Asztali alkalmazás

Az alkalmazás egy .NET framework alapú Windows Forms applikáció.

Adatbázis/API kommunikáció

Az asztali alkalmazást az alkalmazottak használják az adatbázisban szereplő adatok kezelésére, így egy egyszerű gyors és biztonságos módot kell használni az adatbázissal való kommunikációra. Erre a feladatra az API-t használunk, ami az összes lekérdezést, hozzáadást, módosítást és törlést elvégzi, ami az asztali alkalmazásnak kell. Az API-val való kommunikációért az „ApiComm.cs” osztály végzi aminek csak egyetlen statikus „SendPost” metódusa van.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net.Http;

using System.Windows.Forms;

using System.ComponentModel;

using System.Web.Script.Serialization;

namespace Desktop

{

internal class ApiComm

{

public static async Task<object> SendPost(Dictionary<string, string> args)

{

var client = new HttpClient();

//var content = new FormUrlEncodedContent(args);

string parameters = "";

foreach(KeyValuePair<string, string> line in args)

{

parameters += $"{line.Key}={line.Value}&";

}

parameters.Remove(parameters.Length - 1);

var content = new StringContent(parameters, Encoding.UTF8, "application/x-www-form-urlencoded");

var response = await client.PostAsync("http://localhost:8000/BACKEND/api.php", content);

var responseString = await response.Content.ReadAsStringAsync();

JavaScriptSerializer serializer = new JavaScriptSerializer();

var responseDir = serializer.Deserialize(responseString, typeof(List<Dictionary<string, string>>));

client.Dispose();

return responseDir;

}

}

}

Ez a metódus egy „Dictionary<string, string>” típusú szótárat kér argumentumként, ami a POST kérés változóit és azok értékeit tartalmazza. A kérés megformálásáért NEM a FormUrlEncodedContent metódus felelős, mert kép feltöltése esetén túl nagyok voltak a paraméterek. Ennek orvoslása végett a POST request-et a metódus rakja össze magának, így csak a XAMPP beállításai korlátozzák egy POST kérés méreteit. A kérés formátuma és visszatérési értékei a fejlesztői dokumentáció adatbázis részénél található. Választ JSON formátumban kap, amit dekódol és visszaküldi „List<Dictionary<string,string>> ” típusú formátumban.

Kinézeti elemek

Az alkalmazásban található összes bemeneti mező saját fejlesztés nem az alapértelmezett verzió. Elsősorban a kinézet testreszabhatósága végett készültek ezek az elemek.

Bejelentkezés

A program indításakor a „loginForm.cs” lesz meghívva a felhasználó bejelentkeztetése mellet ez tárolja a bejelentkezett alkalmazott adatait, illetve az alkalmazás MainFrom-jának a referenciáját is itt lehet találni.

Főoldal

Bejelentkezést követően a „MainForm.cs” mindig látható lesz, itt található a menü és a többi form is ezen belül fog megnyílni az „OpenChildForm” metódus segítségével.

A felhasználói élmény növelésének érdekében a leggyakrabban használt kilistázó form-okat nem bezárjuk, hanem csak eltüntetjük, hogy a keresési szűrő megmaradjon és így kényelmesebb legyen a használat. Erről az oldalról indítjuk automatikus funkcióikat is, például a lejárt foglalások küldését vagy az új foglalások keresését. Ezek a törlések minden indításkor futnak le és csak egy egyszerű api hívás. Ilyenkor indul el az automatikus „CheckForBookings” metódus is, ami a „bookings” táblában levő változásokat keresi 15 percenként.

public void OpenChildForm(Form childForm)

{

if (activeForm != null)

{

if (activeForm == usersPage || activeForm == allBooksPage)

{

activeForm.Hide();

}

else

{

activeForm.Close();

}

}

prev = activeForm;

roundedCornerPanel\_PageContainer.Controls.Remove(activeForm);

activeForm = childForm;

childForm.TopLevel = false;

childForm.FormBorderStyle = FormBorderStyle.None;

childForm.Dock = DockStyle.Fill;

roundedCornerPanel\_PageContainer.Controls.Add(childForm);

roundedCornerPanel\_PageContainer.Tag = childForm;

