barabás miklós XID1HW

parászka Levente miklós EYQQ5V

varga illés CXJZE7

SZOFTVERFEJLESZTÉSI TECHNOLÓGIÁK FEJLESZTŐI DOKUMENTUM

2021.12.16

TARTALOMJEGYZÉK

1. HASZNÁLT PROGRAMOK ………………………………… 2. oldal
2. KONFIGURÁCIÓK…………………………………………... 3. oldal
3. UML DIAGRAMMOK………………………………………. 4. oldal
4. FUNKCIONÁLIS ÉS NEM FUNKCIONÁLIS

KÖVETELMÉNYEK…………………………………………. 5. oldal

1. A PROGRAM FELÉPÍTÉSE ……………………………........ 6. oldal
2. KÓDOLÁSI SZABÁLYOK…………………………………... 26.oldal
3. FEJLESZTÉSI LEHETŐSÉGEK……………………………... 27. oldal

**HASZNÁLT PROGRAMOK**

A program készítéséhez csapatunk több programot is használt.

Mindennek az alapja a VisualStudio 2019 volt. Ebben hoztuk létre az osztályokat, az MVVM architektúrát, és itt írtuk meg a kódot. A tesztelések írása és azoknak a futtatása is ennek a programnak a keretében került megvalósításra.

Az adatbázis használatára a Microsoft MySQL 2018 nevű programot használtuk. Itt hoztuk létre mind a felhasználók beléptetéséhez szükséges táblákat, mind pedig a többi adat tárolásához szükséges táblákat.

A csapat a kód könnyebb megosztására használta a GITHUB nevű webalkalmazást. Ide töltöttük fel a kész kódrészleteket, innen töltöttük le, hogy több gépen is tudjuk tesztelni a programot.

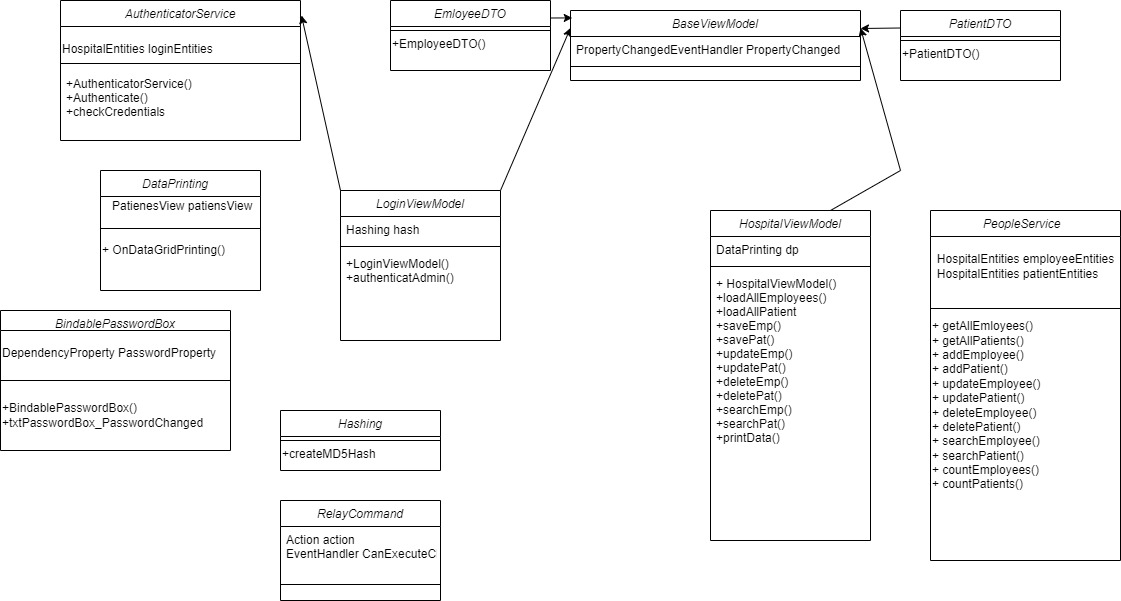
**KONFIGURÁCIÓK**

A csapat 3 tagja mind más más konfigurációjú laptopot illetve PC-t használ. Ez nagy előnyt jelentett a fejlesztések és a tesztek alatt is, mivel így több fajta konfiguráción tudtuk tesztelni a programunkat.

