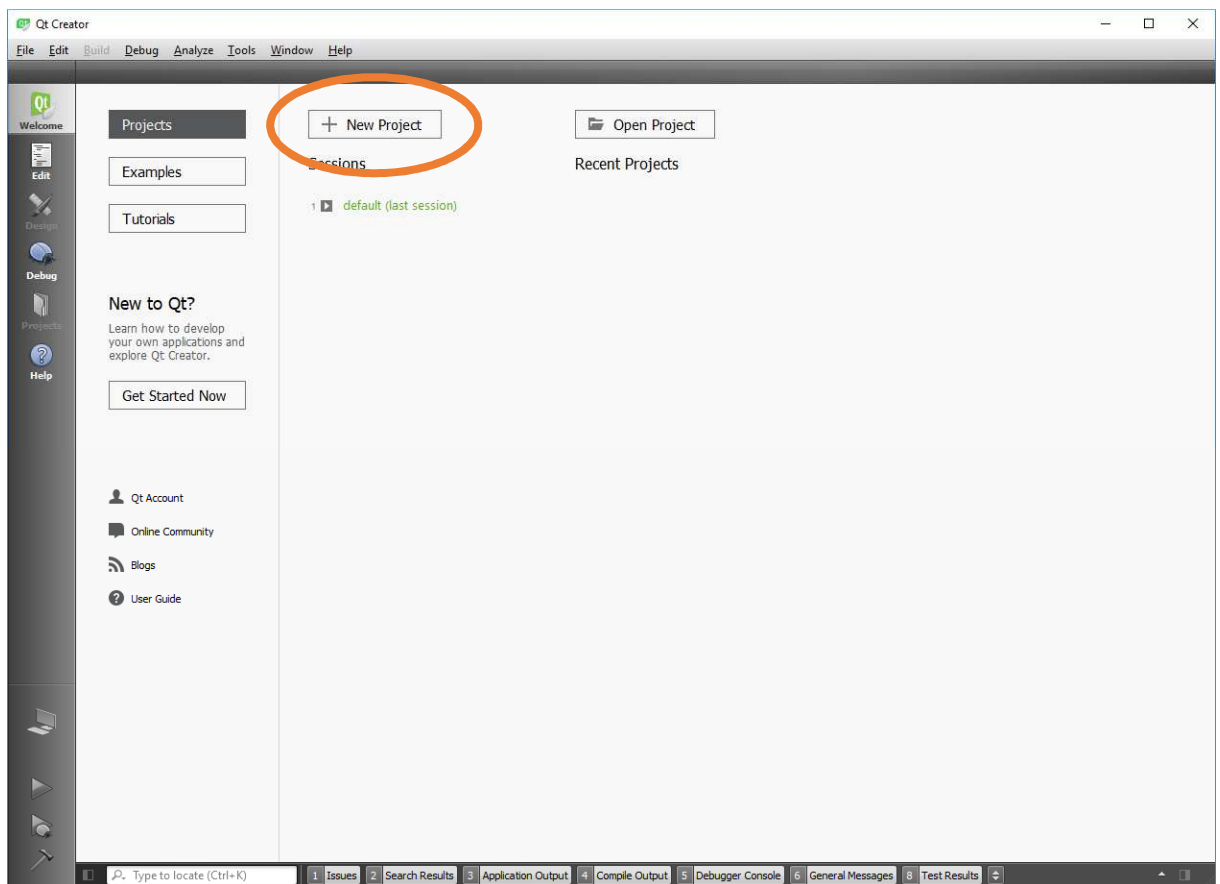
Verzió: 1.0

2017.

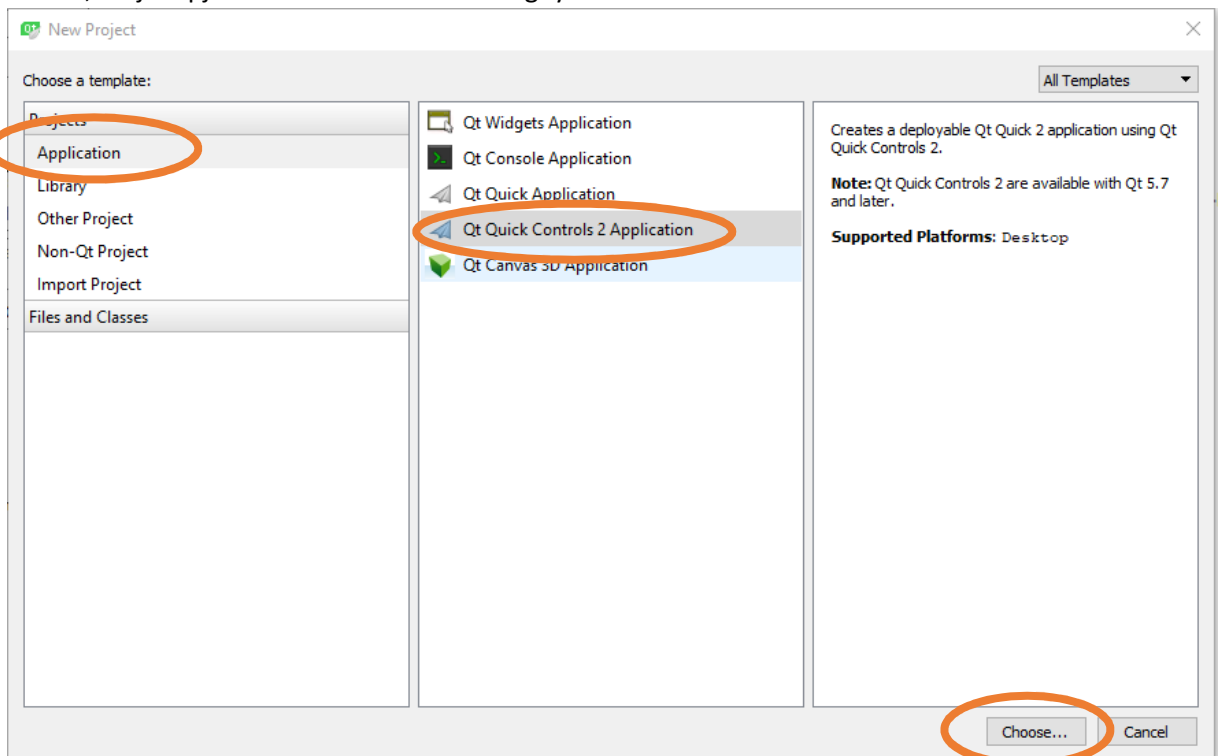
Budapesti Műszaki és Gazdaságtudományi Egyetem

Méréstechnika és Információs Rendszerek Tanszék

1. Nyissuk meg a **Qt Creator** alkalmazást, majd válasszuk a **New Project** menüpontot.




2. Válasszuk ki az **Application** Project sablonok közül a „**Qt Quick Controls 2 Application**” sablon, majd lépünk tovább a **Choose...** megnyomásával.



3. Nevezzük el a projektet és adjuk meg, hogy hová hozza létre a **Qt Creator**, majd lépünk a következő oldalra.

Tipp: Érdemes külön könyvtárban belül létrehozni az egyes projektjeinket, mert a shadow build könyvtárakat a projektkönyvtárral egy hierarchiaszinten (azaz nem azon belül) hozza létre fordításkor a fejlesztőkörnyezet.

