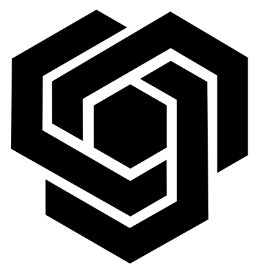
**Технически Университет София**

Факултет Компютърни системи и технологии

**Курсов Проект**

**По пму**

**Приложение за резервация на стаи в хотели**

Изготвили:

Йоанна Светлозарова Василева

Специалност: КСИ

Курс: Трети

Група: 42

Фак. № 121214108

Мила Иванова Иванова

Специалност: КСИ

Курс: Трети

Група: 42

Фак. № 121214080

София, Юни 2017г.

СЪДЪРЖАНИЕ

Увод.......................................................................................................................................... 3

Анализ на съществуващи разработки................................................................................... 3

Проектиране............................................................................................................................ 4

Реализация............................................................................................................................... 6

Потребителски ръководство.................................................................................................. 11

Заключение............................................................................................................................. 15

Литература.............................................................................................................................. 15

Приложение............................................................................................................................ 16

УВОД

В технологичното време, в което живеем, всяко едно действие ни е неимоверно свързано с мобилните технологии. От това да си поръчаме храна за вкъщи чак до това да запазваме стаи в любимия ни хотел за дългоочакваната почивка.

Възможността за използване на геолокационни услуги и други вградени сензори, обособяват мобилните устройства като една много благоприятна платформа за широк спектър от услуги. Това гарантира, че дори да не знаем къде сме, използвайки приложението, ще намерим хотел на добри цени и условия.

АНАЛИЗ НА СЪЩЕСТВУВАЩИ РАЗРАБОТКИ

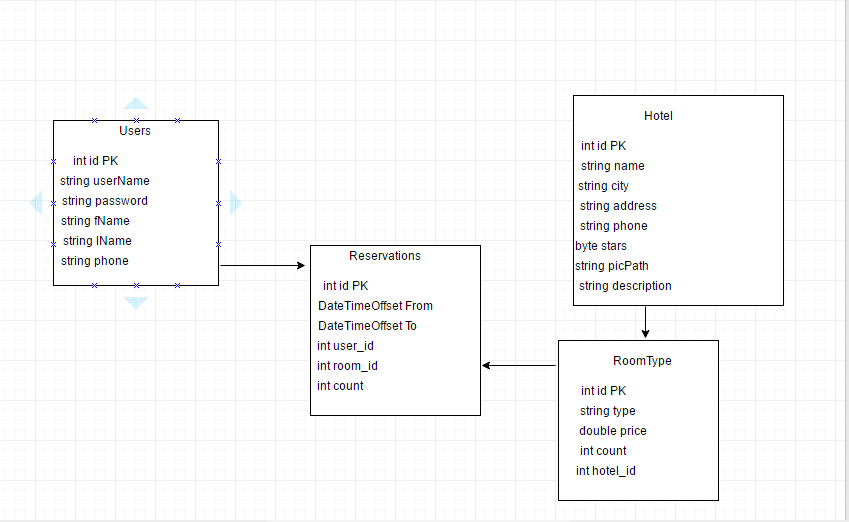
Може да прозвучи амбициозно от наша страна, но за анализ на съществуваща разработка ще използваме едни от най-популярните сайтове за резервация на хотели у нас, а именно trivago.bg, booking.com, hotels.com.

Като предимство на нашия продукт пред посочените платформи по-горе е лесната достъпност до опцията „геолокация“. Също така смятаме, че интерфейсът е по-user friendly от доста популярни сайтове за резервация на хотели. А липсата на реклами допринася за това, приложението да има изчистен дизайн.

Като недостатък на нашия продукт може да отбележим, че при разработката му не взехме под предвид някои от особеностите на приложението и погрешно подходихме към избиране на технологията за създаване на базата. Това от своя страна ще доведе до доста проблеми на разработчиците при евентуално разширяване на приложението.

ПРОЕКТИРАНЕ

Базата данни има следния изглед:



Продуктът ще се използва от туристи, които искат да направят резервация в хотел.

При стартиране на приложението, потребителят достъпва страница за вписване. На страницата има бутон за регистриране в системата. При натискане на бутона, приложението се насочва към страница за регистрация. На страницата за регистрация има полета за въвеждане на информация за потребителя – потребителско име, парола, собствено име, фамилия и телефон. Освен това има бутон за регистрация. Ако валидацията на данни е преминала успешно – новият потребител се добавя в базата данни. Има бутон „назад“, който връща към първоначалната страница.

Въвеждат се потребителско име и парола и при натискане на бутона за вписване, при валидиране на данните, се извършва навигация към страница за информация на потребителя. Там има информация за потребителско име и резервации на потребителя. При натискане на бутон „каталог“, потребителят достъпва страница, която представлява списък от хотели. Подробна информация за хотела може да се разгледа, като се селектира хотел от списъка и се натисне бутона, насочващ към детайли. Има възможност за връщане на предходната страница при натискане бутона „назад“.

В ново показалата се страница излиза името на хотела, звезди, снимка, град, адрес, телефон, описание, типове стаи и цени. Има три бутона със следните опции: резервирай, покажи на карта и обратно.

