# **BEVEZETÉS**

Záródolgozatom témáját korábbi munkaköröm ihlette, annak rendkívül sokat tapasztalataiból merítettem. Korábban a vendéglátóiparban tevékenykedtem, egy kisebb panzió alkalmazott recepciósként, mely munka során elengedhetetlen és elkerülhetetlen volt az adminisztrációs célra szánt szoftverek napi szintű használata.

Az teljesen nyilvánvaló valamennyi ember számára, hogy a mai világban már minden létező problémára készült szoftveres megoldás. Magától értetődően léteznek a piacon korszerű, minden igényt kielégítő szálláshelyi információs rendszerek: számlázók, szobanyilvántartó programcsomagok. Ezen szoftverek egy része havi előfizetés keretein belül birtokolhatók, vagy szoftverlicensz alapján egyszeri áron megvásárolható alkalmazások. Irodai, adminisztratív szoftverek rendre nagyobb költségeket jelentenek egy-egy vállalkozás számára, sok kisebb szálláshely pedig nem engedheti meg magának őket, mivel a vállalkozás méretéből adódóan nincs rá égető szükség, vagy a vezetés nem tartja fontosnak a megfelelő, modern szoftver- és abból következő munkakörnyezet megteremtését alkalmazottai számára.

Esetemben az utóbbi helyzet állt fent. A vállalkozás maga éppenséggel nem volt multinacionális vállalat, de kisvállalkozás sem, hogy ne szorult volna rá egy megfelelő, akár kisebb vállalatirányítási információs rendszerre, hogy a napi adminisztráció alkalmával járó adatok (pl. vendégadatok kezelése) és információk (pl. foglalások felvitele adott időpontra) feldolgozása és kezelése ne jelenthessen többletmunkát, és maguk a folyamatok lebonyolítása minél gördülékenyebben történhessen meg.

A fentebb említett szálláshely teljesen átlagos, néhol igencsak „fapados” megoldásokat alkalmazott az adminisztrációs folyamatok bonyolítására és a feladatok megoldására:

* Microsoft Office Excel 2003 táblázatkezelő alkalmazást a foglalások időpontjainak – annak vizuális megjelenítéséből (pl. minden foglalás külön színnel ellátva, megjegyzések hozzáadása cellákhoz, illetve egyéb Excel lehetőségek alkalmazása) fakadó átláthatóság szerinti vezetésére,
* Microsoft Office Word 2003 szövegszerkesztő alkalmazást a foglalások dokumentálását szöveges formában és egyéb, nem vizuális adatok feljegyzése céljából (afféle foglalási napló),
* papír alapú vendég bejelentkező lap vezetése és papír alapon a foglalás részleteit (vendégszám, fizetés módja, napi ár) részletesen tartalmazó ívek vezetése
* Külön megvásárolt számlázóprogram(ok) alkalmazása, amely ugyan tartalmazott beépített vendégadatbázist – megadva a lehetőséget azon törzsvendégek személyes adatainak kezelésére, számlára importálására a későbbiek újra felhasználhatóságának igényét kielégítve –, de alapvetően mindig papír alapú adathalmazból került felvitelre minden újabb vendég adata.

Mondhatni embert próbáló feladat volt megtanulni ezeket a helyi specialitásokat, valamint a későbbiekben dolgozni is benne. Rengeteg más feladattól elvette az értékes munkaidőt és – véleményem szerint – rengeteg felesleges lépést tartalmazott. Tipikus példa erre az adatok többszörös, avagy redundáns eltárolása – papíralapon –, amely még adatok hatékony visszakeresését sem tette lehetővé, hiszen ha szűrni akartunk volna egy adott vendég adott foglalásának időpontjában a szobaszámra és a vendégszámra, a számlázó a személyes adatokon és a számlakiállítás dátumán kívül más hasznos információt nem tartalmazott, így erre az esetre az Excel táblában lévő időpont oszlopban kellett kikeresni a foglalást, esetlegesen a szöveges foglalási naplóhoz folyamodni plusz információhoz.

Ezen tapasztalatok összességéből körvonalazódott ki és született meg a *Virtual Receptionist* ötlete. Egy korszerű, hatékony, egyszerűen használható, integrált komplex „mikro vállalatirányítási rendszer” – kifejezetten kisvállalkozásban működő szálláshelyek számára.

Egy olyan szoftverrendszer, amely ezen folyamatokat oly módon kívánja megvalósítani, hogy általános szálláshelyi komponenseket (vendégadatbázis, számlázás, foglalási napló) próbálja minél egyszerűbb és egységesebb formába önteni, azon műveletekkel járó procedúrákat leegyszerűsíteni, érthetővé tenni, a célnak megfelelő eredményt produkálva. Ezeket mind úgy megvalósítva, hogy a rendszer szolgáltatásait a recepciós, illetve bármely vállalaton belüli informatikai alaptudással rendelkező személy könnyedén használhassa.