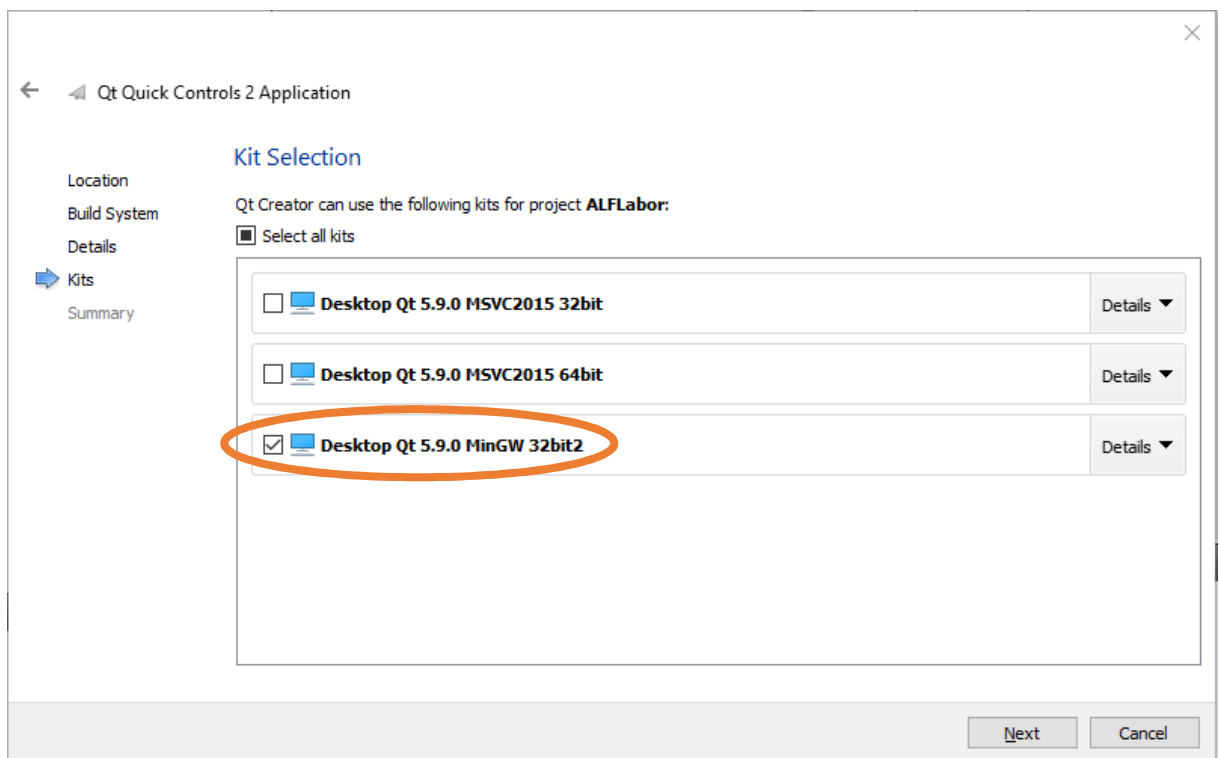


Name:

Create in:

Use as default project location

4. A „**Build System:**” beállítást hagyjuk alapértelmezettként a **qmake**-en, majd kattintsunk a **Next** gombra.
5. A „**Qt Quick Controls 2 Style:**” beállításnál válasszuk ki a legördülő listából a **Material** lehetőséget, majd ismét lépünk tovább.
6. Válasszuk ki a megfelelő fordító és futtató „**Kit**”-nek a **Desktop MinGW**-t, majd lépünk megint tovább.



Qt Quick Controls 2 Application

Location
Build System
Details
Kits
Summary

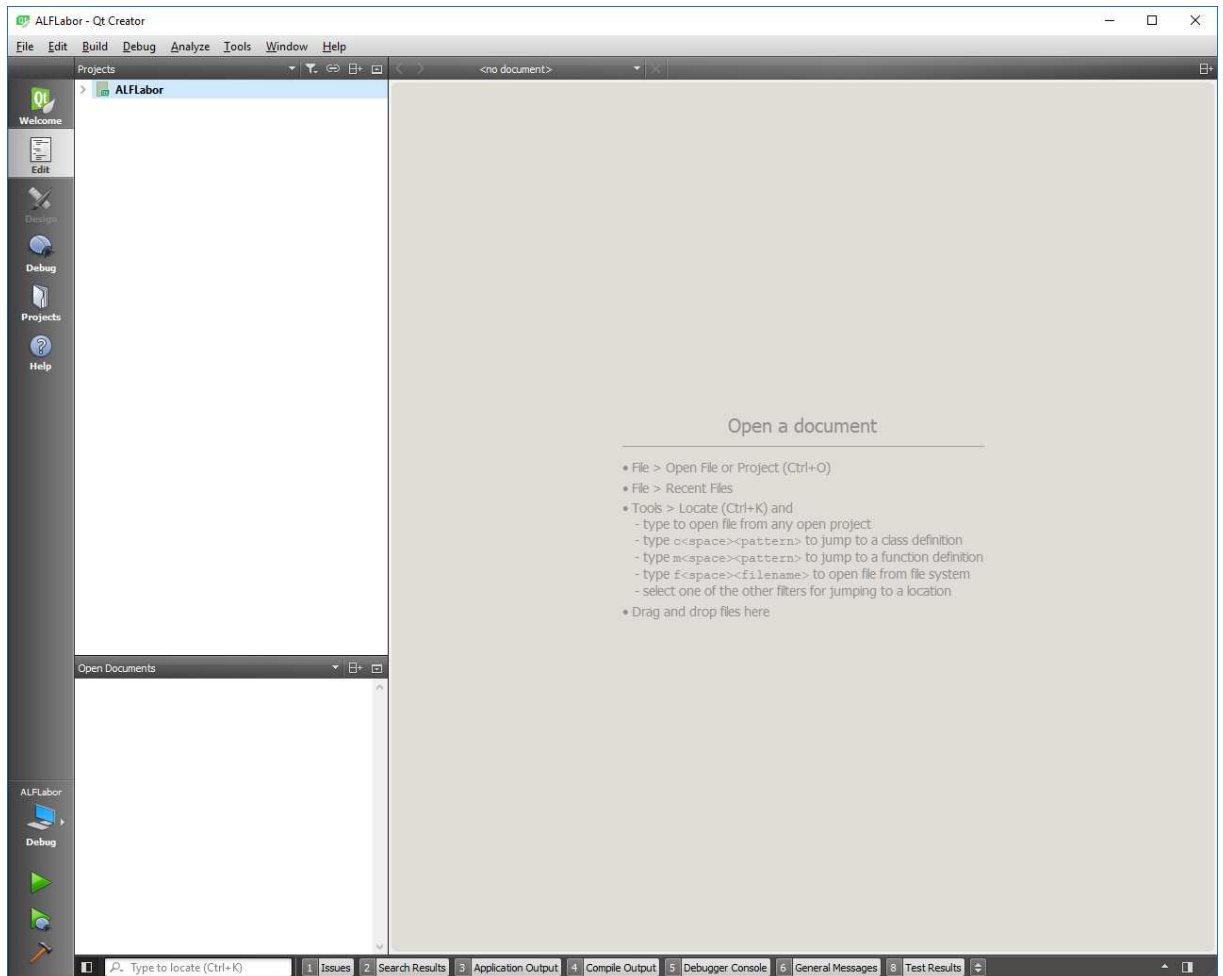
Kit Selection

Qt Creator can use the following kits for project **ALFLabor**:

Select all kits

<input type="checkbox"/> Desktop Qt 5.9.0 MSVC2015 32bit	Details ▼
<input type="checkbox"/> Desktop Qt 5.9.0 MSVC2015 64bit	Details ▼
<input checked="" type="checkbox"/> Desktop Qt 5.9.0 MinGW 32bit2	Details ▼

7. Végül a Project Management fülön nyomjuk meg a **Finish** gombot. Ha mindent jól csináltunk az alábbihoz hasonló képernyőt kell, hogy lássunk.



A **Qt Creator** felületének nagy része hasonlít a többi elterjedtebb fejlesztőkörnyezethez, ugyanakkor picit el is tér tőlük.

Ha nagyon úgy éreznéd, hogy el vagy tévedve, itt egy részletes leírás, ami minden kérdésre válasz tud adni.

