**FaceRecon**

**Rendszerterv**

Arcazonosítás alapú regisztráló- és beléptetőrendszer

Szoftverarchitektúrák tárgy házi feladat

Készítették:

Nandrean David Cristian

Rápolthy Bálint Endre

**A rendszer célja, funkciói és környezete**

**Részletes feladatkiírás**

A cél egy olyan szoftvercsomag lefejlesztése, amely egy eseményre (pl. fesztivál, színházi előadás) arcképes regisztrációt és beléptetést hivatott szolgálni.

Ezen szoftvercsomag neve *FaceRecon* és három részből áll:

• FaceReconRegister: egy szerver oldali frontend webalkalmazás, ami a regisztrációt teszi lehetővé,

• FaceReconDesktop: egy asztali alkalmazás mely arcfelismerőként működik az esemény helyszínén,

• FaceReconServer: egy backend alkalmazás mely az első kettőt szolgálja ki. Az asztali alkalmazás és a backend szoftver C++ programozási nyelven fog íródni.

**A rendszer által biztosított tipikus funckiók**

Vázlatosan az alábbi funkciókat várjuk el a rendszertől:

* **FaceReconRegister**
  + Név és e-mail szöveges mezőkre kitöltési lehetőség
  + Önarckép feltöltésére lehetőség
  + Beküldési gomb
  + Beküldésre adott válaszlehetőségek
    - Regisztráció elfogadva és konfirmációs e-mail kiküldve
    - Sikertelen regisztráció, mert a képen nem található arc
    - Sikertelen regisztráció, mert a képen egynél több arc található
    - Sikertelen regisztráció, mert az arc már meglelhető az adatbázisban
* **FaceReconDesktop**
  + Webkamera által szolgáltatott kép élő közvetítése
  + Webkamerán található arcok bekeretezése
  + Amennyiben a webkamera által szolgáltatott képen csak egy arc található több ideig (pl. 50 frame-en keresztül) kérés küldése a szervernek, hogy léptesse be az adott személyt
  + Szerver válaszlehetőségeinek kiírása:
    - A személy nem található az adatbázisban
    - A személy már belépett az eseményre
    - Személy azonossága (név, e-mail) és erre konfirmációs lehetőség
* **FaceReconServer**
  + Regisztrációs kérés fogadása és lekezelése a weboldaltól
    - Arckép összehasonlítása adatbázissal
    - Válasz visszaküldése
  + Beléptetési kérés fogadása és lekezelése
    - Arckép összehasonlítása adatbázissal
    - Válasz visszaküldése
  + Regisztrált arcképek tárolása
  + Adatbázis karbantartása
  + Kérések konkurrens kezelése

**A program környezete**

A szoftvercsomagot szerver-kliens alkalmazásként készítettük el, natívan, C++ programozási nyelven. Bár jelenleg a cél operációs rendszer a Windows, mivel a fordítási procedúra CMake-kel lett definiálva, ezért kevés módosítással könnyen lefordítható és futtatható más operációs rendszereken is.

A grafikus felület megvalósításához a *Qt* keretrendszert választottuk, a képek kezeléséhez, valamint az előre betanított neurális hálók használatához pedig az *OpenCV* könyvtárat. A webalkalmazás a Crow könyvtárat használja a webes funkcionalitások megvalósításához.

A függőségek menedzselését a *vcpkg*-val valósítottuk meg, jelenleg emiatt tudunk csak Windowsra fordítani.

Különböző konkurrens problémák megoldására, melyeket a klasszikus C++ szabvány jelenleg még nem támogat a *boost* könyvtárat választottuk.

A fordítás és telepítés procedúrák megkönnyítésének érdekében kisebb shell scripteket írtunk, melyek a megfelelő paraméterek beállításával automatikusan elvégzik ezeket a feladatokat.

**Megvalósítás**

Az alkalmazást a feladatkiírásnak megfelelően szoftvercsomagként készítettük el, három különálló szoftver, két natív C++ nyelven íródott alkalmazás és egy weboldal, melyek kliens-szerver architektúrát valósítanak meg.