- При натискане на първия бутон, се отваря нова страница с форма за резервация. В нея е попълнено името на конкретния хотел. Потребителят има опцията да избере от списък типа стая, която иска да резервира, брой на стаите и дати на престоя. При натискане на бутона за резервация, ако валидацията на данни е преминала успешно се добавя резервацията към базата данни за конкретния човек. При натискане на бутона за назад, се изпълнява навигация към предходната страница.

- При натискане на бутона за покажи на карта се отваря нова страница, която показва местоположението на хотела на карта. При натискане на бутон „моята локация“, приложението заявява разрешаване на достъп до местоположението на устройството. При позволяване, програмата добавя на картата местоположението. При натискане на бутона за показване на трасето, свързващо хотела и моментната локация на потребителя – се извежда трасето. Има бутон за връщане на предходната страница.

- При натискане на бутона за връщане назад, програмата изпълнява навигация към предходната страница – каталога с хотели.

- При регистрация, потребителят въвежда потребителско име, парола, собствено име, фамилно име и телефон. Тези данни се записват като тип string. Отделно се прави валидация на всяко поле, чрез regex. Данните се записват в обект от клас User и се запазва в базата данни в таблица User.

- При вписване в системата, потребителят въвежда потребителско име и парола. Полетата са от тип string. Претърсва се базата данни и ако съществува съответното потребителско име със съответната парола-потребителят се вписва. Запазва се информацията за вписания потребител, като се подава потребителското име като параметър при навигация на страници.

- При регистрация, потребителят въвежда дати за престой, тип стая и брой стаи на избран от него хотел. Датите се съхраняват в DateTimeOffset. Типът на стаята се избира от списък стаи на конкретния хотел.

Извършва се проверка дали началната дата е по-голяма от текущата дата. Извършва се проверка дали крайната дата е по-голяма от началната дата. Ако тези две условия не за изпълнени, се връща съобщение за грешка в текстов блок.

Извършва се проверка и на броя стаи, въведени от потребителя. Проверява се в базата данни за избрания период, колко стаи от избрания тип има резервирани. Ако броя на стаите, избрани от потребителя, и по-голям от оставащите в хотела, се връща грешка в текстов блок за недостатъчен брой стаи. Проверява се и дали избраният брой е по-голям от 0. Ако проверката, върне резултат по-малък или равен на 0 - съобщение за грешка се изписва в текстов блок.

Ако входните данни преминат успешно през проверките – резервацията се извършва.

РЕАЛИЗАЦИЯ

Приложението използва бази данни sqlite. При стартиране на приложението се създава файлът на базата данни, ако не съществува. Това се случва, във файла app.xaml.cs на приложението.

if (!CheckFileExists("db.sqlite").Result)

{

using (var conn = new SQLiteConnection(new SQLite.Net.Platform.WinRT.SQLitePlatformWinRT(), DB\_PATH))

{

conn.CreateTable<Users>();

……

putData(conn);

}

}

Методът putData() попълва данни за хотелите и типовете стаи. Тъй като тези данни не се попълват от потребителя, а от администратора на приложението, те са директно попълнени при стартиране на програмата.

Приложението следва MVVM модела. Класовете, чрез които се създава базата данни са в папка Модел на приложението.

Бизнес логиката е имплементирана в различни ViewModels.

За работа с базата данни е създаден клас в папката ViewModel – DataBaseHelperClass. Той се грижи за обхождане на таблиците и вмъкване на данни.

Създаден е клас ViewModelBase, който наследява INotifyPropertyChanged . Този клас бива наследен от всички класове от ViemModels, свързани за View. Чрез него се извиква методът NotifyPropertyChanged(), който засича когато някое property е променено.

Първата страница е свързана с класът SignIn. Командите и текстовите полета са байнднати към property-та и ICommand. При натискане на бутона се извиква методът getData(), който извиква метода searchUser(), който от своя страна претърсва базата данни за наличието на конкретния потребител. Навигацията между страниците се осъществява в code-behind-a на View. Ако потребителят успее да се впише, се пази информацията му в текстово поле, по което е предадено като параметър потребителското име.

if (isSignedIn.Text != "No user with this username and password!")

{

Frame.Navigate(typeof(UserData), usrName.Text);

}

Страницата за регистрация се достъпва при натискане на бутона за регистрация.

Frame.Navigate(typeof(Registration));

Тя е свързана с класа ConnectToUsers.cs. Той съдържа пропъртита за всяко поле от изгледа – потребителско име, парола, име, фамилия и телефон. Прави се валидация на данни за всяко поле. За потребителското име се претърсва базата и ако вече съществува такова- се появява грешка. Останалите полета се валидират чрез различни Regex-и. Имената чрез (Regex.IsMatch(fName, "^[\\p{L} \\.\\-]+$"), телефоните чрез (Regex.IsMatch(phone, @"^\+?[0-9][0-9\s.-]{7,11}$")) и паролите чрез (Regex.IsMatch(password, "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d).{8,15}$")). Освен това ако някое от полетата не е попълнено, приложението прихваща изключение и отново връща съобщение за грешка. Това се повтаря докато потребителя не данни, отговарящи на условията или не натисне бутон „назад“.