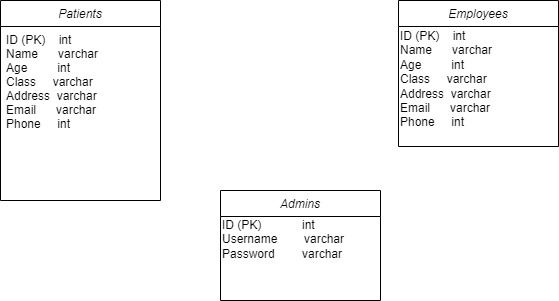
Ezek a következők voltak:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tulajdonos | CPU | Memória | Alaplap | Videókártya | Operációs rendszer |
| Barabás Miklós  (PC) | i5-4460 | 8GB DDR3 | ASUS B85M-E | GTX970 | WINDOWS 11 |
| Barabás Miklós  (laptop) | i5-8250u | 8GB DDR4 |  | GeForce MX150 | WINDOWS 10 |
| Parászka Levente Miklós (PC) | AMD Ryzen 7 2700X | 16GB DDR4 | B450M PRO-VDH MAX (MS-7A38) | GTX1660 | WINDOWS 10 |
| Parászka Levente Miklós (laptop) | i7-7500U | 8 GB DDR4 |  | GTX 950M | WINDOWS 10 |
| Varga Illés | i5-4460 | 8GB | H97 PC MATE | MSI R7 260X | WINDOWS 10 |

**UML DIAGRAMMOK**

****

*Osztályok UML diagram*

****

*Adatbázis UML diagram*

**FUNKCIONÁLIS ÉS NEM FUNKCIONÁLIS KÖVETELMÉNYEK**

Funkcionális követelmények:

* Legyen egy bejelentkezési felület
* Tároljon adatokat, amelyeket lehessen később olvasni, módosítani, törölni, vagy esetleg ki lehessen nyomtatni
* Könnyen lehessen használni, kezdő felhasználó is tudjon boldogulni a kezeléssel

Nem funkcionális követelmények

* Kis méretű legyen a program, ne legyen 10-20Gb
* Az adatok legyenek legalább egy hasheléssel védve, de jó, ha több dolog is van
* A fejlesztés során lehessen módosítani, ha a csapat úgy dönt anélkül, hogy a teljes kódot kukázni kéne
* Windows alatt fusson a program
* Ne jelentsen problémát, ha nem a legújabb processzor van a számítógépben

**A PROGRAM FELÉPÍTÉSE**

A program az MVVM architektúra alapján készült. Ennek az a lényege, hogy tevékenységet teljesen elválassza a megjelenítéstől.

Ez az architektúra három részre osztható szét.

A Modell tartalmazza a program logikai elemeit, a Nézet a felület vezérlőit. A nézetmodell az összekötő a Modell és a Nézet között.

A programunk is így épül fel, ahogy az látható is.

A képen szöveg látható

Automatikusan generált leírásA képen szöveg látható

Automatikusan generált leírás

*BindablePasswordBox osztály*

Ez az osztály lehetővé teszi azt, hogy a jelszó beírásakor ne látszódjon a beírt szöveg, ki legyen takarva.

A program, miután elkezdjük gépelni a jelszót, ellenőrzi, hogy a txtPasswordBox-ban van-e a szöveg, és ha igen, akkor kitakarja.

A képen szöveg látható

Automatikusan generált leírás

**MODELL OSZTÁLYOK**

*AuthenticatorService osztály*

Az AuthenticatorService osztály a bejelentkezés ellenőrzésért felelős. Adatokat kap a LoginViewModel osztálytól, amikor a felhasználó beírja a bejelentkezési adatait (pl.: admin admin). A megkapott adatok alapján az AuthenticatorService osztály ellenőrzi, hogy helyesek-e a kapott adatok. Ezt úgy teszi meg, hogy végig ellenőrzi az adatbázis megfelelő tábláját. Ha talál egyező adatot, akkor a bool változó true értéket kap, ha viszont nem, akkor pedig false értéket. Ezután a bool változó értéket visszaküldi a LoginViewModel osztálynak.

A képen szöveg látható

Automatikusan generált leírás

*DataPrinting osztály*

A DataPrinting osztály felelős azért, hogy a Páciensek menüfül alatt található adatokat ki lehessen nyomtatni. Az OnDataGridPrinting metódusban először létre hozzuk a fájl nevet, ebben az esetben Lista lesz a név.