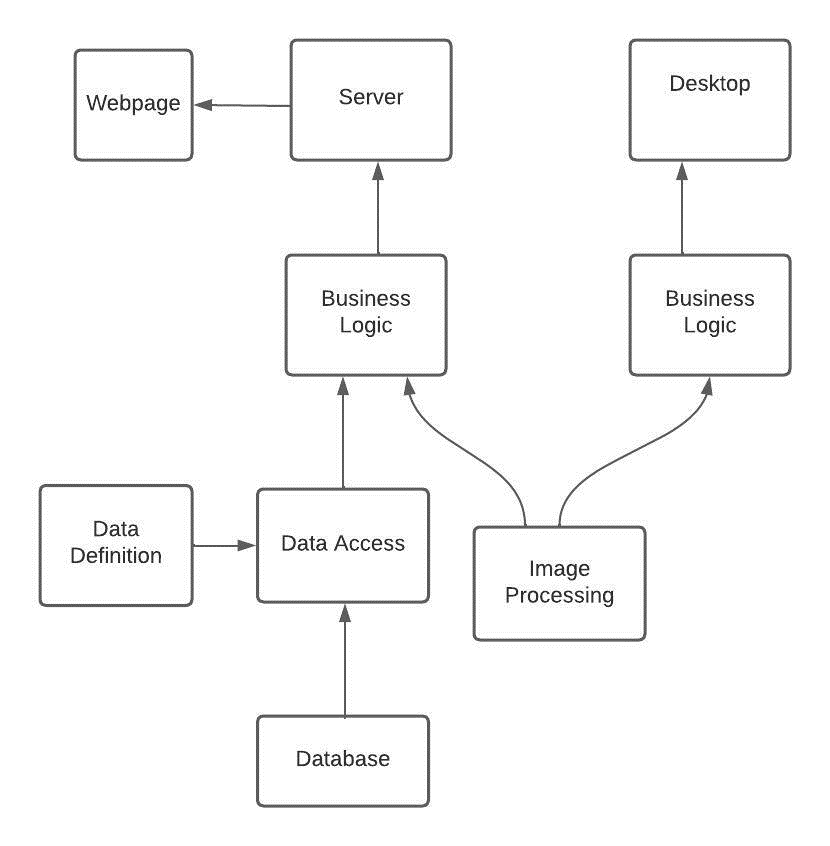
Az általunk elkészített programocsomagot FaceRecon névre kereszteltük (kiejtése: [feɪsrɪˈkɒn]), utalva arra, hogy a szoftvercsomag arcokat derít fel és az “ismerős” arcoknak megengedi a belépést.

**Architektúra**

A FaceRecon architektúrája 9 különálló modulra bontható:

* Adatbázisréteg (Database Layer, DB)
* Adatelérési réteg (Data Access Layer, DAL)
* Képfeldolgozó réteg (Image Processing Layer, IPL)
* GUI üzleti logika réteg (GUI Business Logic Layer, GBLL)
* Szerver üzleti logika réteg (Server Business Logic Layer, SBLL)
* Felhasználói felület (Graphical User Interface, GUI)
* Szerver (Server, SERVER)
* Weboldal (Webpage, WEB)
* Adatdefiníciók (Data Definitions, DATA)

Az egyes rétegek közötti kapcsolatot hivatott az alábbi ábra illusztrálni:



**Adatbázis réteg**

**Célja: Az adatok tárolására alkalmas**

Elegendő egy meglévő weboldal API-hoz hozzáférni, mivel:

* Saját készítésű különálló weboldal (más célra) és API elérést biztosítás PHP-val
* nethelyen biztosított tárhelyen alkalmazzuk az API-t
* Szabadon hozzá lehet férni az URL-hez, megfelelő paraméterezéssel
* Adatbázissal kapcsolatot létesít, nem kell hozzá lokális lekérdezés

Az adatbázis 1 táblából áll, ami a regisztrált felhasználók adatait tárolja el és azt is megmutatja, hogy melyik felhasználó jelentkezett már az eseményre sikeresen.

Név = Nevet tároljuk el, ami lehetőség szerint csak egy felhasználónév

Email = Email-t tároljuk el, ahhoz, hogy konformáljuk valódi létét

pic = Egy feltöltött önarckép, ami majd ellenőrzés alá kerül

is\_registered = Ha önarckép és kamerakép azonos, sikeres az eseményre való beléptetés

**Megjelenés a kódban:** A használt forrás egy kibérelt tárhelyről érkezik, ami JSON adatformátumban jelenik meg, az adatokat könnyedén fel lehet használni osztály szintű entitásokba helyezve.

**Adathozzáférési réteg (Data Access Layer)**

Célja: Az adathoz való hozzáférést egy felsőbb réteg biztosítja, amit elérhetünk api hívással.

Réteg feladatai:

* Új entitások beszúrása az adatbázisba
* Adathozzáférés biztosítása
* Módosítás az adatoknál

**Megjelenés a kódban:** Ez a réteg CURL hívás segítségével megvalósítja a eventshare.hu/v2.0/src/api/apiController.php linnken keresztül az adatbázis műveleteket.