ThemeManager.ApplyThemeToAllForms();

childForm.BringToFront();

childForm.Show();

childForm.Visible = true;

}

public async Task CheckForBookings(bool now)

{

if (!now)

{

while (true)

{

List<Dictionary<string, string>> resp = (List<Dictionary<string, string>>)await ApiComm.SendPost(new Dictionary<string, string> { { "type", "checkForBookings" } });

if (resp.First()["result"] == "true")

{

pbNewBookings.Visible = true;

}

await Task.Delay(TimeSpan.FromMinutes(15));

}

}

else

{

List<Dictionary<string, string>> resp = (List<Dictionary<string, string>>)await ApiComm.SendPost(new Dictionary<string, string> { { "type", "checkForBookings" } });

if (resp.First()["result"] == "true")

{

pbNewBookings.Visible = true;

}

}

}

Két módja van, ha a „now” paraméter hamis, akkor fogja a fix intervallumonkénti keresést indítani. Ha igaz a paraméter akkor a jelen pillanatban fogja megnézni, hogy történt-e változás.

Részletes formok

private async void Form1\_Load(object sender, EventArgs e)

{

OpenChildForm(new HomePage());

lblEmplUname.Text = LoginForm.empl\_uname;

List<Dictionary<string, string>> temp = (List<Dictionary<string, string>>)await ApiComm.SendPost(new Dictionary<string, string> { { "type", "getSystemSettings" } });

system\_settings = temp.First();

CheckForBookings(false);

List<Dictionary<string, string>> reservationResult = (List<Dictionary<string, string>>)await ApiComm.SendPost(new Dictionary<string, string>() { { "type", "deleteExpiredReservations" } });

List<Dictionary<string, string>> bookingResult = (List<Dictionary<string, string>>)await ApiComm.SendPost(new Dictionary<string, string>() { { "type", "deleteExpiredBookings" } });

}

Két részletes oldal van „bookDetailedPage.cs” és „userDetailedPage” itt. Ezeken az oldalakon az adott tábla összes adatát módosítani lehet, illetve extra műveleteket is lehet az oldalakon csinálni (pl.: egy felhasználó részletes oldalán a tagságát lehet meghosszabítani). A form érzékeli, ha változás történt az adatokban úgy, hogy eltárolja az eredeti betöltött adatokat, és ha különbözik a jelenlegi adatoktól akkor történt változás. Figyelmeztet is ha be szeretnénk zárni az adott részletes oldalt. A „bookDetailedPage.cs” is egyedi, mivel pontosan úgy, mint a „PopupSelect.cs” két különböző módban indítható el. Az elsőben, ahogy már említve volt módosítani lehet egy könyvet, a második módjában könyvtípust lehet vele hozzáadni.

Hozzáadós formok

Két hozzáadós form fajta van, mindegyiknek egyetlen lényege, hogy egy új példányt adjanak hozzá a megfelelő táblához. A két fajta között a legfőbb különbség a kinézet. Az első fajtába tartozik az „addEmployee.cs”, „addPublisher.cs” és az „addUser.cs”. Egyszerű hozzáadási formok bemeneti mezőkkel. A másik fajta pedig a „bookLendingPage.cs”, amivel előjegyezni és kivenni lehet könyveket. Két gomb két popup select-et nyit meg amikkel ki kell választani a könyvet és az olvasót. Itt az dönti el, hogy előjegyzést vagy kivételt add hozzá, hogy melyik form hívja meg és milyen paraméterrel, szóval itt is van a konstruktorban egy paraméter, hogy mit fog csinálni. Ehhez a formhoz tartozik még egy extra funkcióként azt is megjeleníti, hogy mikor olvasta legutóbb az adott könyvet az olvasó.

Visszavevő form

Amikor visszavesznek egy könyvet, a „bookTakebackPage.cs” lesz meghívva, ami összesíti a késedelmet és egy végső jóváhagyást kér visszavétel előtt. Amikor a felhasználó oldaláról vesznek vissza könyvet ez nem fog megjelenni, mert ott egyszerre több könyvet is vissza lehet venni.