Jelen esetben igyekeztem minél általánosabb, absztraktabb funkciókat megvalósítani, de néhol elengedhetetlen volt, hogy egyedi igények szerint szabjam meg egy-egy funkció implementációját. Egy képzelt szálláshely, az Aranyerdő Vendégház képére formálva jött létre a kezdeti verziója – egyfajta prototípus gyanánt – egy webalapú és egy asztali alkalmazásnak, amely egy közös adatbázison keresztül ismerik tudnak közvetetten kommunikálni. Ismerik a konfigurált adatokat és azokkal dolgozhatnak meghatározott feladatokon, meghatározott hatáskörrel, amelyet az alábbi felsorolás taglal:

1. Webalkalmazás: a szálláshely belépése a rendszerbe ezen a konfigurációs felületen történik egy regisztrációval, azonosítással (autentikációval). Szálláshely alapvető tulajdonságainak testre szabása (szálláshely cégadatai: neve, székhelye, adószáma stb.; számlázási tételek beállítása, szobák beállítása: neve, kapacitása, kategóriája stb.) és későbbi konfigurálhatósága felhasználó által, amely alapján a szálláshely működni képes és ismeri önmagát a rendszerben.
2. Asztali alkalmazás: magát a szálláshelyi adminisztrációt, azaz a sok kisebb összeadódó folyamatok kezelését végző felület, amely magába foglal egy egyszerű számlázó, vendégadatbázis és foglalási napló modult.
3. Adatbázis: a szálláshely számára nélkülözhetetlen adatok perzisztens tárolásának céljából, amelyet elér a webes és asztali alkalmazás adat lekérdezés, adat manipulációs műveletek és közvetett kommunikáció céljából.

Ezen az elveken vezérelve a rendszer megtervezése abszolút felhasználó központú tervezést igényelt. Mivel esetemben egy képzelt szálláshely számára készült a rendszer, úgy nagyrészt korábbi tapasztalataimban bízhattam ismételten, illetve azokból a munkatapasztalatokból, amely a fentebb részletezett problémákat vázolta fel, így tudtam mi hasznos és mi nem, illetve mire van szükség. Azonban éltem a lehetőséggel és vendéglátóipari személyes kapcsolataimat felhasználva egy másik, létező szálláshely tulajdonosával végeztem követelmény feltárást specifikáció gyanánt szerepjáték módszer segítségével meg, egy felhasználói interjú elvégzését „projektmenedzser-fejlesztőként” a „megrendelő”-vel. Ezen interjú rengeteg egyéb más ötletet adott a rendszer megtervezéséhez és megvalósításához, lényegében csak hasznos tanácsokkal gazdagodtam. Az interjú kérdéseit és válaszait egy dokumentumban rögzítettem, amely a záródolgozat mellékletében teljes formájában helyet foglal. Adatmodellezéshez környezeti modellt, UML[[1]](#footnote-1) használati-eset diagram vázolja a rendszer funkcionális követelményeit (*Melléklet, 1. ábra*), az adatbázis megtervezéséhez pedig EK-diagram[[2]](#footnote-2) elkészítését alkalmaztam (*Melléklet, 2. ábra*). Készítettem pluszban a grafikus felületeket vázlatosan megjelenítő képernyőterveket (*Melléklet, 3. ábra*).

Minden szükséges plusz információ megszerzése után nekiláttam a komplex szoftver megvalósításához, amely megvalósításának részletes menetét a záródolgozat fejlesztői dokumentációja tartalmazza. Nagyvonalakban a megoldásokról: az adatbázis megvalósításhoz relációs adatmodellt alkalmazva, egy korszerű, kis és közepes alkalmazásokhoz tökéletes, az Oracle által ingyenesen használható MySQL adatbázist valósítottam meg az EK-diagram relációsémává képzésével, melyhez az Apache jóvoltából ingyenesen letölthető XAMPP programcsomag részét képező phpMyAdmin webes RDBMS[[3]](#footnote-3) nyújtott segítségét. A webalkalmazást HTML, CSS, JavaScript és PHP jelölő- illetve szkript nyelvekkel valósítottam meg, külső ingyenes API-t is felhasználva (jQuery, Bootstrap 4, PopperJS stb.). Az asztali alkalmazásom nagyobb funkcionalitását és robosztus tartalmát tekintve többrétegű architektúrában került megvalósításra a Microsoft .NET Frameworkben megtalálható, Windows platformra hagyományos grafikus alkalmazások készítését lehetővé C# magas szintű, tisztán objektum-orientált programozási nyelvvel történő fejlesztéssel. Az alkalmazások fejlesztéséhez további segítséget nyújtott, hogy alkalmazkodva a modern szoftverfejlesztési trendeket követve verziókövető rendszer segítségével fejlesztettem ki. Azon belül is a GitHub alkalmazással webes kezelőfelületével és az asztali grafikus verziókövető menedzser alkalmazással, amely igencsak gördülékennyé és hatékonnyá tette a projektek elkészítését.