Ако потребителят успее да се впише в системата, при натискане на бутона SignIn приложението го пренасочва към страница за информация на потребителя. Страницата получава като параметър потребителското име и го байндва към текстово поле. Във кореспондиращия ViewModel - UserDataVM прочитаме потребителското име, обхождаме таблицата Users и намиране id-то на съответния потребител. От там обхождаме таблицата Reservations и намираме какви резервации има потребителя. След това обхождаме таблиците RoomType и Hotel, за да намерим името на хотела, в който сме запазили стая. Информацията, съдържаща името на хотела, типа стая, дата на настаняване и дата на напускане се записват в променлива от тип String, като са разделени със запетая. Тази променлива се запазва в добавя към списък от тип ObservableCollection, към който е байнднато ListView

public void getInfo()

{

ObservableCollection<int> hotelsId = getHotelId();

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Hotel> list = Db\_helper.ReadAllHotels();

ListRes = new ObservableCollection<string>();

foreach (Hotel hotel in list)

{

foreach (int htl in hotelsId)

{

if (htl == hotel.id)

{

HotelName = hotel.name;

Info = HotelName + ", " + RoomTyp+ ", " + From+ ", " + To;

ListRes.Add(Info);

}

}

}

}

При натискане на бутона насочващ към Каталога се извиква метода Frame.Navigate, към който се подава името на следващата страница и потребителското име като параметър.

В страницата с каталог има списък на хотелите – техните имена. Освен това в текстово поле се пази името на вписания потребител. Списъка с хотелите се намира в ListView, което е байндато към ObservableCollection. В конструктора на VieModel-a се обхожда таблицата Hotels, взима името на хотела на всеки обект и се добавя към ObservableCollection<string>(); Избрания елемент от списъка се байндва към променлива от тип стринг. При натискане на бутона за показване на детайли се изпълнява навигация на страници в code behind-a и потребителското име и името на хотела се предават като параметри на следващата страница.

private void btnShowHotel\_Click(object sender, RoutedEventArgs e)

{

string[] parameters = new string[2];

parameters[0] = usNm.Text;

string selected = (string)listHotels.SelectedValue;

parameters[1] = selected;

if (selected != null)

{

Frame.Navigate(typeof(HotelOverview), parameters);

}

}

При натискане на бутона за показване на карта, приложението извиква нова страница с Frame.Navigate, на която и предаваме потребителското име, името на хотела, града, в който е хотела и адреса му. От тук в code-behind-a се работи с картата. Тя взима информация за адреса на хотела и слага точка на картата с местоположение.

protected async override void OnNavigatedTo(NavigationEventArgs e)

{

base.OnNavigatedTo(e);

string[] parameters = new string[2];

parameters = (string[])e.Parameter;

var vm = new MapViewModel();

vm.UserName = parameters[0];

vm.HotelName = parameters[1];

vm.getLocation();

this.DataContext = vm;

string address = getAddress();

var results = await MapLocationFinder.FindLocationsAsync(address, null);

double lat = 0.0;

double lon = 0.0;

if (results.Status == MapLocationFinderStatus.Success)

{

lat = results.Locations[0].Point.Position.Latitude;

lon = results.Locations[0].Point.Position.Longitude;

}

BasicGeoposition position = new BasicGeoposition()

{

Latitude = lat,

Longitude = lon

};

Geopoint gPoint = new Geopoint(position);

MapIcon mapIcon1 = new MapIcon();

mapIcon1.Location = gPoint;

mapIcon1.NormalizedAnchorPoint = new Point(0.5, 1.0);

mapIcon1.Title = "Your Hotel";

mapIcon1.ZIndex = 0;

// Add the MapIcon to the map.

mapControl.MapElements.Add(mapIcon1);

// Center the map over the POI.

mapControl.Center = gPoint;

mapControl.ZoomLevel = 14;

}

При натискане на бутона за моето местоположение, приложението приканва потребителя да позволи използването на местоположение. От там намира къде се намира потребителя и добавя точка с местоположението му на картата.

Geoposition pos = await geoLocator.GetGeopositionAsync();

При натискане на бутон за маршрут се взимат двете точки от картата и се изчертава маршрута между тях. Програмата показва разстоянието в километри и време между потребителя и хотела.

private async void Directions\_Click(object sender, RoutedEventArgs e)

{

Geopoint begin = null;

Geopoint end = null;

foreach (MapIcon item in mapControl.MapElements)

{

if (item.Title.Equals("You are here"))

{

begin = item.Location;

}

else if (item.Title.Equals("Your Hotel"))

{

end = item.Location;

}

}

try

{

MapRouteFinderResult routeResult = await MapRouteFinder.GetDrivingRouteAsync(begin, end);

if (routeResult.Status == MapRouteFinderStatus.Success)

{

MapRouteView viewOfRoute = new MapRouteView(routeResult.Route);

viewOfRoute.RouteColor = Colors.Yellow;

viewOfRoute.OutlineColor = Colors.Black;

mapControl.Routes.Add(viewOfRoute);

await mapControl.TrySetViewBoundsAsync(

routeResult.Route.BoundingBox,

null,

Windows.UI.Xaml.Controls.Maps.MapAnimationKind.None);

System.Text.StringBuilder routeInfoKm = new System.Text.StringBuilder();

System.Text.StringBuilder routeInfoTime = new System.Text.StringBuilder();

routeInfoTime.Append("Total estimated time = ");

routeInfoTime.Append(routeResult.Route.EstimatedDuration.TotalMinutes.ToString());

routeInfoTime.Append("min");

routeInfoKm.Append("\nTotal length = ");

routeInfoKm.Append((routeResult.Route.LengthInMeters / 1000).ToString());

routeInfoKm.Append("km");

infoKm.Text = routeInfoKm.ToString();

infoTime.Text = routeInfoTime.ToString();

