# **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 nem volt éppenséggel multinacionális vállalat, de annyira kisvállalkozás sem, hogy ne szorult volna rá egy megfelelő, kisebb vállalatirányítási információs rendszerre, hogy a napi adminisztráció alkalmával járó adatok (pl. vendégadatok kezelése) és információk (pl. foglalások felvitele adott időpontra) feldolgozása ne jelentsen terhet és minél gördülékenyebben történjen.

ndéglátóiegy szálláshely megrendelésére megtervezett és kifejlesztett bkfbfkb gfr rhkgi erighgi rhgrigh rir ghrighrghirighrigrhg ri ihgrighrig h gihrgi r gii rgr g rgrgnfb kfnbkfnb kf bnfkbfkbfk b nfkbnfkbnf kbn fbkfnbkfbkfnbkgr r fkbnfkbn fbkfgrg rg rnb grg r grg rkfbnfkbnfkbn 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.

# **FELHASZNÁLÓI DOKUMENTÁCIÓ**

## **TELEPÍTÉSI ÚTMUTATÓ**

kfnkkfbnfkbnfbkfbfkb gfr rhkgi erighgi rhgrigh rir ghrighrghirighrigrhg ri ihgrighrig h gihrgi r gii rgr g rgrgnfb kfnbkfnb kf bnfkbfkbfk b nfkbnfkbnf kbn fbkfnbkfbkfnbkgr r fkbnfkbn fbkfgrg rg rnb grg r grg rkfbnfkbnfkbn 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.

# **FEJLESZTŐI DOKUMENTÁCIÓ**

## **AZ ADATBÁZIS TERVEZÉSE**

kfnkkfbnfkbnfbkfbfkb gfr rhkgi erighgi rhgrigh rir ghrighrghirighrigrhg ri ihgrighrig h gihrgi r gii rgr g rgrgnfb kfnbkfnb kf bnfkbfkbfk b nfkbnfkbnf kbn fbkfnbkfbkfnbkgr r fkbnfkbn fbkfgrg rg rnb grg r grg rkfbnfkbnfkbn 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.

# **ÖSSZEGZÉS**

kfnkkfbnfkbnfbkfbfkb gfr rhkgi erighgi rhgrigh rir ghrighrghirighrigrhg ri ihgrighrig h gihrgi r gii rgr g rgrgnfb kfnbkfnb kf bnfkbfkbfk b nfkbnfkbnf kbn fbkfnbkfbkfnbkgr r fkbnfkbn fbkfgrg rg rnb grg r grg rkfbnfkbnfkbn 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.

ADATBÁZIS

Az alkalmazás magját az adatbázis adja. Az adatbázison keresztül tud az asztali és a webalkalmazás kommunikálni egymással közvetetten, mindkét alkalmazásnak szállít szükséges adatot, bizonyos adatait a web- és az asztali manipulálhatja is, de magát az adatbázis sémát egyik sem!

Az adatbázis XAMPP alkalmazás segítségével készült el localhoston tesztkörnyezetben, amely magában foglalja a MySQL RDBMS-et és phpMyAdmin webes adatbázis adminisztrátor alkalmazás segítségével. Az adatbázis az Oracle által ingyenesen felhasználható MySQL szabvány szerinti InnoDB tárolómotorral meghajtott adatbázis.

Az adatbázis normalizált táblákat (3NF-ig) tartalmaz, leginkább 1:N kapcsolattal, minél kevesebb redundancia megvalósítása idegenkulcs megkötésekkel az anomáliák és inkonzisztencia elkerülése végett.

EK-diagram

WEBALKALMZÁS

fejlesztői:

A webalkalmazás a rendszer 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

ADATBÁZIS CSATLAKOZÁS WEBEN

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

ASZTALI ALKALMAZÁS

Az asztali alkalmazás a rendszer adminisztrációs felületét adja. A korábban webes alkalmazásban bekonfigurált adatokkal dolgozik, adatbázison keresztül történik meg a kommunikáció a két alkalmazás között.

Az asztali, adminisztrációs felület többrétegű architektúrában került megtervezésre és kifejlesztésre. Nem a hagyományos MVC modellt követi, mivel nem tartja az MVC egyedi szabályait, hanem hagyományos többrétegű architektúra, azonban MVC komponensekkel:

* Model
* View
* Controller

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.

VIEW

Az alkalmazás legalsóbb rétege a View réteg, azaz a grafikus felhasználói felület. A felhasználó ezekkel tud kommunikálni. Ezen réteg a GUI ablakokat tartalmazza. Valamely esemény hatására (pl. gombnyomás) egy megfelelő vezérlőmetódus hívás történik, amely adatokkal tér vissza a View számára megjelenítésre.

Mivel egyes események hatására (pl. törlés esetén egy rekord eltüntetése az ablakról) nem igényelhet ismételt adatbáziskapcsolatot, így azt közvetlenül a View-ban írt megjelenítési logikával történhet, így a View is tartalmaz némi logikát, de csakis megjelenítési logikát - a Nézet ebben az architektúrában nem minden esetben passzív, hanem aktív résztvevője az eseményeknek.

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

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.

TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

web:

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

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