<http://doc.qt.io/qtcreator/creator-quick-tour.html>

8. Mielőtt rátérnénk a kódra, nyomjuk meg a bal alsó sarokban alulról a harmadik gombot, amin egy zöld „nyíl” van.



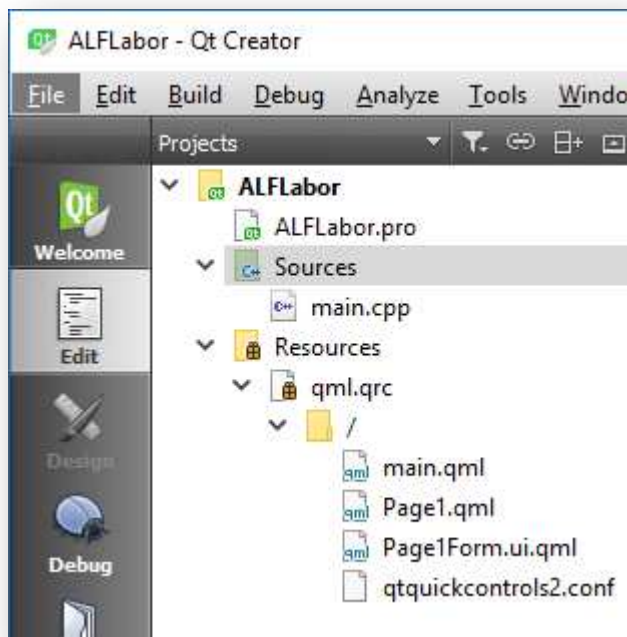
Ezzel elindul az alkalmazásunk. Ha nincs lefordítva, a fordító automatikusan le is fordítja nekünk a projektet. Az alkalmazást természetesen lehetséges debugger-rel együtt futtatni (alulról a második gomb, a „nyíl a bogárral”), ugyanakkor ezt a laborgépeken csak indokolt esetben tegyük meg, mert a szükséges .dll állományok betöltése akár egy percig is eltarthat.

Ezt a problémát az okozza, hogy a MinGW fordító nem annyira otthonos Windows környezetben, mint a Visual Studio. Utóbbi használatához ugyanakkor szükség van további komponensek telepítésre is:

<https://stackoverflow.com/questions/34957158/qt-creator-debug-mode-is-really-slow>

Az alkalmazás két, lapozható nézetből áll, aminek az első oldalán, ha beírunk valamit a beviteli mezőbe, majd megnyomjuk a gombot, a szöveg megjelenik a **Qt Creator** alján, az **Application Output** nézetben. Ezt fogjuk egy kicsit feldobni.

9. Most, hogy futtattuk az alkalmazást, nézzük meg, hogy mit generált le számunkra a fejlesztőkörnyezet. Ehhez a nyissuk le a **Left Sidebar**-ban, a **Projects** menüpont alatt látható **ALFLabor**-t a mögötte lévő nyíl megnyomásával. Ezt folytassuk rekurzívan, amíg lehet: Az újonnan előbukkanó lenyitható elemeket is nyissuk le, amíg mindet le nem nyitottuk.

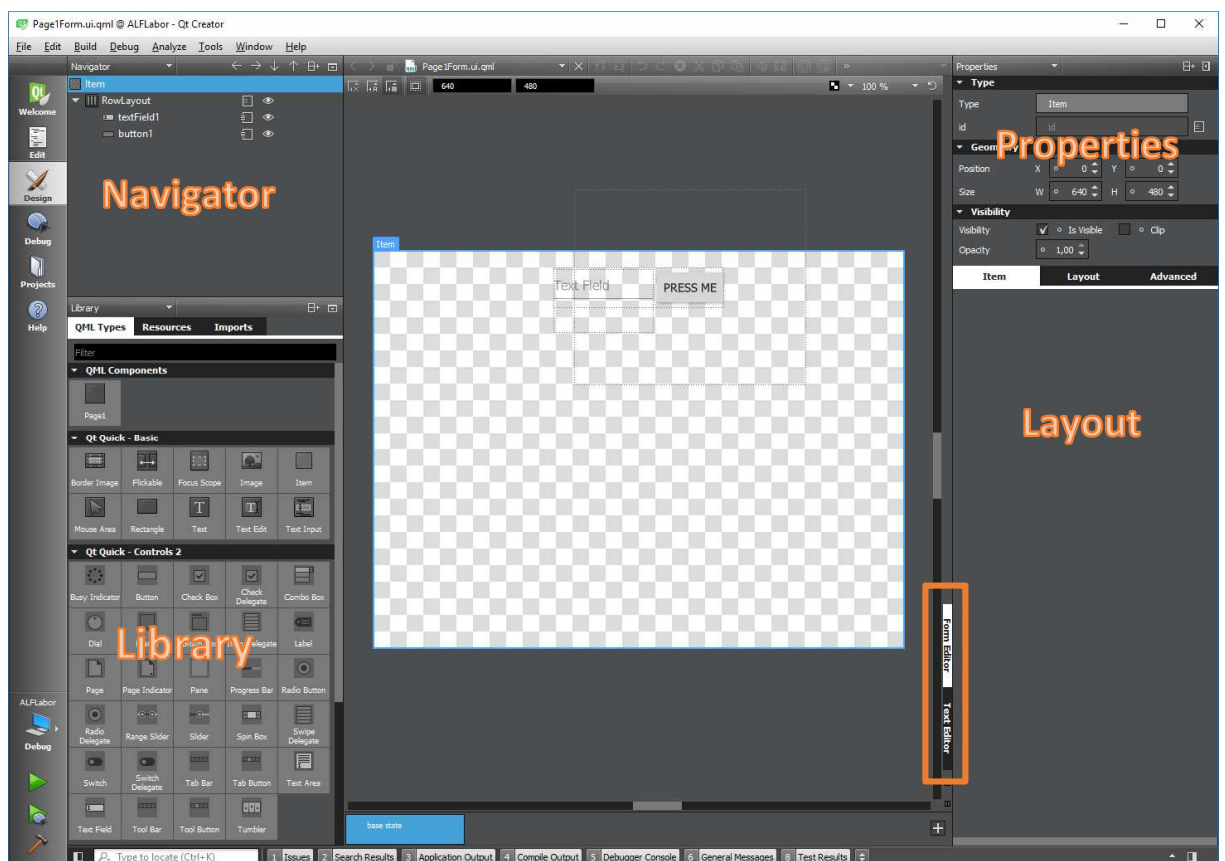


10. A sorrendben első fájl, amit szerkeszthetünk, az **ALFLabor.pro** project fájl. Itt tudunk módosításokat elvégezni a projekt szerkezetével kapcsolatban. Szerencsére a **Qt Creator** a forrásfájlok tekintetében helyettünk elvégzi ezeket a módosításokat, így nekünk nem kell ezen a laboron foglalkoznunk vele. Akit érdekel, hogy néz ki közelebbről dupla kattintással megnyithatja.
11. Továbbhaladva, a **main.cpp** fájlban van implementálva a QML tartalmak betöltése, illetve majd ide fognak kerülni a **signal-slot** összekötések is.