Mivel az rendszer webalkalmazása közös W3C[[4]](#footnote-4) szabványok által ajánlott nyelvekben történt, tekinthető cross-platform[[5]](#footnote-5) alkalmazásnak (operációs rendszer tekintetében), amely komponenseinek megjelenítésében csak az internetes böngészők szabhatnak határt; a C# nyelvben íródott asztali alkalmazás Windows platformon történő használatát teszi lehetővé, natív alkalmazás[[6]](#footnote-6) formájában. Így az ajánlott használati szoftverkörnyezet a Microsoft Windows 7, vagy afölött elérhető valamely operációs rendszerek ajánlottak, illetve egyes komponensek működését garantálva a .NET Framework környezet telepítésével. Hardverkörnyezet tekintetében a felhasznált technológiák tekintetében minimum 2 GB operatív tár, illetve valamely 10 évnél nem régebbi processzor használata igényelt, egy alap videókártya típussal.

A továbbiakban a fejlesztői és felhasználói dokumentáció részleteit ismerteti a záródolgozat.

# **FEJLESZTŐI DOKUMENTÁCIÓ**

## **1.1 ADATBÁZIS**

A Virtual Receptionist magját az adatbázis adja. A korábbiakban említettem itt tárolódik minden fontos adat, amely az alkalmazások számára nélkülözhetetlenek (pl. szálláshely adatok, szálláshely felhasználói fiókjához tartozó szálláshelyazonosító és jelszó páros, vendégek adatai stb.), valamint ezen keresztül tud közvetetten kommunikálni a web- és asztali alkalmazás.

Az adatbázis mindkét alkalmazásnak szállít szükséges adatot, bizonyos adatait a web- és az asztali manipulálhatja is, de magát az adatbázis sémát egyik sem. Az adatok manipulációját kifejtő hatásköröket a web és asztali alkalmazás alfejezetek részletezik.

Az adatbázis XAMPP alkalmazás segítségével készült el, amely magában foglalja localhost környezetben a phpMyAdmin RDBMS-t. Ez egy egyszerű, grafikus webalkalmazás, amely segítségével SQL[[7]](#footnote-7) utasítások begépelésével vagy grafikusan, menü vezérelt funkciókkal létrehozhatók egyszerű MySQL relációs adatbázisok. Ezen módon történő adatbázis eltárolás egyszerűbbé teszi az alkalmazásfejlesztéshez kifejlesztendő adatbázis használatát a fejlesztési és tesztelési szakaszban, mivel nem kell élő szervert üzemeltetnünk ehhez, hanem az említett localhoston (127.0.0.1[[8]](#footnote-8) IP-címen), helyi hálózaton belül.

Az EK-diagram relációsémává leképzése után én is ebben az RDBMS-ben hoztam létre a Virtual Receptionist adatbázisát, *virtual\_receptionist* néven. Az EK-diagram alapján hoztam létre az adatbázis szerkezeti elemeit is: táblákat, azok mezőit, táblák közötti kapcsolatokat, idegenkulcs megkötéseket, tesztrekordok beszúrását.

Az adatbázis táblák kivétel nélkül magyar nyelvű karakterkódolást, InnoDB tárolómotort tartalmaznak, valamint 3NF-ig normalizált táblákat tartalmaz, a relációs adatbázisra leginkább jellemző egy-több (1:N) kapcsolatokkal, minél kevesebb redundancia megvalósítása idegenkulcs megkötésekkel az anomáliák és inkonzisztens adatok elkerülése céljából.

Az alábbi táblák alkotják az alkalmazások alapját:

* accomodation (regisztrált szálláshely adatai; cégnév, adószám stb.)
* accomodation\_profile (szálláshely felhasználói fiókjának szálláshelyazonosító-jelszó párosa)
* country (az alkalmazáshoz kapott tábla, amely Föld országainak neveit tartalmazza)
* guest (vendégek adatai; név, lakcím stb.)
* booking (foglalás adatai; érkezés dátuma, távozás dátuma, vendégszám stb.)
* room (szobák adatai; szobaszám, kapacitás stb.)
* billing\_item (szálláshely által beállított saját számlázási tételek; reggeli, idegenforgalmi adó stb.)
* billing\_item\_category (számlázási tételek kategóriáinak adatai, amely az alkalmazáshoz kapott tábla; szállás, ÁFA értéke, mennyiségi egysége stb.)