**Paraméterek, amik kellenek az API híváshoz:**

query\_table=face\_recon\_event, tábla neve

query\_type=get,post,put (attül függ mit szeretnénk)

select (mezők kiválasztásához, mindent kiválaszt ha nincs egyenlőség)

where (opcionális)

value (Csak az új adat beszúrásához fontos)

Az adatokhoz úgy tudunk hozzáférni, hogy egy API által kérdezzük le az adatbázis tartalmát. A nethely.hu tárhelyen phpMyAdmin jellegű szerver fut. Az API-t url paraméterezés által tudjuk JSON formátumban visszakapni, amit majd CURL hívással intéznénk el.

Ez a felső réteg egy úgynevezett Data osztályt alkalmazz tárolásra:

* id (int)
* name (string)
* email (string)
* path (string)
* registered (bool)

Az ID kitöltése az utolsó adathoz képest eggyel nővekszik.

Ez a réteg lokális adatbázis fenntartásához szükséges, ami a felhőben tárolt adatoknak egy lokális verziója. Fontos, hogy mindig friss adatokkal dolgozzunk, mert eltérő lehet a felhőben tárolt adatok tárolása és egyeznie kell mielőtt múveleteket végzünk el.

PHP mysqli-t alkalmazz, amit az url adaton keresztül irányít tovább egy query felépítéséhez. Az adatbázis csatlakozást követően a query-t lefuttatjuk és egy asszociativ tomb segítségével visszaadjuk ha volt hiba és adat. Általában ha nincs hiba, akkor sikeresnek vesszük az adatbázis kérést.

**Képfeldolgozó réteg**

Célja: A képfeldolgozó algoritmusok (mi esetünkben neurális hálók) funkcionalitásainak egy közös interfészbe burkolása, melyet a Desktop alkalmazás és a Server alkalmazás is fel tud használni.

Ez mindkét alkalmazás által intenzíven használt réteg, hiszen az adatbázisban található minden kép, valamint a webkamera által szolgáltatott minden kép is ezen megy keresztül.

Négy különálló funkcionalitást szolgáltat melyek szorosan összefüggnek: az első a képeken az arcok azonosítása, második az azonosított arcok kivágása a képről, a harmadik a kivágott arcból egy *feature matrix* előállítása, a negyedik pedig két kép összehasonlítása *feature matrix* alapján.

A Desktop alkalmazás nem többszálú, ezért ezek mindig szekvenciálisan vannak hívva, így szálkezelési problémák nem léphetnek fel. Ezzel ellentétben a Server alkalmazás többszálú, ezért implementálni kellett ennek egy szálbiztos változatát is.

Megjelenés a kódban: a Server és a Desktop alkalmazás üzleti logikai rétege (GBLL és SBLL) által egyik legintenzívebben felhasznált rétege, így a kódban sok helyen jelenik meg.

**Szerver üzleti logika réteg**

Célja: A szerver rétegtől jövő hívások kiszolgálása, minden egyes hívás ezen a rétegen megy keresztül. Mivel a szerverhívások több szálon történnek, ezért fontos volt ennek a rétegnek szálbiztossá tétele. Mivel a képfeldolgozó réteg összehasonlító funkciójának minden egyes képhez egy feature mátrixra van szüksége, ezért a szerver indításakor ezeket minden adatbázisban lévő képre kiértékeljük és a hozzá tartozó személy adatait is számon tartjuk lokális memóriában, ezáltal gyorsítva az összehasonlító eljárást.

Három kérés létezik, melyet ez a réteg le kell kezeljen:

* Regisztráló kérések a weboldalról érkeznek, ezek esetében a base64-ben lévő képet dekódolni kell és ha pontosan egy arcot tartalmaz, azt kivágni és összehasonlítani az adatbázissal, majd lementeni a képet, adatbázisba (lokálisba is) beszúrni az új adatot és válaszolni a weboldalnak.
* Beléptető kérések a kliens alkalmazás(ok)ból érkeznek, ezek már a kliens által kivágott képek, így csak az összehasonlító eljárást kell elvégezni, választ küldeni a Desktopnak
* Megerősítő kérés: Desktopból jön, egyezés esetén a felhasználó megerősítheti személyazonosságát. Ez pedig maga után vonja az adatbázis frissítését.

**GUI üzleti logika réteg**

Célja: A grafikus GUI felület működésének logikáját írja le, felhasználva a képfeldolgozó réteget. Amennyiben 50 frame-en keresztül a képen egy arcot lát az utolsó képről kivágja az arcot és beléptető kérést küld a Servernek, majd annak válasza alapján, ha van egyezés akkor megerősítő kéréssel megerősítheti személyazonosságát.