}

}

catch (NullReferenceException ex)

{

err.Text = "Your location must be set!";

}

}

Ако в страницата за детайли за хотела се натисне бутона за резервацията, приложението пренасочва към страницата с резервационна форма. Към нея са предадени като параметър потребителското име и името на хотела. Потребителя може да избере какъв тип стая иска да запази, брой на стаите и от и до кой ден ще бъде престоя му. Прави се валидация на данните. Обхожда се таблицата за резервации и се намира колко свободни стаи има в зададения период. Ако избрания брой стаи превишава оставащия брой – съобщение за грешка се появява.

Валидират се и дните за престой. Правят се проверки дали началния ден е по-голям от днешния. Също така се проверява дали последния ден е по-голям от първия. Ако потребителят е въвел грешно дните за престой се появява грешка. Резервацията в възможна само ако данните са въведени правилно.

public Reservations reservation()

{

Reservations res = new Reservations();

int remaingngCnt = getRemainingRmCnt();

if (RmCount < remaingngCnt && RmCount > 0)

{

res.room\_id = getRmId();

res.user\_id = getUsrId();

if (From >= DateTime.Now && To > From)

{

res.dateFrom = From;

res.dateTo = To;

Error = null;

}

else

{

Error = datesPopulationError;

}

res.count = RmCount;

}

else if (RmCount > remaingngCnt)

{

Error = noRoomsAvailableError;

}

else if (RmCount <= 0)

{

Error = noRoomsSelectedError;

}

return res;

}

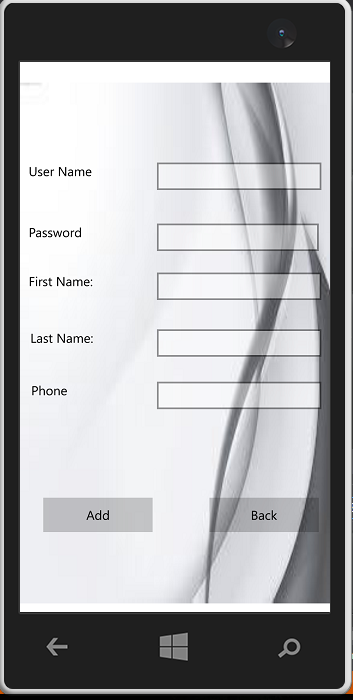
В повечето страници има бутон за назад, който извиква предната страница чрез Frame.Navigate, като предава нужните параметри.

ПОТРЕБИТЕЛСКО РЪКОВОДСТВО



Това е първоначалният изглед при стартиране на приложението. На това View има 2 текстови полета – съответно за въвеждане на потребителско име и на парола.

View-то също така разполага с 2 бутона. Бутонът “Sign in” служи за вписване в системата. Бутонът “Register” от своя страна води до друго View, което служи за регистрация на нов потребител в системата.



Това е View-то, което се визуализира след натискане на бутона “Register”. Разполага с 5 textbox-a съответно за:

username

password

first name

last name

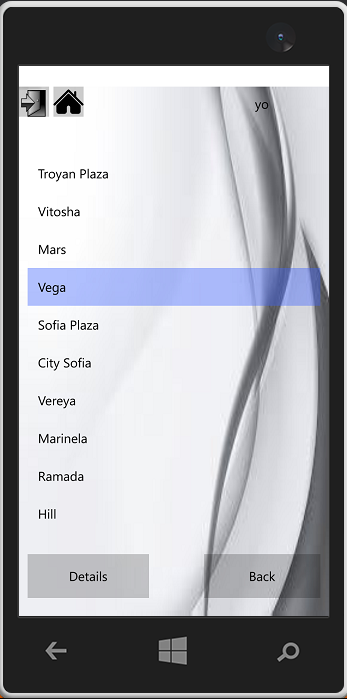
phone

Бутонът “Add” служи са добавяне на новия потребител със съответните данни. При положение, че данните са коректни, потребителят е регистриран успешно в системата, след което ни връща на първоначалното View.

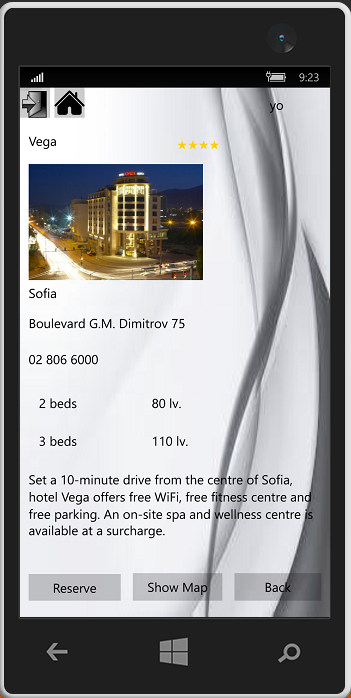


Това View се появява след натискане на бутона “Sign in” при положение, че сме въвели данни на потребител, който има регистрация в системата.

Тук се визуализират username на даден потребител, заедно с неговите резервации, ако са направени такива. Бутонът “Go to Catalog” те пренасочва към каталога с оферти на хотели.