Ahogyan a *Melléklet, 4. ábrá*ja is mutatja, a legtöbb tábla kapcsolatban áll egymással. Az accomodation és az accomodation\_profile táblák között egy-egy (1:1) kapcsolat áll fent, adott felhasználói fiók szálláshelyazonosító-jelszó párosát csakis egyetlen egy szálláshelyhez köthetjük (egy szálláshelynek egy regisztrációja lehet elven).

A *country* tábla egy alkalmazáshoz kapott tábla (nincs olyan funkció, amely a tábla adatait manipulálni tudná), amelyben országnevek tárolódnak, ezt alkalmazza az asztali alkalmazás vendég felvitel esetén országnév kiválasztással, vagy számlázásnál a számlázási ország űrlapelem kiválasztásával.

A *guest* tábla tárolja a vendégek személyes adatait, idegenkulcsként tárolódik benne az ország tábla egy rekordjának kulcsa, amely a vendég országát hivatott tárolni.

A *room* táblában a szálláshely által beállított szobák tárolódnak, szobanév, szobaszám és maximális férőhely adatok tárolásra ad lehetőséget, valamint egy kapcsolómező, amely a szoba számláhatósági kategóriáját adhatja meg a billing\_item táblából, amely a szoba adott árát tartalmazza.

A *billing\_item* tábla szintén kapcsolatban áll egy másik táblával, a *billing\_item\_category*-val, amely már egy adott kategória ÁFA tartalmát adj meg vagy mennyiségi egységének elnevezését a számlázás folyamatában (nem módosítható tábla).

A *booking* tábla az a tábla, amely a több-több (M:N) kapcsolatot relációsémában leképző kapcsolótábla szerepét tölti be. Ez a tábla idegenkulcs megkötést tartalmaz vendég és szoba azonosítóra, hogy hatékonyan tudjuk eltárolni mikor melyik vendég érkezik, hány fővel melyik szobába érkezési és távozási dátummal kiegészülve.

Ezenkívül ahol szükséges volt, egyedi indexek vannak egyes mezőkre állítva (pl. guest táblában telefonszám vagy e-mail), amely meggátolja, hogy ugyanazon adat többször megismétlődhessen több rekordban (egy vendégnek egyedi e-mail címe van, más nem birtokolhatja más e-mail címét).

Saját konvencióm szerint minden egyes tábla ID automatikus növekvő, egyedi értékű kulcsmezővel rendelkezik, hogy minden egyed előfordulást maradéktalanul be tudjunk azonosítani.

## **1.2 WEBALKALMZÁS**

A webalkalmazás a Virtual Receptionist konfigurációs felülete.

Bootstrap 4 reszponzív felülettel rendelkezik és ő is közvetlenül csatlakozik az adatbázishoz (ez teszi lehetővé asztali és webalkalmazás között a kommunikációt).

PHP nyelven íródott, nincs architektúrába rendezve és natív PHP nyelven íródott keretrendszer nélkül, mivel kicsi az alkalmazás és alapvető funkciókat lát el (PHP nyelv erre tökéletes), és programtervezési minták sincsenek benne alkalmazva.

Számlázási tételek és szobamenedzsment feladatokat lát el, amellyel az asztali alkalmazás képes adminisztrálni.

alkalmazott library-k, API-k:

AJAX, Bootstrap 4 CSS és JS, jQuery

NINCS FRAMEWORK, ARCHITEKTÚRA,   
  
ADATBÁZIS CSATLAKOZÁS WEBEN

connect.php kapcsolatleíró hozza létre, globálisan felhasználható a projekt bármely állományában.

## **1.3 ASZTALI ALKALMAZÁS**

Az asztali alkalmazás Visual Studio 2017 Community Edition IDE[[9]](#footnote-9)-ben került kifejlesztésre, *WinForms* beépített grafikus osztálykönyvtár felhasználásával .NET 4.6.1 keretrendszer verziójában, C# programozási nyelven.

Az asztali alkalmazás a Virtual Receptionist adminisztrációs felületét adja. A korábban, webalkalmazásban is használható szálláshelyazonosító-jelszó párossal lehet belépni az alkalmazásba, amely alapján adatbázisból beazonosítja az adott szálláshelyet a rendszerben.