12. A **qml.qrc** állományt átugorva eljutottunk a **main.qml**-ig. Ez a fájl tartalmaz egy **ApplicationWindow** objektumot azon belül pedig egy **SwipeView**-t, aminek **swipeView** az **id** *property*-je.
- A **SwipeView**-n belül egy **Page1**, illetve egy simán **Page** névre hallgató elemet találunk. Utóbbi egy rendszer által definiált elem, míg előbbi egy specializált, „saját” elem, ami a **Page1.qml** állományban van definiálva. Ezért nem találunk sehol **TextField**, illetve **Button** komponenst a **main.qml**-ben.
- A **swipeView** alatt pedig az **ApplicationWindow footer** *property*-jeként van egy **TabBar** elem. Ha ezt az elemet megtaláltuk, töröljük is ki a „**footer:**”-rel együtt, mivel nem lesz rá szükségünk.
13. Ugyan fejlesztés során nem látjuk, de ha végiggondoljuk, világos, hogy az imént vizsgált **swipeView**-t is módosítani kell. A **currentIndex** *property* ugyanis az előbb töltött **tabBar**-ra **currentIndex** *property*-jét veszi alapul. Ezt írjuk át egyszerűen 0-ra.

```
currentIndex: 0
```

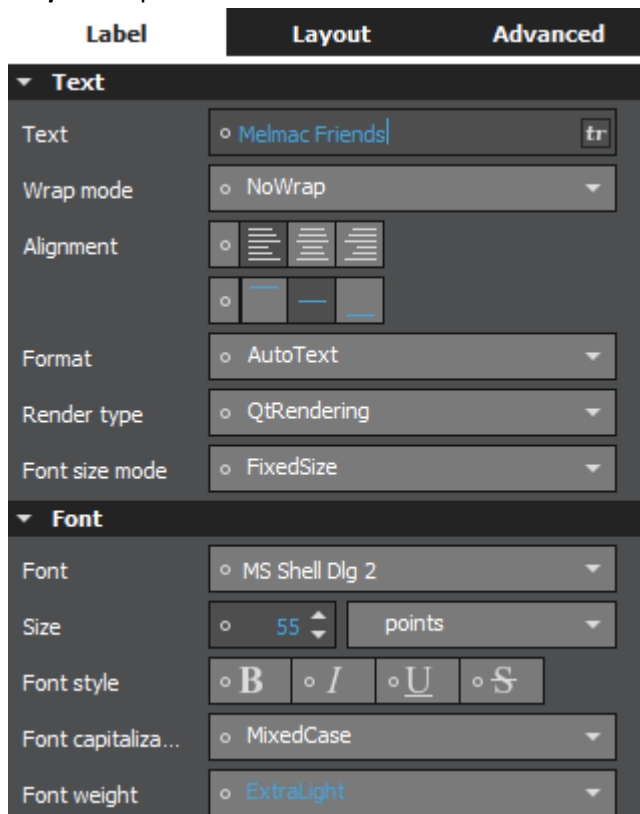
14. A **Page1.qml**-t átugorva, nyissuk meg a **Page1Form.ui.qml** fájlt kettőt kattintva rajta. Ha jól csináltuk, akkor megnyílik a **Design** nézet.



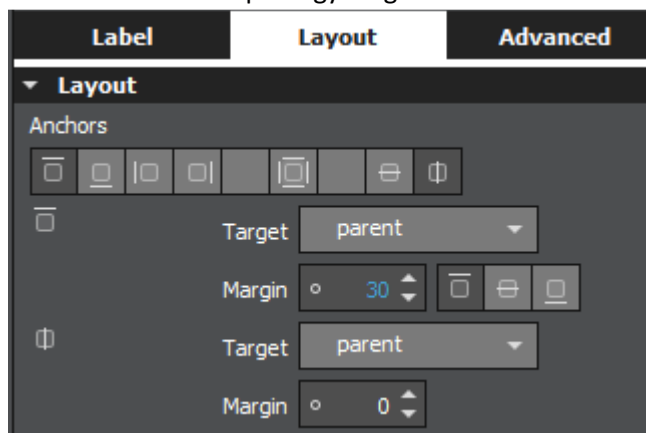
Emlékeztető: A .ui.qml kiterjesztés azt jelezi, hogy ebben a nézetben csak olyan tartalom van, ami a Designerben is szerkeszthető. Ha megpróbálunk visszaváltani az Edit földre, a Qt Creator jelzi is nekünk, hogy „ezt nem kéne”.

A **Qt Designer**, ami megnyílik a **Qt Creator** saját UI designere. Aki kicsit kaotikusnak találja a helyzetet, itt átnézheti a főbb funkciókat: <http://doc.qt.io/qtcreator/creator-using-qt-quick-designer.html>

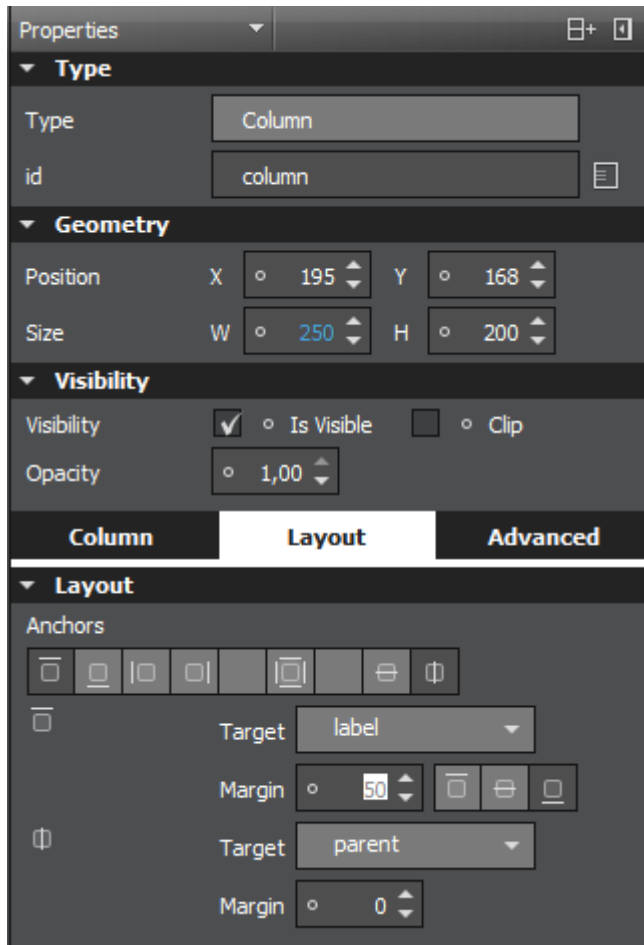
15. Először itt is törölni fogunk. Jelöljük ki a bal felső sarokban lévő **Navigator** nézetben az **Item**-en belül lévő **RowLayout**-ot, majd nyomjuk meg a **Delete** vagy a **Backspace** gombot.
16. A Navigator alatt lévő **Library** panelen belül keressük meg a „Qt Quick - Controls 2” szekciót és adjunk hozzá egy egyszerű drag&droppal az **Item** elemhez egy **Label**-t. Majd állítsuk be a **Properties** panel alsó részében **Label** fülnél az alábbiakat:



17. Majd váltsunk át a **Layout** fülre és állítsuk be, hogy ablak tetejétől 30 pixel legyen a távolság és vízszintesen középre legyen igazítva a felirat.

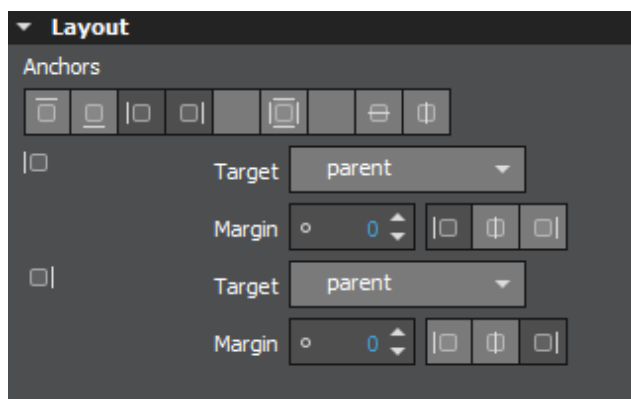


18. Ezután keressük ki a **Library**-ban a „Qt Quick – Layouts” szekciót és adjunk hozzá az **Item** elemhez egy **Column Layout**-ot is. A **Column Layout** legyen szintén vízszintesen középre rendezve, legyen a **Label** aljától 50 pixellel lejjebb, illetve szélessége legyen 250 pixel.

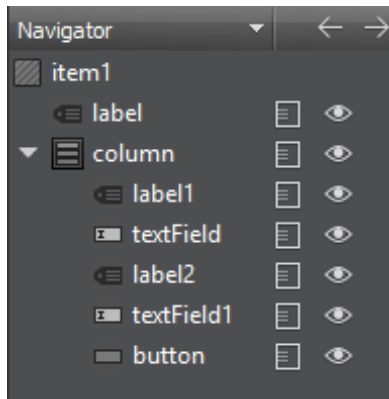


19. Ezután adjuk hozzá a **Column Layout**-hoz felváltva két-két **Label**-t („User name” és „Password” feliratokkal), illetve **TextField**-et. Az összes elem legyen úgy beállítva, hogy a **Column Layout** bal, illetve jobb oldalától **0-0** pixel távolságra legyenek.