Това View се появява след натискане на бутона “Go to Catalog”. Съдържа лист с наличните оферти на хотели и два бутона. Бутонът “Show details” показва информация за съответния селектиран хотел. Бутонът “Back” пренасочва потребителя към предходната страница.



При маркиране на (в конкретния случай „Troyan Plaza“) и натискане на бутона “Show details” се визуализира следния екран. На него е раположена инфирмацията за съответния хотел:

Име на хотел

Степен на комфорт

Местоположение

Телефон за връзка

Видове стаи и съответните им цени

Описание на хотела

Трите бутона в долната част на View-то служат съответно за:

“Reserve” – пренасочва към нова страница за резервация на стаи в хотела

“Show on map” – визуализира се нов прозорец, на който е разположена картата, на която е обозначен хотелът

“Back” – навигира към предишната страница.



При натискане на бутона “Reserve” се визуализира следното View. На него са разположени следните компоненти:

Име на потребителя

Фамилия на потребителя

Име на хотел

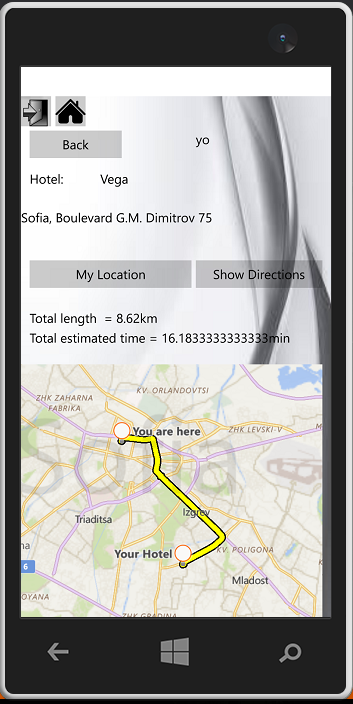
Тип стая (избираемо)

Чрез CalendarDatePicker се избира от коя дата до коя дата да се направи резервацията.

Въвеждаме колко стаи желаем да резервираме в текстово поле

В долната част на страницата има два бутона:

“Make reservation” – след натискането на този бутон се добавя резервацията.



След натискане на бутона “Show on map” се визуализира следното View. “My Location” бутонът служи за включване на локацията на устройството. След това ако натиснем бутона “Show Directions” на картата ни се визуализира маршрут от местоположението на устройството до избрания хотел.

ЗАКЛЮЧЕНИЕ

Приложението изпълнява целите, които са поставени при проектиране. Използван е лесен за употреба интерфейс. Основната функционалности за резервация на хотел е изпълнена, както и са добавени допълнителни функционалности за регистриране на потребители и потребителска информация. Използвани са функции на телефона – местоположение. Приложението може да работи без да се използва връзка с интернет.

Възможните подобрения биха били качване на базата данни към сървър.

ЛИТЕРАТУРА

1. https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide
2. https://stackoverflow.com
3. http://tasheva.info/exercises.php
4. https://msdn.microsoft.com/en-us/library/hh848246.aspx
5. https://www.tutorialspoint.com/sqlite/
6. https://www.sqlite.org/quickstart.html
7. Денис Колисниченко. Програмиране на мобилни приложения за Windows 10, 16-01-2017 г.
8. Майкъл Бъртън, Дон Фелкер. Разработване на приложения за Android For Dummies, 21-10-2015 г.

ПРИЛОЖЕНИЕ

<https://github.com/jvassileva/HotelReservation>

<https://github.com/milaivanova1995/C-Projects/tree/master/HotelReservation>

* Модел Hotel:

public class Hotel

{

[PrimaryKey]

public int id { get; set; }

public string name { get; set; }

public string city { get; set; }

public string address { get; set; }

public string phone { get; set; }

public byte stars { get; set; }

public string picPath { get; set; }

public string description { get; set; }

public Hotel() { }

}

* Модел RoomType:

public class RoomType {

[PrimaryKey]

public int id { get; set; }

public string type { get; set; }

public double price { get; set; }

public int count { get; set; }

public int hotel\_id { get; set; }

public RoomType() { }

}

* Модел Users:

public class Users

{

[PrimaryKey, AutoIncrement]

public int id { get; set; }

public string userName { get; set; }

public string password { get; set; }

public string fName { get; set; }

public string lName { get; set; }

public string phone { get; set; }

public Users() { }

public Users( string usnm, string pass, string firstN,string lastN, string ph)

{

userName = usnm;

password = pass;

fName = firstN;

lName = lastN;

phone = ph;

}

}

* Модел Reservations:

public class Reservations

{

[PrimaryKey,AutoIncrement]

public int id { get; set; }

public DateTimeOffset dateFrom { get; set; }

public DateTimeOffset dateTo { get; set; }

public int user\_id { get; set; }

public int room\_id { get; set; }

public int count { get; set; }

public Reservations() {

}

public Reservations(DateTimeOffset from, DateTimeOffset to, int userId, int roomId, int cnt)

{

dateFrom = from;

dateTo = to;

user\_id = userId;

room\_id = roomId;

count = cnt;

}

}

* View Model ViewModelBase:

public class ViewModelBase:INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public void NotifyPropertyChanged([CallerMemberName]String propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

}

* View Model DelegateCommand:

public class DelegateCommand:ICommand

{

private Action \_action;

private bool \_canExecute;

public DelegateCommand(Action action, bool canExecute)