A korábban, webalkalmazásban bekonfigurált adatokkal tud az alkalmazás dolgozni, amely korábban is említett három fő modulból épül fel:

* Foglalási napló
* Vendégadatbázis
* Számlázó

A foglalási napló modulban van lehetőség foglalások rögzítésére, módosítására vagy törlésére. Minden foglalás felvitel esetén rögzítésre kerül az adatbázis *booking* kapcsolótáblájában a vendég minden adata, így azok későbbiekben újra felhasználhatóak, megspórolva az újbóli vendégadat űrlapmezők kitöltését.

A vendégadatbázis modulban a meglévő vendégek adatait tarthatjuk karban egy adattábla segítségével, amelyen szintén végezhetők adatmanipulációs műveletek a *guest* adatbázistáblájában.

A számlázó modulban a korábban felvitt, új foglalásokat számlázhatjuk ki. Minden felvitt foglalás alapértelmezetten nem számlázottként kerül be az adatbázisba, amelyet egy logikai típusú „Paid” mező tárol, 0 értékkel inicializálva. A számlázóban a webalkalmazásban beállított számlázási tételek közül válogathatunk és vihetjük fel egy adott foglalásra, amelyek megjelennek, mint nem számlázott foglalás. A számlanyomtatás funkcióval a felvitt rekordokból kész számlát kapunk, miközben a „Paid” mező az adatbázis adott foglalás rekordjában 1-es értékre változik, így később nem jelenik meg a számlázóban az adott foglalás nem számlázottként.

Ezen robosztus alkalmazás komplexitása rendszertervezés szintjén már nem engedhette meg azt a luxust, hogy nem alkalmazunk benne többrétegű architektúrát, ne alkalmazzunk programtervezési mintákat. Itt szükségessé válik a későbbi nagyobb továbbfejlesztések lehetőségének megteremtése céljából izolálni a grafikus felhasználói felületet, az alkalmazás interakciós logikáját, üzleti logikáját és perzisztens programszintű adattárolási metódusait. Ezekből kifolyólag az asztali alkalmazást többrétegű architektúrában terveztem és valósítottam meg. Ennek alapját a szoftverfejlesztési trendekben közismert *MVC*[[10]](#footnote-10) architektúra ihlette. Maga az alkalmazás nem tartja az MVC egyedi szabályait, inkább hagyományos többrétegű architektúrában készült, alkotói szabadságot adva a rétegek saját működésbeli definíciójának meghatározását, azonban felhasznál az MVC komponenseinek elnevezéséből. A továbbiakban rétegről rétegre fejtem ki az alkalmazás működését és funkcióit.

### **1.3.1. VIEW (NÉZET)**

Az alkalmazás legalsó rétege a nézet, azaz a grafikus felhasználói felület (GUI). A felhasználó ezekkel tud kapcsolatot teremteni.

Fontos szerepet játszott a megtervezésük során az ergonómia, mivel a felhasználó ezen keresztül látja az alkalmazás működését, ez teremt vele kapcsolatot. Az alkalmazás minden ablaka kerüli a kirívó színeket, egységes kinézetet biztosít, jól látható pontokon helyezkednek el a gombok, szövegmezők, táblázatok és egyéb GUI komponensek.

Maga a nézetek, úgynevezett *code-behind[[11]](#footnote-11)* részében maga a C# programozási nyelv színtisztán objektum-orientált mivolta okán egy osztály található, amely öröklődik a WinForms keretrendszer ős Form osztályából, amely egy ablakot hoz létre ezáltal. Ebből adódóan egy gyermekosztályként fogható fel minden ablak, azonkívül töredék osztályként, mivel *partial* módosító kulcsszóval van ellátva (ezzel lehetővé válik egy logikailag összetartozó osztály tárolása több, szuverén fájlban). Az ablak konstruktorában található egy InitializeComponent() elnevezésű metódus, ez az a metódus, amely az ablakra ráhúzott grafikus vezérlőket létrehozza és megjeleníti, és ez logikailag ugyanezen osztályban létező kód, azonban a jól olvashatóság miatt egy másik fájlban találhatók meg.

Mivel többrétegű alkalmazást fejlesztettem, így a code-behind fájlokban csak megjelenítési kódot írtam, de logikát nem; ilyen kód lehet egy törlés esemény hatására egy táblázat egy rekordjának eltüntetése. A logikai kód már a következő rétegben, a Controllerben, azaz a vezérlőben helyezkedik el. Valamely felhasználói interakció (pl. gombnyomás) azonnal abba a rétegbe kerül, hogy ott valamely logika, vagy adathalmaz kerüljön visszatérésre, hogy azt a nézet megjeleníthesse. Így a nézetek eseményeiben csak megjelenítési logika és megfelelő vezérlőmetódus hívás történhet.