20.



21. Végezetül pedig egy **Button**-t „Login” felirattal, vízszintesen középre rendezve.



22. Módosítsuk a két **TextField** *id*-jét a **Properties** panelen **userNameTextField**-re, illetve **passwordTextField**-re.
23. Ezzel el is készültünk a UI szerkeszthető részével. Most kapcsoljuk át a középső nézetet **Text Editor** módba a **Properties** panel melletti kapcsolóval.
24. Az **Item**-en belül, de a többi benne lévő elemen kívül, adjuk hozzá következő három sort:

```
property alias button: button
property alias username: userNameTextField.text
property alias password: passwordTextField.text
```

Ezzel azt tudjuk elérni, hogy a **Page1.qml**-ben lévő **Page1Form** elem – aminek a definícióján éppen most dolgoztunk – el tudja érni az ebben az állományban definiált elemeket, illetve azok *property*-jeit.

25. Mivel a **passwordTextField** elég szegényes így, hogy látszik a beírt jelszó szövege, állítsuk be az **echoMode** *property*-jét **TextInput.Password**-re.

```
TextField {
    echoMode: TextInput.Password
    id: passwordTextField
}
```

26. Most lépünk át a **Page1.qml** fájlra és az **Edit** fülön maradva adjuk hozzá a következő **loginPressed** névre hallgató *signal*-t, amit majd a **Button** fog elsütni annak **onClicked** eseménykezelőjében.

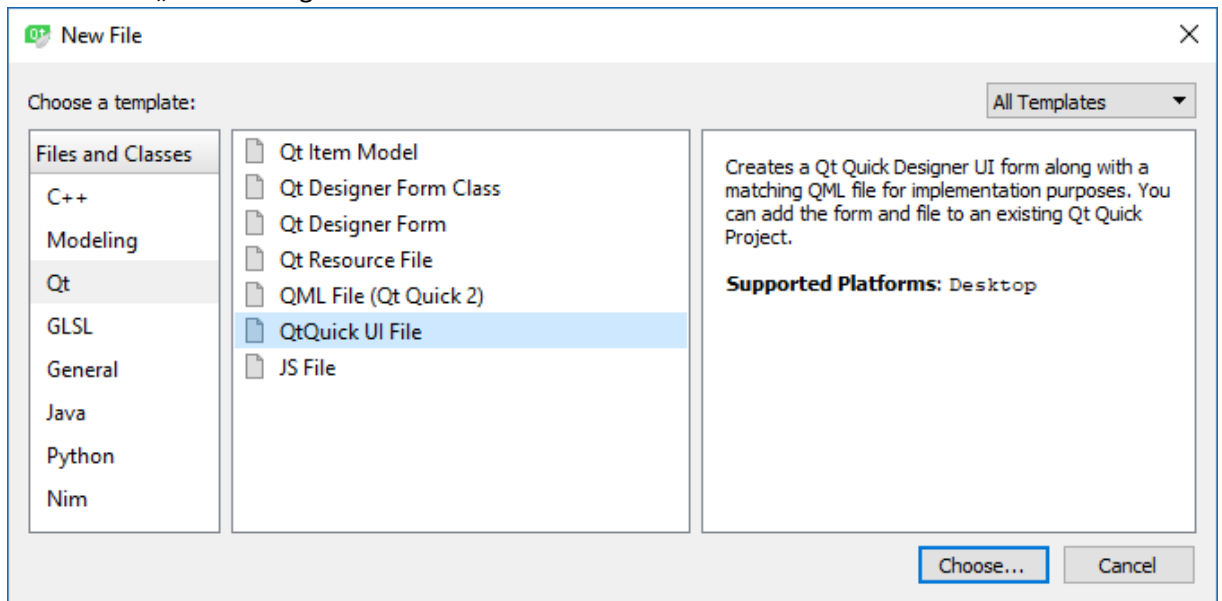
```
Page1Form {
    signal loginPressed(var username, var password);

    button.onClicked: {
        loginPressed(username, password)
    }
}
```

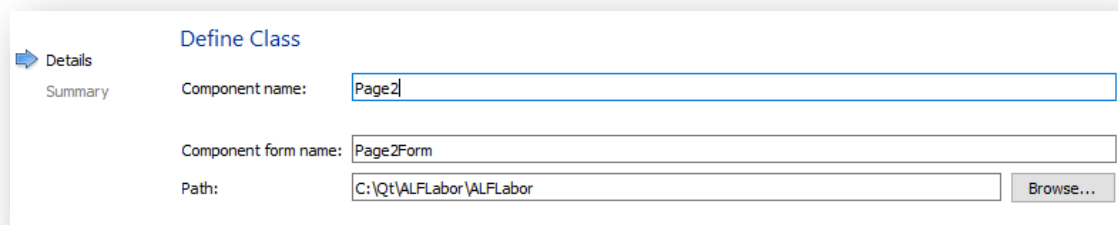
27. Most térjünk vissza a **main.qml** állományhoz és töröljük ki **SwipeView Page** paneljét. Ezt kellene kapnunk:

```
SwipeView {
    id: swipeView
    anchors.fill: parent
    currentIndex: 0
    Page1 {
    }
}
```

28. A **qml.qrc** fájlt kijelölve, jobb egérgombbal kattintva, hozzunk létre egy új fájlt az „Add New...” menüpontot kiválasztva.
29. A felugró menüben A **Qt** sablonok közül válasszuk ki a **QtQuickUI File** sablont, majd kattintsunk a „Choose...” gombra.



30. **Component name**-nek írjuk be a nem éppen informatív, de a laborfeladat összetettségéhez illeszkedő **Page2** nevet, majd lépünk tovább a **Next**, majd a **Finish** gomb megnyomásával. A „Component form name” automatikusan fog illeszkedni az új névhez, azt nem kell módosítani.



31. Ha megnyitjuk a **Page2Form.ui.qml**-t észrevehetjük, hogy a mérete nem akkora, mint a **Page1** form-é. Ezt orvosolandó, először is adjuk hozzá az új **Page2** komponensünket a **main.qml SwipeView**-jához és állítsuk be az **id** property-jét **page2**-re.

```

SwipeView {
    id: swipeView
    anchors.fill: parent
    currentIndex: 0
    Page1 {
    }
    Page2 {
        id: page2
    }
}

```

32. Majd töröljük ki a **Page2Form.ui.qml**-ben a **width** és a **height** property-eket értékeikkel együtt. Ezzel elérjük, hogy a szülő nézettől vegye át a méreteket.