**Server**

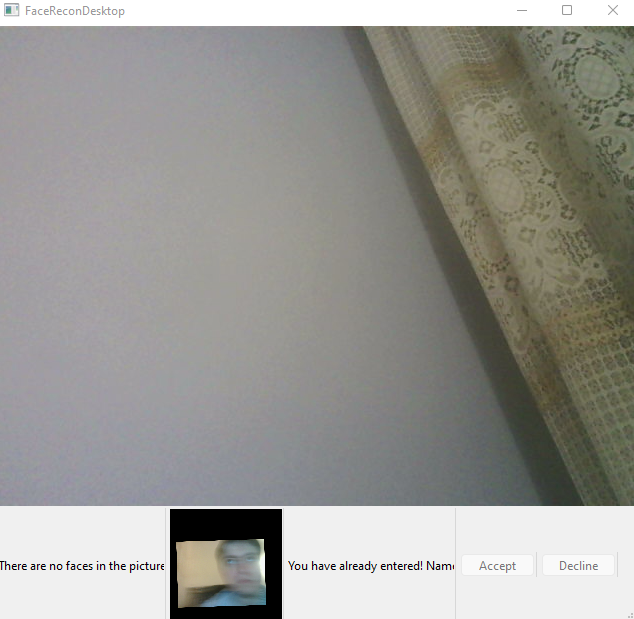
Célja: A HTML endpointok definiálása és az üzleti logika réteg felhasználásával a weboldal és a Desktop alkalmazások kiszolgálása.

Mindezt CURL hívásokkal és API-t felhasználva teszi meg.

**Grafikus felület**

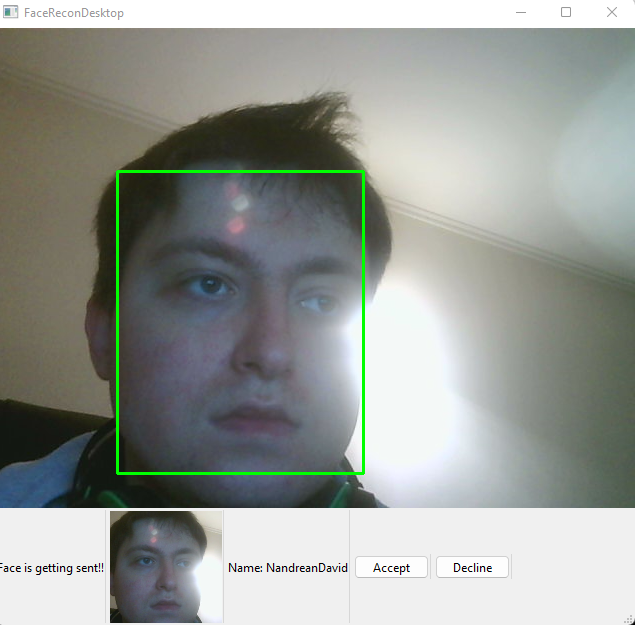
A grafikus felelet QT segítségével történik, ahol kameraképet láthatjuk az alkalmazás elindításakor. Az arcfelismerő rendszer az arcképen található feature-ok alapján számítja ki, a két kép közötti hasonlóságot.

Fontos kiemelni, hogy 3 állapotot ismer fel a Desktop alkalmazás. Ha nem arcot lát, akkor kiírja, hogy nincsen a képen arc és úgy nem lép érvényességbe a kép elküldése a szervernek.

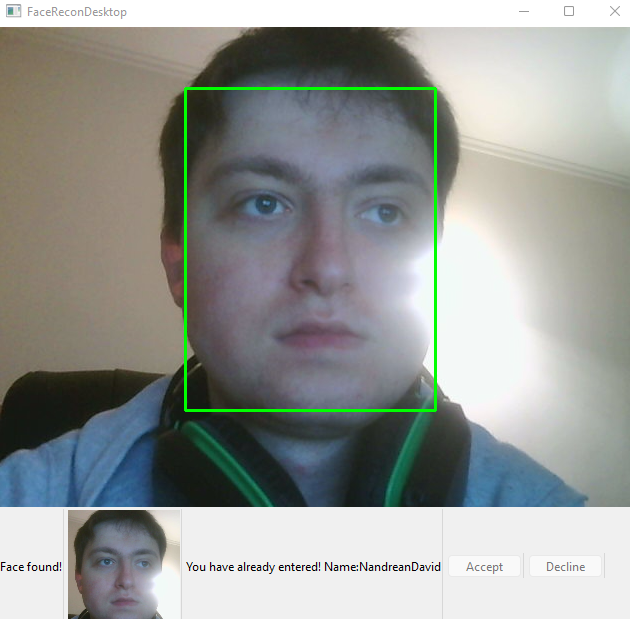


A kamerát felismeri a rendszer, akkor az arcfelismerő rendszer algoritmusa felismeri az arcot és kivágja, majd kikeresi kié az arckép. Ekkor a kép megnézi a lokális adatbázisban talált képeket és azt is, hogy kié az arc és mennyire felismerhető.