### **1.3.2 CONTROLLER (VEZÉRLŐ)**

A Model és a View réteg ezáltal csak a Controlleren keresztül kommunikálhat (közvetetten), közvetlenül nem. Minden réteg a mellette lévő réteggel kommunikálhat csak közvetlenül.

virtuális objektum-adatbázist

### **1.3.3 REPOSITORY (ADATTÁR)**

MODEL

A legfelsőbb réteg a Repositories, amely perzisztens adattároló rétegként szolgál az alkalmazásban adatbázis és alkalmazás között, de több komponenst is magába foglal:

* Egy saját fejlesztésű adatbázis csatlakozást segítő osztályt, amely Singleton programtervezési mintát követve fejlesztettem ki, saját kivétellel (MySQLConnection/Database.cs, MySQLConnection/InvalidConnectionTypeException.cs)
* Az adatbázis és alkalmazás közötti perzisztens adattár osztályokat, amelyben a Repository osztály összeköti a legfelsőbb réteget az adatbázis csatlakozást segítő osztállyal, így eltárolhatók a Repositoryból öröklődő kisebb adattár osztályok adatszerkezeteiben az adatbázis adatai (SQL utasítások itt tárolódnak).
* Model osztályokat, amelyek az adatbázis egyedek (táblák) object-relational mapping objektumai (adattáblából konkrét forráskódszintű egyedeket, modelleket vázoltam fel).
* Tartalmaz egy DAO (Data-Access Object) programtervezési mintában megírt generikus interfészt (IGenericDAO<T>.cs), amelyben deklarálva vannak az alapvető adatbázis adatmanipulációs metódusok, úgymint törlés, módosítás, új felvitel. Az általánosságát az adja, hogy bármely típusra működik (T). Ezen interfésznek szüksége van egy osztályba, amelyben definiáljuk a metódusait, jelen esetben az adott, specializált öröklődött adattár osztály fogja az adatmanipulációs műveleteket implementálni, mint üzleti logika. A típus a megadott modell típusa lesz, pl. IGenericDAO<Guest>.

A Repositories lényegében a legfelsőbb réteg, üzleti logikát tartalmaz (pl. számlázóban kedvezmény számítás) és azon adatszerkezeteket, amelyek tárolják az adatbázis minden adatát, valamint az egyedek ORM-jeit.

CONTROLLER

Az alkalmazás köztes rétege a Controller, azaz az alkalmazás vezérlője. Ez a réteg a felhasználói felület (UI /user interface/ vagy View) és a Repositories réteg közötti irányításért felel. A bejövő, felhasználói adatokat a Controller validálja meghatározott validátor osztályok segítségével és küldi tovább, ha helyes. Amennyiben nem helyes egy input, azt a Controller saját, specializált kivételek formájában visszadobja a View-nak és hibaüzenetet ír ki a grafikus felületre. A Controller réteg lényegében megfelelő adattár metódusokat hív és ad vissza rá eredményt inputok alapján, validátor osztályokat és saját kivételek osztályait tartalmazza. A forma itt is a következő: az ős Controller osztály az ős Repository osztályt inicializálja, illetőleg tartalmazza azon ellenőrző (validátor) metódusokat, amelyeket az alkalmazás több modulja felhasznál, illetőleg olyan adatszerkezetekkel tér vissza, amelyeket több modul is felhasznál, illetőleg olyan formában adja vissza a lekért adatokat, hogy azt a View megtudja jeleníteni a felhasználónak, a többi felhasználói felülethez pedig örököltetett, specializált Controllerek inicializálják a nekik szükséges örököltetett adattárt vagy adattárakat.

ADATBÁZIS CSATLAKOZÁS ASZTALIN

MySQLConnector

Mind előbbiekben említettem, egy saját kifejlesztett adatbázis csatlakozást és adatmanipulációs műveleteket tartalmazó osztály végzi a Model réteg és az adatbázissal történő kommunikációt.

Ezen osztály olyan metódusokat tartalmaz, amelyek megnyitják majd lezárják az adatbázis csatlakozást (erőforrás-felszabadítás céljából), valamint a négy alapvető adatbázis CRUD műveletek (Create, Read, Update, Delete) MySQL kommunikáció szintjén megvalósító metódus, illetőleg nehezíti a helyzetet, hogy az osztály Singleton programtervezési mintával íródott.