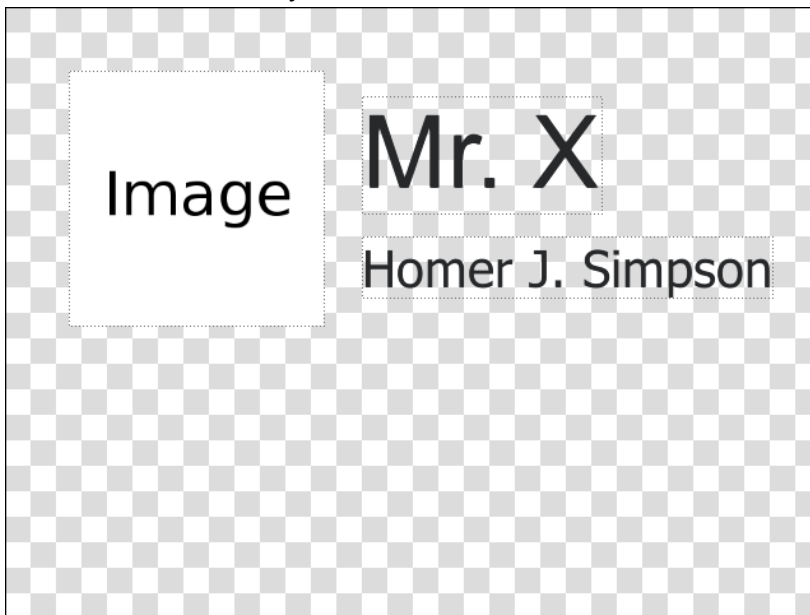
```
import QtQuick 2.4
Item {
    width: 400
    height: 400
}
```

33. Ahogy az észrevehető, a **Qt Designer** elég szegényes komponens palettát kínál a **Page2Form.ui.qml** esetében. A *Page1Form* esetében sokkal több komponens volt megjelenítve a **Library** panelen. Ez azért van így, mert nincs „import-olva” a *QtQuick.Controls 2.0*, illetve a *QtQuick.Layout 1.3*. a fájlban. Továbbá a *QtQuick 2.7* helyett **2.4**-es verzióval van importolva. Pótoljuk a hiányosságokat.

```
import QtQuick 2.7
import QtQuick.Controls 2.0
import QtQuick.Layouts 1.3

Item {
}
```

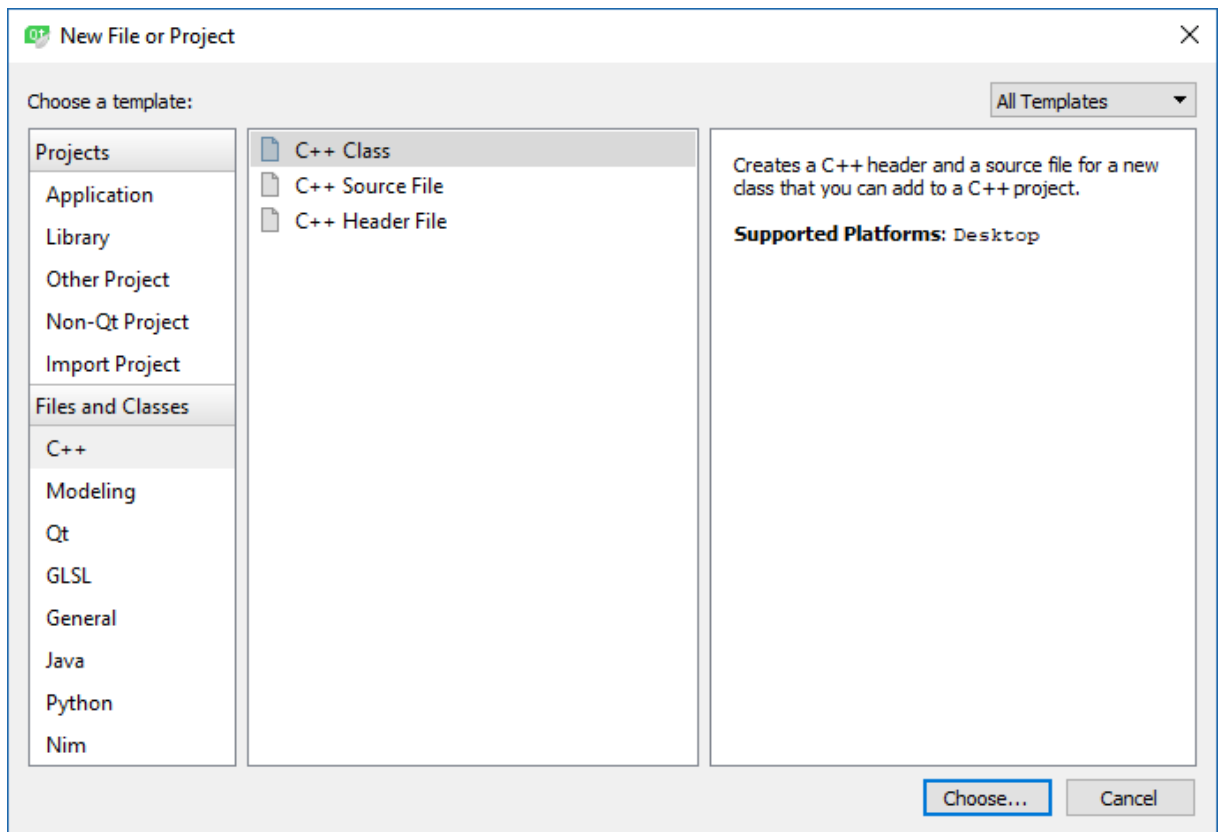
34. Most, hogy minden készen áll, adjunk hozzá a nézethez egy **Image**-t, és két **Label**-t. Az **Image** *id*-je legyen **image**, a két **Label**-é **title** és **subtitle**. Módosítsuk a layout property-ket, hogy az alábbi elrendezés alakuljon ki:



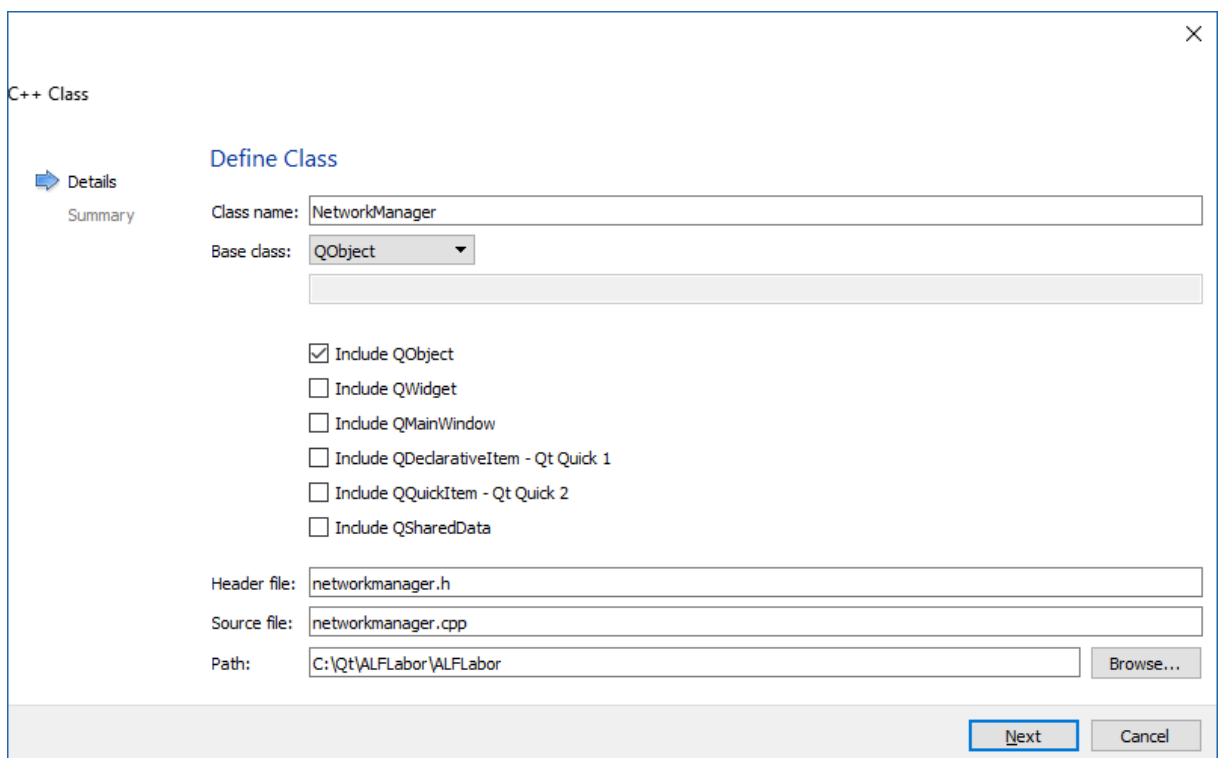
35. Végül adjuk hozzá *Text Editor* módban a következő *property alias*-okat:

```
Item{
    property alias title: title.text
    property alias subtitle: subtitle.text
    property alias imageSource: image.source
}
```

36. Ezek után hagyjuk magára UI-unkat és készítsük el a hálózati lekérdezést végző C++ kódunkat. Először is a **File** menüből válasszuk ki a „**New File or Project**” elemet, majd a fájlsablonok közül válasszuk ki a **C++**-t, azon belül pedig a **C++ Class**-t, majd lépünk tovább a „Choose...” gombbal.