Utána a program megjeleníti a Nyomtatásbeállítást, a PrintDialog-ot. Ha ez megjelenik, tehát az érték true, akkor a méretet a pixelben megadott méretre állítja (ebben az esetben A4-es méretre), majd beállítja, hogy a tájolása fekvő legyen, végül pedig kinyomtatja.

A képen szöveg látható

Automatikusan generált leírás

*EmployeeDTO osztály*

Az EmployeeDTO osztály öröklődik a BaseViewModel osztályból. Lekéri az adatbázisból adatokat, annak megfelelően, hogy az adott változó milyen típusú (pl: ID-> int, Name-> string).

A képen szöveg látható

Automatikusan generált leírás

*Hashing*

A Hashing osztály felelős azért, hogy a jelszavakat biztonságosan tároljuk. Az osztály a mentett jelszavakat először encoding-olja, majd a StringBuilder összerakja. Az for ciklusban pedig megtörténik a hashelés.

Így nem egyszerű planetext-ben lesznek elmentve az adatok, hanem MD5-ben.

A képen szöveg látható

Automatikusan generált leírás

*PatientDTO osztály*

Az EmployeeDTO osztály öröklődik a BaseViewModel osztályból. Lekéri az adatbázisból adatokat, annak megfelelően, hogy az adott változó milyen típusú (pl: ID-> int, Name-> string).

A képen szöveg látható

Automatikusan generált leírás

*PeopleService osztály*

A PeopleService osztály látja el a feladatok nagy részét, ennek az osztálynak köszönhetjük szinte teljes egészében azt, amit a View-ban látunk.

Ez az osztály mind a Páciensek, mind pedig az Alkalmazottak menüpontban megjelenő adatokért felel.

Létrehozunk egy listát, amiben az alkalmazottak adatait fogjuk tárolni. Ehhez hozzáadjuk az adatbázisban már meglévő adatokat sorban, ahogy az adatbázisban is szerepelnek az adatok.

Ha valamilyen problémát észlel a program, akkor kivételt dob.

A képen szöveg látható

Automatikusan generált leírás

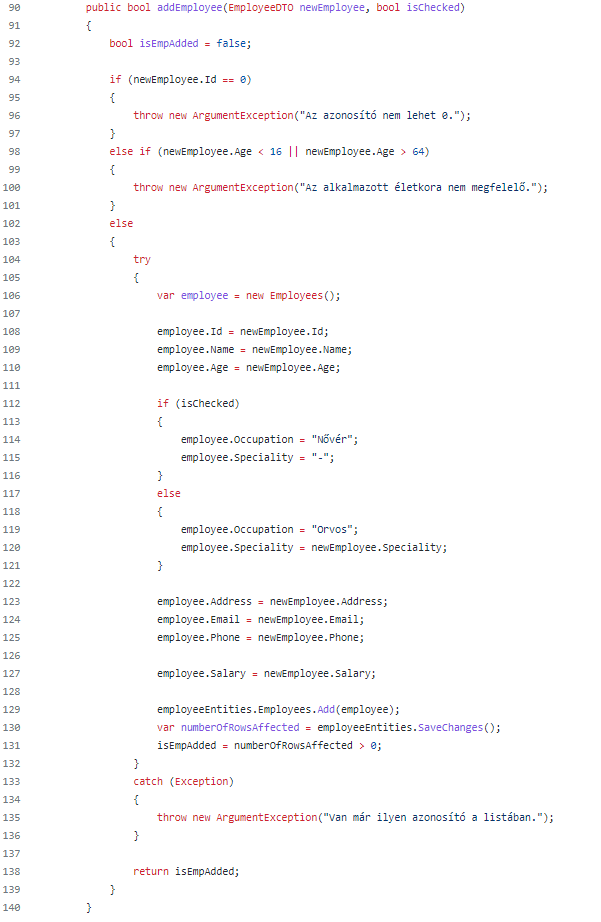
Ezután ugyan így létrehozunk egy páciens listát is.