Az első adatbázis kapcsolat megteremtése az autentikáció pillanatában indul el és ekkor állítja be a Database osztály az adatbázis elérésnek útvonalát, amely futás idejű konfigurálást tesz lehetővé.

Egy SetConnection metódus állítja be, hogy iskolai vagy otthoni adatbázis útvonalat akarunk-e használni, amely paraméteren kapja grafikus felületről a kapcsolódás típusát.

Otthoni és iskolai típus esetén egy beépített statikus osztály, a ConfigurationManager olvassa ki az előzőleg App.config XML állományba statikusan megadott adatbázis elérési útvonalat.

A ConfigurationManager egy metódusa az XML-ben megadott “local” és “remote” attribútumokból választhat, hogy a helyi vagy távoli kiszolgálóhoz csatlakozik. Ezt a SetConnection() metódus végzi feltétel vizsgálattal, attól függően, hogy ‘iskolai’ vagy ‘otthoni’ típust kap karakterláncként paraméterben.

Ha nem kap semmit, akkor képződik a saját kivételünk:

## **ALKALMAZOTT PROGRAMTERVEZÉSI MINTÁK**

Korábban már említettem, hogy asztali alkalmazásomban alkalmaztam úgynevezett “design pattern”-eket (programtervezési mintákat).

Kettő ilyen ismert pattern implementáltam:

* Singleton minta (egyke programtervezési minta),
* DAO minta (Data Access Object, magyarul: adathozzáférési objektum minta)

Az adatbázis kapcsolódást és adatbázisokban adat lekérdezést és adatmanipulációs metódusokat szolgáló osztályom, a *Database* és a szálláshely egyedet leíró modell osztály, az *Accomodation* osztályom ezen tervezési minta révén fejlesztettem ki.

A Singleton minta lényege, hogy az egész programfutás során egyetlen egy darab példány létezhet egy adott osztályból. Ennek megoldására a statikus változók jelentenek megoldást, amely a futás idején mindvégig élnek a memóriában és nem vesztik el értéküket. Az objektum-orientált nyelvekben a globális változók létezését próbálja megteremteni a fejlesztőnek egy statikus változó.

Ahhoz, hogy egy darab példány jöhessen létre egy osztályból, egy az osztály típusával megegyező típusú statikus változót kell a mezők között deklarálnunk, a konstruktort *private* láthatósági módosítóval kell ellátni (kívülről ne lehessen példányosítani végtelen számú objektumot) és a példányosított objektumot egy statikus getter metódus fogja nekünk visszaadni.

A getterben egy feltétel vizsgálat ellenőrzi, hogy a a statikus mező null értékű-e. Első alkalommal az, így a getter visszaadja a privát elérésű konstruktor hívás utáni példányosításával a példányt (belülről jön létre), amely a programfutás végéig élő, inicializált objektum lesz.

Amennyiben később is szükségünk lenne egy példányra az osztályból, mivel egy metódusára szükségünk lesz, úgy a továbbiakban is getter metódushoz kell forduljunk és visszatér a példányosodott objektummal. Ezekben az esetekben már inicializált objektumként él a memóriában a statikus változó, a feltétel vizsgálat kiértékelése során nem lép be az igaz ágra, hanem a már létező objektummal tér vissza.

Ugyanezen mintával építettem fel a szálláshely osztályt, amely autentikációs feladatokat végez (a szoftverben egy szálláshely dolgozik, így egy példány létezhet belőle elv szerint).

A másik programtervezési minta a DAO. Az adathozzáférési objektum lényege, hogy egy absztrakt interfészt biztosít adatbázis adat lekérdezés vagy adatmanipulációs mechanizmusokhoz. Általános elvárás, hogy minden modell osztályra legyen DAO interfész, hogy azt a modell osztályra örököltetve implementálható legyen.

Jelen projektben nem a modell osztályok metódusai lesznek a CRUD műveletek, hanem a specializált, örököltetett adattárakban kerülnek definiálásra, mint üzleti logika metódusok. Ahhoz, hogy ne kelljen minden adattárra megírni egy specializált DAO interfészt, így egy generikus DAO interfészt fejlesztettem.

Az interfész az adatlekérés metódusán kívül deklarálja az adatmanipulációs metódusokat, úgy mint törlés, módosítás, új felvitele. Paramétereik minden esetben várnak egy objektumot, amelyet elküldenek az adatbázis felé. Ha ezt a generikus interfészt örököltetem egy specializált adattára, megadhatom neki paraméterként milyen típusú objektumokra implementáljam a CRUD metódusokat, amelyek aztán feldolgozásra kerülnek adat manipuláció gyanánt a metódus blokkjában. A metódus blokkjában pedig ezáltal megírtam az SQL utasításokat, amelyet az adatbázis osztályom adat manipulációs metódusa eljuttat az adatbázisba.