37. Az osztály neve legyen **NetworkManager**, az ősztyály pedig legyen a **QObject**. A *Next* majd a *Finish* gombok megnyomásával véglegesítsük a folyamatot.



38. A **networkmanager.h**-ban először is include-oljuk a **QNetworkAccessManager** és a **QVariant** osztályokat.

```
#include <QObject>
#include <QNetworkAccessManager>
#include <QVariant>
```

39. Majd hozzunk létre egy **manager** nevű *QNetworkAccessManager* típusú változót. Ez az osztály képes hálózati kérések kiküldésére, illetve az ezekre érkező válaszok feldolgozására. Az alkalmazásunk ezen az osztályon keresztül fog HTTP protokollon keresztül üzenetet küldeni és fogadni egy távoli webszervernek/webszervertől.
40. Deklaráljunk egy **signal**-t *loginSuccess* névvel és három *QString* típusú paraméterrel sorrendben **imageSource**, **title**, **subtitle**.
41. Hozzunk létre egy publikus *slot* metódust **login** névvel, ami két *QVariant* típusú változót vár; név szerint **username** és **password**. Ez lesz az a metódus, ami majd a **Login** gomb megnyomására meghívódik.
42. Végül deklaráljunk egy *privát slot* metódust **requestFinished** névvel, ami egy *QNetworkReply** paramétert vár **reply** néven. Ez akkor fog lefutni, amikor **manager**-en keresztül elküldött a hálózati kérésünkre megérkezett a válasz. Mivel a **manager** az osztály tagváltozója, így a *slot*-nak nem kell publikusnak lennie.

```
class NetworkManager : public QObject
{
    Q_OBJECT
    QNetworkAccessManager manager;

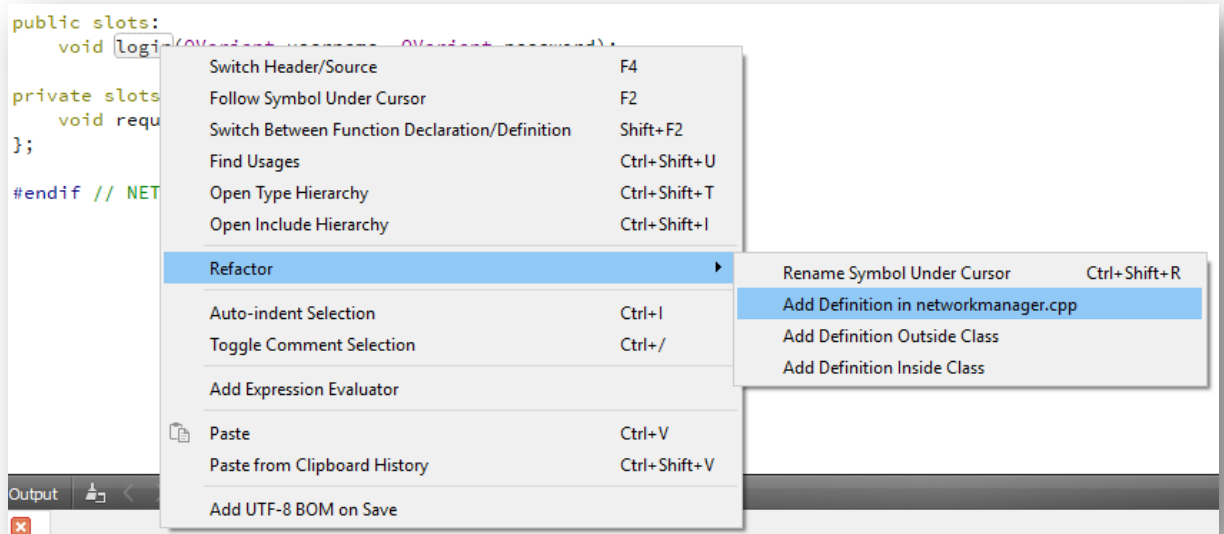
public:
    explicit NetworkManager(QObject *parent = nullptr);

signals:
    void loginSuccess(QString imageSource, QString title, QString
subtitle);

public slots:
    void login(QVariant username, QVariant password);

private slots:
    void requestFinished(QNetworkReply *reply);
};
```

43. Mielőtt átlépnénk a **networkmanager.cpp** állományba, generáltassuk le a **Qt Creator**-ral a *slot*-jaink törzsét. (A *signal*-nak nem kell törzsét generálni, hisz nincs is neki.) Ehhez mindössze le kell nyomni a jobb egérgombot az egyes metódusok nevein, és a **Refactor** menüpontnál ki kell választani az „**Add Definition in networkmanager.cpp**” lehetőséget.



44. Most, hogy már megvannak a függvényeink törzsei, töltsük meg őket. Először kössük össze a **manager finished** signal-ját a mi osztályunk **requestFinished** slot-jával rögtön a konstruktorban. Így fogunk majd értesülni a lekérdezés végéről.

```
NetworkManager::NetworkManager(QObject *parent) : QObject(parent)
{
    connect(&this->manager, SIGNAL(finished(QNetworkReply*)),
           this, SLOT(requestFinished(QNetworkReply*)));
}
```

45. Mielőtt implementálnánk a **login** slot-ot, „include-oljuk” a **QNetworkRequest** és a **QNetworkReply** osztályokat.

```
#include <QDebug>
#include <QNetworkRequest>
#include <QNetworkReply>
```

46. Most már rátérhetünk a **login**-ra.

A távoli szerveren ezen az url-en érhetjük el: <http://silent-sound-6780.getsandbox.com/login> egy egyszerű http GET üzenettel. (Ezt ki is lehet próbálni a böngészőben, mivel az is GET üzeneteket küld a webszervereknek.)

Hogy ne alapból „Failed” választ kapjunk, url paraméterekben el kell küldenünk a **username**-et és a **password**-öt. Pontosan azokat az adatokat, amiket a függvény megkap. Ehhez hozzuk létre az alábbi **urlString** névre hallgató **QString** típusú objektumot, amiben a behelyettesítendő paramétereket **%i** jelöli – ahol **'i'** egy index, ami 1-től indul. Ezeket a paramétereket a **QString arg** nevű metódusával lehet feltölteni.

Mivel a bejövő paraméterei a **login** metódusnak **QVariant**-ok, így ezeket ki kell csomagolni a **toString()** metódussal. Azért **QVariant**, mert a QML csak ilyet tud küldeni **signal**-okon keresztül (var).

```
QString urlString =
QString("http://silent-sound-6780.getsandbox.com/login?username=%1&password=%2").arg(username.toString()).arg(password.toString());
```

Ezt a jelszó-küldő megoldást éles, vagy ennél a labornál komolyabb feladatok esetén messze kerülni kell, mert nagyon veszélyes és egyáltalán nem biztonságos. Mindazonáltal most tökéletesen megfelel a céljainknak.

47. Azután, hogy elkészült a lekérdezés „szövege”, példányosítsunk egy **QNetworkRequest**-et az imént generált **urlString**-gel (amit átkonvertálunk **QUrl**-é, mert a **QNetworkRequest** **QUrl**-t vár.) és indítsunk a **manager** segítségével egy **get** kérést. és az alábbi módon:

```
QNetworkRequest request = QNetworkRequest(QUrl(urlString));
manager.get(request);
```

48. Most térjünk át a **request**-re adott válasz feldolgozására, írjuk meg a **requestFinished** metódus törzsét.
49. Először is olvassuk ki a **QNetworkReply** típusú bemenő paraméterből a szervertől kapott választ. Ezt a **readAll()** metódussal tehetjük meg, ami egy **QByteArray**-t ad vissza. A visszaadott értéket tároljuk egy lokális változóban, mivel szükségünk lesz rá.

```
QByteArray data = reply->readAll();
```

50. Ha szeretnénk, hogy az alkalmazás futás közben kiírja, hogy mit sikerül visszakapni használjuk a **QDebug** osztályt. Ehhez először is include-olni kell a **QDebug** osztályt.

```
#include <QDebug>
```

51. Majd az alábbi módon, tudjuk kiírni az átadott objektumokat.

```
QDebug() <<QString(data);
```

Ha nem vagyunk biztosak benne, hogy az objektumunk típusát támogatja-e a **QDebug**, konvertáljuk át **QString**-é.