{

\_action = action;

\_canExecute = canExecute;

}

public bool CanExecute(object parameter)

{

return \_canExecute;

}

public event EventHandler CanExecuteChanged;

public void Execute(object parameter)

{

\_action();

}

}

* View Model DataBaseHelperClass:

public class DatabaseHelperClass

{

public void InsertUser(Users usr)

{

using (SQLite.Net.SQLiteConnection conn = new SQLite.Net.SQLiteConnection(new SQLite.Net.Platform.WinRT.SQLitePlatformWinRT(), App.DB\_PATH))

{

conn.RunInTransaction(() =>

{

conn.Insert(usr);

});

}

}

public void InsertReservation(Reservations reservation)

{

using (SQLite.Net.SQLiteConnection conn = new SQLite.Net.SQLiteConnection(new SQLite.Net.Platform.WinRT.SQLitePlatformWinRT(), App.DB\_PATH))

{

conn.RunInTransaction(() =>

{

conn.Insert(reservation);

});

}

}

public Users ReadUsers(int user\_id)

{

using (SQLite.Net.SQLiteConnection conn = new SQLite.Net.SQLiteConnection(new SQLite.Net.Platform.WinRT.SQLitePlatformWinRT(), App.DB\_PATH))

{

var existingusr = conn.Query<Users>("select \* from Contacts where Id =" + user\_id).FirstOrDefault();

return existingusr;

}

}

public ObservableCollection<Users> ReadAllUsers()

{

try

{

using (SQLite.Net.SQLiteConnection conn = new SQLite.Net.SQLiteConnection(new SQLite.Net.Platform.WinRT.SQLitePlatformWinRT(), App.DB\_PATH))

{

List<Users> myCollection = conn.Table<Users>().ToList<Users>();

ObservableCollection<Users> listUsers = new ObservableCollection<Users>(myCollection);

return listUsers;

}

}

catch

{

return null;

}

}

public ObservableCollection<Hotel> ReadAllHotels()

{

try

{

using (SQLite.Net.SQLiteConnection conn = new SQLite.Net.SQLiteConnection(new SQLite.Net.Platform.WinRT.SQLitePlatformWinRT(), App.DB\_PATH))

{

List<Hotel> myCollection = conn.Table<Hotel>().ToList<Hotel>();

ObservableCollection<Hotel> listRmT = new ObservableCollection<Hotel>(myCollection);

return listRmT;

}

}

catch

{

return null;

}

}

public ObservableCollection<RoomType> ReadAllRoomTypes()

{

try

{

using (SQLite.Net.SQLiteConnection conn = new SQLite.Net.SQLiteConnection(new SQLite.Net.Platform.WinRT.SQLitePlatformWinRT(), App.DB\_PATH))

{

List<RoomType> myCollection = conn.Table<RoomType>().ToList<RoomType>();

ObservableCollection<RoomType> listRmT = new ObservableCollection<RoomType>(myCollection);

return listRmT;

}

}

catch

{

return null;

}

}

public ObservableCollection<Reservations> ReadAllReservations()

{

try

{

using (SQLite.Net.SQLiteConnection conn = new SQLite.Net.SQLiteConnection(new SQLite.Net.Platform.WinRT.SQLitePlatformWinRT(), App.DB\_PATH))

{

List<Reservations> myCollection = conn.Table<Reservations>().ToList<Reservations>();

ObservableCollection<Reservations> listReservations = new ObservableCollection<Reservations>(myCollection);

return listReservations;

}

}

catch

{

return null;

}

}

}

public class ReadAllUserList

{

DatabaseHelperClass Db\_Helper = new DatabaseHelperClass();

public ObservableCollection<Users> GetAllContacts()

{

return Db\_Helper.ReadAllUsers();

}

}

* View Model SignIn(свързва се със страницата за вписване):

public class SigIn : ViewModelBase

{

private string \_usrNm;

private string \_pass;

private ICommand \_sigIn;

private bool \_canExecute;

private string \_info;

public string info

{

get

{

return \_info;

}

set

{

\_info = value;

NotifyPropertyChanged();

}

}

public string usrNm

{

get

{

return \_usrNm;

}

set

{

\_usrNm = value;

NotifyPropertyChanged();

}

}

public string pass

{

get

{

return \_pass;

}

set

{

\_pass = value;

NotifyPropertyChanged();

}

}

public ICommand sigIn

{

get

{

return \_sigIn ?? (\_sigIn = new DelegateCommand(() => getData(), \_canExecute));

}

}

public SigIn()

{

\_canExecute = true;

}

public void getData()

{

Users usr = new Users();

string message = "No user with this username and password!";

string name, passW;

bool result;

name = usrNm;

passW = pass;

result = searchUser(name, passW);

if (result == false)

{

info = message.ToString();

}

else

{

info = null;

}

}

public bool searchUser(string usName, string passW)

{

bool result = false;

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Users> list = Db\_helper.ReadAllUsers();

foreach (Users selUsr in list)

{

try

{

if (usName.Equals(selUsr.userName) && passW.Equals(selUsr.password))

{

result = true;

}

}

catch (NullReferenceException e)

{

result = false;

}

}

return result;

}

}

* View Model ConnectToUsers(свързва се със страницата за регистрация):

public class ConnectToUsers : ViewModelBase

{

private Users \_usr;

private string userNm;

private string pass;

private string firstN;

private string lastN;

private string ph;

private ICommand \_addUsr;

private bool \_canExecute;

private string \_err;