# **FELHASZNÁLÓI DOKUMENTÁCIÓ**

## **TELEPÍTÉSI ÚTMUTATÓ**

Hhhjhzjzjbnfbkfnbkfnbkfb nfkbnfkbnfbk fn b kfbnfkbnfkbf nbkfnbkfnbk fbnfkbgf..

JFjfeofjefoefoejfefofjefoefjeofjfeof ejfeofjeoe jf eofjeejfo ejf oej of fejo jefoe fjo efjo efojo jefo jef joef eoj ojefoj efjoe foj efoje fj oef ejofjo feoj f jeof.

# **ÖSSZEGZÉS**

Afnkkfbnfkbnfbkfbfkb gfr rhkgi erighgi rhgrigh rir ghrighrghirighrigrhg ri ihgrighrig h gihrgi r gii rgr fkbnfbkfnbkfnbkfb nfkbnfkbnfbk fn b kfbnfkbnfkbf nbkfnbkfnbk fbnfkbgf..

JFjfeofjefoefoejfefofjefoefjeofjfeof ejfeofjeoe jf eofjeejfo ejf oej of fejo jefoe fjo efjo efojo jefo jef joef eoj ojefoj efjoe foj efoje fj oef ejofjo feoj f jeof.

TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

web:

* jelszó hash-elt tárolása adatbázisban
* regisztrációs űrlap (jelenleg előregisztrált minden)
* jelszócsere lehetőség szálláshelyek számára
* DatTables intelligens tábla framework
* MVC, PHP framework (SlimPHP), OOP PHP

asztali:

* hashelt jelszó dekódolása autentikáció esetén
* saját, helyi adatok SQLite adatbázisban történő tárolása a program részeként (ne központi szerveren)
* vendég azonosítóval történő foglalás módosítás azonos nevű vendégek esetén inkonzisztencia elkerülése (azonosító)
* Számla generálás PDF-ben, vagy nyomtató portra vezetett fizikai nyomtatás
* Vendégadatimportálás a foglalási naplóban.

ADATBÁZIS:

* relációséma átalakítása, úgy, hogy redundancia nélkül lehessen céges vendég adatokat eltárolni személyes vendégadatok mellett. Ne csak a számlázónál lehessen cégnevet megadni, hanem automatizált legyen

# **MELLÉKLET**

A záródolgozat mellékletét képzik ezenfelül egy CD lemez az alábbi tartalommal:

1. A záródolgozat szövege egy-egy példányban .docx és .pdf kiterjesztésű állományként
2. A gyakorlati munkák forráskódjai
3. A gyakorlati munkák futtatható állapotú állományai

1. Unified Modelling Language: általános célú modellező nyelv [↑](#footnote-ref-1)
2. Egyed-kapcsolat diagram: adatbázisok logikai megtervezését segítő grafikus leíró eszköz [↑](#footnote-ref-2)
3. Relational Database Management System: relációs adatbázis-kezelő rendszer [↑](#footnote-ref-3)
4. World Wide Web Consortsium: világhálóra történő szabad szoftver ajánlásokat készítő szervezet [↑](#footnote-ref-4)
5. Platformfüggetlenség: bármely operációs rendszerrel rendelkező számítógépen, ugyanolyan formában futtatható alkalmazás [↑](#footnote-ref-5)
6. Natív alkalmazás: egy platform saját nyelvében íródott, és képes annak a platformnak minden lehetőségét kihasználni [↑](#footnote-ref-6)
7. Structured Query Language: strukturált lekérdezőnyelv, a relációs adatbázisok által használt 4. generációs nyelv [↑](#footnote-ref-7)
8. Localhost: hálózati kártya visszacsatolási címe, helyi hálózaton történő alkalmazásfejlesztés hálózati csatlakozás meglétének tesztelése [↑](#footnote-ref-8)
9. Integrated Developement Enviroment: integrált fejlesztőkörnyezet, valamely alkalmazásfejlesztés során alkalmazott felület a kódolás folyamatának megteremtése érdekében [↑](#footnote-ref-9)
10. Model-View-Controller: Modell-nézet-vezérlő architektúra, leginkább webalkalmazások esetében, ahol jól elkülönített rétegben kerül implementációra a grafikus felhasználói felület, az üzleti logika, interakciós logika, adattárolás programszinten [↑](#footnote-ref-10)
11. Háttérkód: valamely grafikus ablak mögött elhelyezkedő fájl, amely GUI komponens inicializáló kódot tartalmaz [↑](#footnote-ref-11)