Elfogadni és elutasítani tudja, hogy jelentkezik-e az eseményre.



A másik eset, amikor már jelentkezett a felhasználó az eseményre és ezt az adatbázisból tudjuk meg, hogy a felhasználó már jelentkezett.



**Weboldal**

A szerver oldali alkalmazás FaceReconSever fogja biztosítani és felállítani egy localhoston futható port-on keresztül a weboldalt. A weboldal egy HTML file, ami funkcionalitásban Javascript-et használ, a képnek a tartalma kinyeréséhez. Az oldal kinézetéhez CSS-t alkalmazunk. A html form segítségével beállított GET kérést továbbít a szervernek beküldött paraméterezésekkel és kérésre egy választ kapunk.

Kettő fajta kérés lehet a szerver felé:

* /submit = Új felhasználó regisztrálásához, megvalósítja a kommunikációt a weboldal a szerverrel és mindezt továbbítva a backend részhez, ahol lokális adatokkal nézzük össze.
* /send = A desktop alkalmazza egy Mat kiterjesztésű file-t fog kéréssel küldeni a szervernek, amit a backendben nézzük meg, hogy a webkamerás és a beküldött eltárolt kép egyezik-e.

Mind a kettő esetben válaszol a weboldalhoz, hogy sikeres vagy nem sikeres a kérésünk, regisztrálás és jelentkezésre értve.

Mezők

* Név (kötelező)
* email (kötelező)
* kép feltöltés (kötelező)

Submit gombra kattintva átirányít minket a ’/submit’ path-ra. Ahol a képet elküldjük a backend szervernek és validálja a képet.

Kép megtekintő, amint a feltöltés utáni pillanatban látunk egy képkeretben.



**Adatdefiníciók**

Célja: Az adatbázisban tárolt adatstruktúra definiálása C++ nyelven elősegítve a lokális tárolást.

Az adatokat egy Data-ban tároljuk el.

* Adatentitások (pld. név, email, kép, jelentkezett-e)
* Lokális adatok frissítéséhez
* Fenntartjuk a felhőben biztosított adatbázis másolatát

A szerver kérés után lekérdezett adatokat eltároljuk egy osztályban, ami lokális adatbázisnak felel meg.

|  |  |  |
| --- | --- | --- |
| C++ mezőnév | C++ adattípus | MySQL adattípus |
| id | int | integer |
| name | string | varchar |
| email | string | varchar |
| pic | string | varchar |

**GUI-terv**

A Desktop alkalmazás fejlesztésekor az egyszerű, érthető felépítést tartottuk szem előtt, ennek megfelelően a program működését egyetlen ablak látja el.

A képernyő közepén a webkamera élő közvetítése, mely amennyiben arcokat érzékel bekeretezi azokat. Továbbá alatta az aktuális detekció állapota van leírva (nem látható arc a képen, túl sok arc van a képen vagy pont egy arc van a képen). Amennyiben a képen csak egy arc látható, úgy azt kivágva mutatja a státusz mellett.

Amennyiben beléptető kérést küld a Desktop és van egyezés, a két gomb (“Confirm” és “Decline”) klikkelhető állapotba kerülnek, ezáltal a felhasználó megerősítheti személyazonosságát.

**Telepítési leírás**

A programcsomag fejlesztése során hangsúly fektettünk a fordítás és telepítés leegyszerűsítésére, ezért két paraméterezhető scriptet írtunk, egyik elvégzi a fordítást, másik pedig programcsomagokat készít.

Szükséges előfeltételek:

* MSVC fordító
* vcpkg <https://github.com/microsoft/vcpkg>
* Qt6

Meghívásra példa:

*QT6\_DIR="/c/Qt/6.4.0/msvc2019\_64/" CMAKE\_BUILD\_TYPE=Release CMAKE\_TOOLCHAIN\_FILE=/vcpkg/scripts/buildsystems/vcpkg.cmake ./scripts/build.sh*

Csomagkészítésre példa:

*QT6\_DIR="/c/Qt/6.4.0/msvc2019\_64/" CMAKE\_BUILD\_TYPE=Release ./scripts/create\_package.sh*

Ezt követően a *package* folderbe keletkeznek a futtatni készen lévő programok.