A képen szöveg látható

Automatikusan generált leírás

Ezután következik a hozzáadás. Először alkalmazottat adunk a listához, utána pácienst. Először is false-ra állítjuk a bool változót, utána következnek az ellenőrzése. Megnézzük, hogy a megadott ID nagyobb e mint nulla, ellenkező esetben hibaüzenetet kapunk. Utána ellenőrizzük az alkalmazotthoz megadott életkort, aminek nagyobbnak kell lennie, mint 16, de kisebbnek, mint 64. Ellenkező esetben ismét hiba üzenetet kapunk.

A try-ban láthatjuk, hogy szintén van egy if-else ág. Ez vizsgálja meg, hogy az alkalmazott Nővér, vagy pedig Orvos e. A megadott adatokat elmentjük az adatbázisban, ami egyből meg is jelenik az alkalmazásban.



Ugyan így létre hozunk pácienst is, azzal a különbséggel, hogy a try-ban nem lesz if-else ág.

A képen szöveg látható

Automatikusan generált leírás

Ezután következik a meglévő alkalmazottak illetve a meglévő pácienseknek adatainak frissítése.

Itt is ellenőrizzük, hogy az ID nagyobb legyen, mint nulla. Ellenőrizzük, hogy az adott adatokban történt e változás. Ha történt, tehát a megváltoztatott adatok száma, nagyobb mint nulla, akkor frissíteni fog a program, ha nulla, nem fog csinálni semmit.

Ugyan ezen logika mentén történik a páciens adatainak frissítése.

A frissítés után következik a törlés. A törlés ID szám megadásával történik. Itt is ellenőrzi a program, hogy a szám nem lehet nulla, utána pedig kikeresi az adatbázisból a megfelelő ID számú rekordot, majd elvégzi a törlést. Ahogy eddig, itt is ugyan azon elven működik a páciens és az alkalmazott törlése.

A képen szöveg látható

Automatikusan generált leírás

Az előzővel ellentétben a keresés nem ID, hanem név alapján történik. Amennyiben a név mező nincs kitöltve, hiba üzenetet kapunk. Ha ki van töltve, a program az adatbázisban megkeresi, hogy van-e a keresésnek megfelelő eredmény. Ha igen, betölti a keresett személy adatait.

A képen szöveg látható

Automatikusan generált leírás

Az osztály utolsó funkciójaként pedig megszámolja az adatbázisban szereplő tábla sorait, és azokat számolja össze. Ezt a számot pedig a Főoldalon kiírja.

A képen szöveg látható

Automatikusan generált leírás

**NÉZETMODELL OSZTÁLYOK**

*BaseViewModel osztály*

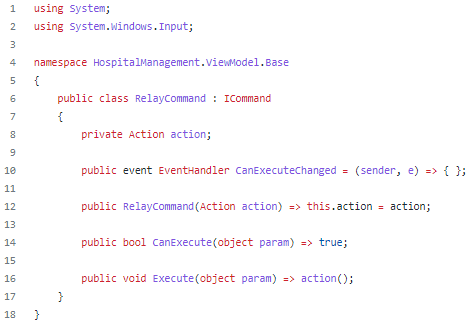
A BaseViesModel ősosztály biztosítja az INotifyPropertyChanged interfész megvalósítását a ViewModel osztályoknak. Amennyiben a kódban változás történik, leköveti a változásokat.

A képen szöveg látható

Automatikusan generált leírás

*RelayCommand osztály*

Megkönnyíti a kód olvashatóságát és rövidít a hosszán, mivel ennek az osztálynak köszönhetően, nem kell külön osztályokat létrehozni minden egyes parancshoz.



*HospitalViewModel*

Ez az osztály öröklődik a BaseViewModel osztályból. A RelayCommand osztálynak köszönhetően a gombok végre fogják hajtani a megírt parancsokat, nem kell nekik külön osztályokat létrehozni. A gombok megnyomása után, a parancs egyből végre fog hajtódni, az adatbázis frissül és megjelenik az alkalmazásban.

A képen szöveg látható

Automatikusan generált leírás A képen asztal látható

Automatikusan generált leírás

Ebben az osztályban találhatóak meg azok a parancsok, amelyek megtalálhatóak a PeopleService osztályban is.

Először az alkalmazott/páciens adatainak a mentése. Attól függően, hogy a mentés sikeres vagy pedig sikertelen volt, kapunk visszajelzésül üzenet.