#region errors

const string emptyFieldsError = "Fill the empty fields";

const string existingUsernameError = "This user name already exists";

const string emptyUsernameError = "Username cannot be an empty field";

const string emptyPasswordError = "Password cannot be an empty field";

const string emptyFNameError = "First name cannot be an empty field";

const string emptyLNameError = "Last name cannot be an empty field";

const string emptyPhoneError = "Phone cannot be an empty field";

const string nameRequirementError = "Name must contain only letters";

const string passwordRequirementError = "Password must contain upper and lower letters and digits";

const string phoneRequirementError = "Wrong phone";

#endregion

#region getters and setters

public string Err

{

get

{

return \_err;

}

set

{

\_err = value;

NotifyPropertyChanged();

}

}

public string userName

{

get

{

return userNm;

}

set

{

userNm = value;

NotifyPropertyChanged();

}

}

public string password

{

get

{

return pass;

}

set

{

pass = value;

NotifyPropertyChanged();

}

}

public string fName

{

get

{

return firstN;

}

set

{

firstN = value;

NotifyPropertyChanged();

}

}

public string lName

{

get

{

return lastN;

}

set

{

lastN = value;

NotifyPropertyChanged();

}

}

public string phone

{

get

{

return ph;

}

set

{

ph = value;

NotifyPropertyChanged();

}

}

public Users usr

{

get

{

return \_usr;

}

set

{

\_usr = value;

NotifyPropertyChanged();

}

}

public ICommand addUsr

{

get

{

return \_addUsr ?? (\_addUsr = new DelegateCommand(() => getData(), \_canExecute));

}

}

#endregion

#region get and check user input

public void getData()

{

string usName, passW = null, first = null, second = null, mob = null;

try

{

if (checkUserName() == true)

{

Err = existingUsernameError;

}

else if (!Regex.IsMatch(fName, "^[\\p{L} \\.\\-]+$") || !Regex.IsMatch(lName, "^[\\p{L} \\.\\-]+$"))

{

Err = nameRequirementError;

}

else if (!Regex.IsMatch(password, "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d).{8,15}$"))

{

Err = passwordRequirementError;

}

else if (!Regex.IsMatch(phone, @"^\+?[0-9][0-9\s.-]{7,11}$"))

{

Err = phoneRequirementError;

}

else

{

Err = null;

}

}

catch (Exception e)

{

Err = emptyFieldsError;

}

if (Err == null)

{

usName = userName;

passW = password;

first = fName;

second = lName;

mob = phone;

DatabaseHelperClass Db\_Helper = new DatabaseHelperClass();

Db\_Helper.InsertUser(new Users(usName, passW, first, second, mob));

}

else

{

ConnectToUsers cn = new ConnectToUsers();

}

}

public bool checkUserName()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Users> listUsr = Db\_helper.ReadAllUsers();

bool isRegistrated = false;

foreach (Users u in listUsr)

{

if (userName.Equals(u.userName))

{

isRegistrated = true;

}

}

return isRegistrated;

}

#endregion

public ConnectToUsers()

{

\_canExecute = true;

}

}

* View Model UserDataVM(свързва се със страницата за информация за потребителя):

public class UserDataVM : ViewModelBase

{

private string \_userName;

private string \_info;

private ObservableCollection<Reservations> \_reservations;

private string \_hotelName;

private string \_from;

private string \_to;

private string \_roomType;

private ObservableCollection<string> \_listRes;

public string Info

{

get

{

return \_info;

}

set

{

\_info = value;

NotifyPropertyChanged();

}

}

public ObservableCollection<string> ListRes

{

get

{

return \_listRes;

}

set

{

\_listRes = value;

NotifyPropertyChanged();

}

}

public string RoomTyp

{

get

{

return \_roomType;

}

set

{

\_roomType = value;

NotifyPropertyChanged();

}

}

public string UserName

{

get

{

return \_userName;

}

set

{

\_userName = value;

NotifyPropertyChanged();

}

}

public ObservableCollection<Reservations> Reservation

{

get

{

return \_reservations;

}

set

{

\_reservations = value;

NotifyPropertyChanged();

}

}

public string HotelName

{

get

{

return \_hotelName;

}

set

{

\_hotelName = value;

NotifyPropertyChanged();

}

}

public string From

{

get

{

return \_from;

}

set

{

\_from = value;

NotifyPropertyChanged();

}

}

public string To

{

get

{

return \_to;

}

set

{

\_to = value;

NotifyPropertyChanged();

}

}

public void getInfo()

{

ObservableCollection<int> hotelsId = getHotelId();

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Hotel> list = Db\_helper.ReadAllHotels();

ListRes = new ObservableCollection<string>();

foreach (Hotel hotel in list)

{

foreach (int htl in hotelsId)

{

if (htl == hotel.id)

{

HotelName = hotel.name;

Info = HotelName + ", " + RoomTyp+ ",from:" + From+ ",to:" + To;

ListRes.Add(Info);

}

}

}

}

}

public ObservableCollection<int> getHotelId()

{

ObservableCollection<int> hotelsId = new ObservableCollection<int>();

ObservableCollection<int> roomsId = getReservations();

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<RoomType> list = Db\_helper.ReadAllRoomTypes();

foreach (RoomType room in list)