Alkalmazottak

Az „EmployeeDetailed.cs” az jelenleg bejelentkezett alkalmazott adatait tölti be. A jelszóváltoztatás itt nem olyan biztonságos, mint a felhasználók jelszóváltoztatása, itt csak összehasonlíja a régi jelszót az újjal, illetve az új ismétlésével és ki is cseréli. Itt is van a konstruktorban egy paraméter, ami meghatározza milyen módban indul el. Itt, ha a felhasználó admin jogosultsággal rendelkezik a rendszer adatait és az alkalmazottakat is tudja felügyelni módosítani.

Kilistázó formok

Ebbe a kategóriába tartozik a „AllBooksPage.cs”, „BookingsPage.cs”, „BorrowingsPage.cs”, „PublishersPage.cs”, „ReservationsPage.cs”, „UsersPage.cs” és gyakorlatilag a „PopupSelect.cs” is de arról egy másik pontban részletesebben is lesz írva. Az összes ilyen form gyakorlatilag az adatbázis valamelyik tábláját jeleníti meg egy datagridview-ben és onnan lehet további műveleteket végezni specifikus sorokkal, hozzáadni újat, meglévőt módosítani, illetve meglévőt törölni. Mindegyik hasonló frissítő metódust használ, de sosem ugyanazt.

private async void updateBooksDgv(string search)

{

cdgvBooks.DataSource = null;

cdgvBooks.Columns.Clear();

ids.Clear();

List<Dictionary<string, string>> response = (List<Dictionary<string, string>>)await ApiComm.SendPost(new Dictionary<string, string> { { "type", "getBooks" }, { "search", search } });

if (response.Count>0)

{

DataTable dt = new DataTable();

DataColumn col;

DataRow row;

//make the dataset columns

foreach (KeyValuePair<string, string> item in response[0])

{

if (item.Key != "book\_id" && item.Key != "available")

{

col = new DataColumn();

col.DataType = typeof(string);

col.ReadOnly = true;

col.ColumnName = item.Key;

col.Caption = item.Key;

dt.Columns.Add(col);

}

}

//add button row for detailed page

DataGridViewButtonColumn btns = new DataGridViewButtonColumn();

btns.Name = "Részletesen";

btns.Text = "Részletes";

btns.UseColumnTextForButtonValue = true;

//add button column for booking/available/borrowed

DataGridViewButtonColumn btncol = new DataGridViewButtonColumn();

btncol.Name = "Műveletek";

//make rows

for (int i = 0; i < response.Count(); i++)

{

row = dt.NewRow();

foreach (KeyValuePair<string, string> item in response[i])

{

if (item.Key == "book\_id")

{

ids.Add(item.Value);

}

else if(item.Key != "available")

{

row[item.Key] = item.Value;

}

}

dt.Rows.Add(row);

}

cdgvBooks.DataSource = dt;

cdgvBooks.Columns.Add(btns);

cdgvBooks.Columns.Add(btncol);

foreach (DataGridViewColumn column in cdgvBooks.Columns)

{

column.SortMode = DataGridViewColumnSortMode.NotSortable;

}

addBtnTextsManual();

}

Feltölti a datagridview-t a kapott adatokkal, a sorok azonosítóját nem jeleníti meg, de eltárolja egy listában. Majd formtól függően különböző gomb sorokat ad hozzá.

Popup select

Ez a form azért speciális, mert nem csak egy funkciót lát el, hanem többet egyszerre. Mindig ez jelenik meg ha a felhasználónak választania kell valamit (pl.: kivételkor melyik könyvet szeretné kivenni az olvasó). Ezért először indítási módtól függően (ami az osztály konstruktorának paramétere) meghatározza, hogy mi fog megjelenni (pl.: nem minden esetben jelenik meg kereső mező), majd miután a felhasználó megtalálta a megfelelő elemet és kiválasztotta, a form visszaküldi az elem fontos paramétereit, amit a meghívó form fel fog dolgozni. A felhasználói élmény javítása végett bizonyos esetekben egyszerre több opciót is lehet választani (pl.: amikor egy könyvnek több kategóriája van ne egyesével kelljen kiválasztani mindegyiket).