A képen szöveg látható

Automatikusan generált leírás

A mentés után az update történik. Egy bool típusú változó ellenőrzi, hogy történt e változás és hogy az sikeres volt-e, majd szintén kapunk egy üzenetet.

A képen szöveg látható

Automatikusan generált leírás

A frissítés után jön a törlés. Az ID alapján megtörténik a keresés, majd pedig a törlés. Itt is sikeresség vagy sikertelenség következtében kapunk üzenetben visszajelzést.

A képen szöveg látható

Automatikusan generált leírás

A törlés után jön a keresés. A keresés lefutása után, amennyiben van találat, akkor kiírja a beteg adatait, ha nincs, akkor hibaüzenetet kapunk.

A képen szöveg látható

Automatikusan generált leírás

*LoginViewModel osztály*

Begyűjti az adatokat a bejelentkezési felületről. Ezután továbbítja az AuthenticatorService osztály számára, ahol megtörténik a vizsgálat. A vizsgálat eredményét (true vagy false) formában visszakapja a LoginViewModel osztály. Ha sikeres az autentikáció, belépteti a felhasználót, ha viszont nem, akkor hiba üzenet dob.

A képen szöveg látható

Automatikusan generált leírásA képen szöveg látható

Automatikusan generált leírás

**KÓDOLÁSI SZABÁLYOK**

A kódoláskor több szabályt is figyelembe vettünk:

* tagolt program: a program írásakor felügyeltünk arra, hogy minél átláthabb legyen a kód, hogyha a csapat bármely tagja belenéz, vagy beleír, tudja folytatni esetleg a másik gondolatmenetét és elképzelését. Fontos volt, hogy egy esetleges továbbfejlesztés esetén ne kelljen egész osztályokat kukázni vagy újraírni azért, mert nem tudjuk, mit miért csináltunk
* megfelelő nevek használat: mind az osztályok, mind a változók létrehozásakor szempont volt, hogy hangzatos neveket adjunk, lehetőleg olyanokat amik nem hasonlítanak egymásra, nem keverhetőek össze. Például ne legyen olyan, hogy: „data, datas, Data” stb.
* CamelCase: előre egyeztettünk, hogy az írásforma kövesse a CamelCase szabályt.
* Egységes nyelv: az egyszerűség kedvéért minden változó, osztály, adatbázis elem (adatbázis név, táblák, oszlopok) angol elnevezést kaptak, így véletlenül se került egyikbe sem ékezetes betű, ami később gondot okozhatott volna.

Ugyan nem kódolási szabály, de ügyeltünk arra, hogy minden döntés egyhangú legyen. Ha valakinek volt valamilyen ötlete, azt bedobta, megvitattuk, pro és kontra érvekkel, majd döntöttünk róla. Így elkerültük a külön utakon járást.

**FEJLESZTÉSI LEHETŐSÉGEK**

A programot elég sok féle terület felé lehetne továbbfejleszteni. Ezeket a területeket csoportokra lehet bontani:

* Egyéni felhasználás:
  + időpontfoglalás: a felhasználó időpontot tud foglalni. Lényege az lenne, hogy nem csak egy adott kórháznál vagy cégnél lehetne időpontot foglalni, hanem országosan bárhol
  + orvos értékelése: az adott vizsgálat után az orvos értékelése előre megadott szempontok alapján (pl: segítőkészség, modor, időpont betartás) és egyéni (le lehetne írni, amit a felhasználó pluszban gondol)
  + leletek lekérése: a felhasználó vissza tudná nézni, hogy mikor hol volt vizsgálaton, és ahhoz tartozó leleteket, vagy orvosi véleményt le tudja tölteni, vissza tudja nézni
* Vállalati:
  + egy cég ezen keresztül tudja rendezni az orvosok és egyéb beosztottnak a beosztását
  + egy időpontfoglalást ezen keresztül lehet elfogadni vagy lemondani a cég részéről, amennyiben a vizsgáló orvos beteg lesz vagy valami miatt nem ér rá
  + beosztás mellett admin szinten lehetne ezen keresztül utalni a béreket
  + vállalati hírlevél mehetne róla

A fejlesztések lehetőségei szinte végtelenek, de egy ilyen kis fős csoportban ezek kevésbé megvalósíthatóak, mind a kis létszám miatt, mind pedig amiatt, hogy egyikőnk se szerzett hatalmas tapasztalatot vállalati környezetben való fejlesztésben.