52. A válasz formátumáról azt érdemes tudni, hogy nem megfelelő paraméterek esetében, azaz amikor nem tudjuk a jelszót, egy egyszerű „Failed” üzenetet kapunk. Ezzel szemben, ha sikerül az „autentikáció”, akkor egy JSON objektumot kapunk vissza a következő formátumban:

```
{
  "url" : ""
  "title" : ""
  "subtitle" : ""
}
```

Ha nem ismered a JSON formátumot, ezen a linken érdemes szétnézned:

<http://json.org/example.html>

53. Szerencsére, mivel a JSON nagyon elterjed formátum, nem nekünk kell egyenként kiszedegetni a szükséges adatokat, van rá támogatás a Qt-ban.
54. Hogy használni tudjuk a JSON támogatást, include-oljuk a **QJsonDocument** és **QJsonObject** osztályokat.

```
#include <QJsonDocument>
#include <QJsonObject>
```

55. Konvertáljuk a bejövő adatot **QJsonDocument**-té.

```
QJsonDocument jsonDoc = QJsonDocument::fromJson(data);
```

56. Mivel nem biztos, hogy értelmes JSON formátumú választ kaptunk, vizsgáljuk meg, hogy a keletkezett **jsonDoc** objektum nem üres-e az **isEmpty()** tagfüggvény meghívásával.

57. Sikeres ellenőrzés esetén a JSON üzenet egyetlen JSON objektumból áll, így egyszerűen konvertáljuk át a **jsonDoc**-ot **QJsonObject**-té és tároljuk el a kapott új objektumot a **rootObject** nevű **QJsonObject** típusú változóban.

58. Egy **QJsonObject**-ből az alábbi szintaxissal tudjuk lekérdezni az egyes kulcsokhoz tartozó értékeket:

```
rootObject["url"]
```

59. Figyeljünk arra, hogy a visszaadott értékek **QJsonValue** típusúak, amiket még **QString**-é kell konvertálni a **toString()** tagfüggvény segítségével, hogy emittálni tudjuk a **loginSuccess** *signal*-ban.

```
if(!jsonDoc.isEmpty())
{
    QJsonObject rootObject = jsonDoc.object();
    emit loginSuccess(rootObject["url"].toString(),
                     rootObject["title"].toString(),
                     rootObject["subtitle"].toString());
}
```

60. Ezzel el is készültünk a hálózati lekérdező osztályunkkal.

61. Mielőtt összekötnénk a két oldalt, térjünk vissza a **main.qml** fájlhoz és egészítsük ki a **SwipeView**-t egy **Connections** elemmel. Ez az elem teszi lehetővé, hogy a nem QML állományokban (azaz C++ oldalon) definiált objektumokat elérjük QML oldalon.

62. A **target** *property*-nek állítsuk be a **networkManager** nevet, majd implementáljuk a **loginSuccess** *signal*-nak megfelelő *slot*-ot, **onLoginSuccess:** néven. Ha nem pontosan ezt így írjuk, akkor nem fog működni.

The signal handler is named **on<SignalName>**,
with the first letter of the signal in uppercase.

63. Paraméterlistát nem kell írni, azokat automatikusan el tudjuk érni az eseménykezelőn belül, így csak értékül kell adni őket a **page2** megfelelő *property*-jeinek. Ezért kellett *property alias*-okat létrehozni, hiszen ezek nem a **Page2** *property*-jei, hanem az azon belül lévő elemeké.

```
Connections {
    target: networkManager
    onLoginSuccess: {
        swipeView.currentIndex = 1
        page2.imageSource = imageSource
        page2.title = title
        page2.subtitle = subtitle
    }
}
```



```
}
```

64. Végezetül állítsuk be a **Page1** elem **objectName** property-jét „**page1Object**”-re, hogy C++ oldalról is meg tudjuk találni.

```
Page1 {  
    objectName: "page1Object"  
}
```

65. Most térjünk vissza a C++ oldalra és menjünk át a *main.cpp* állományba és másoljuk be az alábbi két függvény-implementációt a main függvényünk fölé.

```
QObject* findItemByName(QList<QObject*> nodes, const QString& name)  
{  
    for(int i = 0; i < nodes.size(); i++)  
    {  
        // Node keresése  
        if (nodes.at(i) && nodes.at(i)->objectName() == name)  
        {  
            return nodes.at(i);  
        }  
        // Gyerekekben keresés  
        else if (nodes.at(i) && nodes.at(i)->children().size() > 0)  
        {  
            QObject* item = findItemByName(nodes.at(i)->children(),  
                                           name);  
            if (item)  
                return item;  
        }  
    }  
    return nullptr;  
}  
  
QObject* findItemByName(QObject *rootObject, const QString& name)  
{  
    Q_ASSERT(rootObject != nullptr);  
  
    if (rootObject->objectName() == name)  
    {  
        return (QObject*)rootObject;  
    }  
    return findItemByName(rootObject->children(), name);  
}
```

66. Hogy ne legyen baj a speciális karakterekből, a két metódust érdemesebb innen kimásolni:

<https://gist.github.com/blazovics/d12288867f1bf745612fe710233df219#file-main-cpp>

67. A main függvényen belül (lehetőleg a második *return* előtt) hozzunk létre egy példányt az imént definiált **NetworkManager** osztályunkból **myManager** néven.

```
NetworkManager *myManager = new NetworkManager();
```

68. Hogy ne kapjunk hibaüzenetet nyilván include-olnunk kell a **NetworkManager** header fájlját is. És ha már ott vagyunk egyúttal include-oljuk a **QQmlContext** osztályt is.

```
#include "networkmanager.h"  
#include <QQmlContext>
```

69. Most, hogy már van `NetworkManager` objektumunk, keressük meg a **Page1**-objektumunkat az újonnan implementált **findItemByName** függvényünkkel.

Ehhez tudnunk kell, hogy melyik a QML nézetünk `rootObject`-je. Aki arra tippel, hogy a **QQmlApplicationEngine** típusú `engine` nevű objektum tudja rá a választ, az jól tippelt. Mivel ez az objektum hozza létre a teljes QML-es ablakot, így tőle tudjuk elkérni a gyökér elemet, pontosabban a gyökér elemek tömbjét, amiből nekünk az első – és jelenleg egyetlen – elem kell.

A függvény második paraméteréül pedig adjuk át a `main.qml`-ben létrehozott **Page1** elemnél megadott **objectName** *property* értékét.

```
QObject *page1 = findItemByName(engine.rootObjects()[0], "page1Object");
```

70. Hogy le tudjuk kezelni a QML gombnyomás eseményét, és el tudjuk küldeni a megfelelő lekérdezést, kössük össze a **page1** objektum **loginPressed** *signal*-jét a **myManager** **login** *slot*-jával.

```
QObject::connect(page1, SIGNAL(loginPressed(QVariant, QVariant)),  
                 myManager, SLOT(login(QVariant, QVariant)));
```

71. Végezetül adjuk át a **myManager** objektumunk pointerét az `engine` `rootContext`-jének „**networkManager**” néven, mint *contextProperty*. Ez teszi lehetővé azt, hogy a QML oldalról is elérjük az objektumot.

```
engine.rootContext()->setContextProperty("networkManager", myManager);
```

72. Futtassuk az alkalmazást! A bejelentkezéshez szükséges adatok: alf/ALF