{

foreach (int rm in roomsId)

{

if (rm == (room.id))

{

RoomTyp = room.type;

hotelsId.Add(room.hotel\_id);

}

}

}

return hotelsId;

}

public ObservableCollection<int> getReservations()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

Reservation = Db\_helper.ReadAllReservations();

ObservableCollection<int> roomsId = new ObservableCollection<int>();

int usrId = getUserId();

foreach (Reservations reservation in Reservation)

{

if (reservation.user\_id == usrId)

{

From = reservation.dateFrom.ToString("dd/MM/yyyy");

To = reservation.dateTo.ToString("dd/MM/yyyy");

roomsId.Add(reservation.room\_id);

}

}

return roomsId;

}

public int getUserId()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Users> list = Db\_helper.ReadAllUsers();

int usrId = 0;

foreach (Users usr in list)

{

if (UserName.Equals(usr.userName))

{

usrId = usr.id;

}

}

return usrId;

}

}

* View Model CatalogVM(свързва се със страницата за каталога хотели):

public class CatalogVM : ViewModelBase

{

private ObservableCollection<string> \_hotelNames;

private string \_username;

private string \_name;

public ObservableCollection<string> HotelNames

{

get

{

return \_hotelNames;

}

set

{

\_hotelNames = value;

NotifyPropertyChanged();

}

}

public string UserName

{

get

{

return \_username;

}

set

{

\_username = value;

NotifyPropertyChanged();

}

}

public string name

{

get

{

return \_name;

}

set

{

\_name = value;

NotifyPropertyChanged();

}

}

public void SelectHotel()

{

}

public CatalogVM()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Hotel> list = Db\_helper.ReadAllHotels();

HotelNames = new ObservableCollection<string>();

foreach (Hotel item in list)

{

name = item.name;

HotelNames.Add(name);

}

}

}

* View Model HotelOverviewVM(свързва се със страницата за информация за хотел):

public class HotelOverviewVM : ViewModelBase

{

private string \_username;

private string \_name;

private string \_city;

private string \_address;

private string \_phone;

private string \_stars;

private string \_picPath;

private string \_description;

private string \_rmType;

private String \_price;

private ObservableCollection<string> \_listTypes;

private ObservableCollection<string> \_listPrices;

public string UserName

{

get

{

return \_username;

}

set

{

\_username = value;

NotifyPropertyChanged();

}

}

public string Name

{

get

{

return \_name;

}

set

{

\_name = value;

NotifyPropertyChanged();

}

}

public string City

{

get

{

return \_city;

}

set

{

\_city = value;

NotifyPropertyChanged();

}

}

public string Address

{

get

{

return \_address;

}

set

{

\_address = value;

NotifyPropertyChanged();

}

}

public string Phone

{

get

{

return \_phone;

}

set

{

\_phone = value;

NotifyPropertyChanged();

}

}

public string Stars

{

get

{

return \_stars;

}

set

{

\_stars = value;

NotifyPropertyChanged();

}

}

public string PicPath

{

get

{

return \_picPath;

}

set

{

\_picPath = value;

NotifyPropertyChanged();

}

}

public string Description

{

get

{

return \_description;

}

set

{

\_description = value;

NotifyPropertyChanged();

}

}

public string RmType

{

get

{

return \_rmType;

}

set

{

\_rmType = value;

NotifyPropertyChanged();

}

}

public string Price

{

get

{

return \_price;

}

set

{

\_price = value;

NotifyPropertyChanged();

}

}

public ObservableCollection<string> ListTypes

{

get

{

return \_listTypes;

}

set

{

\_listTypes = value;

NotifyPropertyChanged();

}

}

public ObservableCollection<string> ListPrices

{

get

{

return \_listPrices;

}

set

{

\_listPrices = value;

NotifyPropertyChanged();

}

}

public void setStar(int num)

{

if (num == 1)

{

Stars = "★";

}

else if (num == 2)

{

Stars = "★★";

}

else if (num == 3)

{

Stars = "★★★";

}

else if (num == 4)

{

Stars = "★★★★";

}

else if (num == 5)

{

Stars = "★★★★★";

}

}

public void viewHoteInfo()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Hotel> listHotels = Db\_helper.ReadAllHotels();

ObservableCollection<RoomType> listRmTypes = Db\_helper.ReadAllRoomTypes();

ListTypes = new ObservableCollection<string>();

ListPrices = new ObservableCollection<string>();

int num = 0;

int hotelId = getHotelId();

foreach (Hotel item in listHotels)

{

if (item.id == hotelId)

{

Name = item.name;

City = item.city;

Address = item.address;

Phone = item.phone;

num = item.stars;

setStar(num);

PicPath = item.picPath;

Description = item.description;

foreach (RoomType rmTyp in listRmTypes)

{

if (rmTyp.hotel\_id == item.id)

{

RmType = rmTyp.type;

Price = rmTyp.price + " lv.";

ListPrices.Add(Price);

ListTypes.Add(RmType);

}

}

}

}

}

public int getHotelId()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Hotel> list = Db\_helper.ReadAllHotels();

int htlId = 0;

string n = this.Name;

foreach (Hotel htl in list)

{

if (n.Equals(htl.name))

{

htlId = htl.id;

}

}

return htlId;

}

}

* View Model ReservationForm(свързва се със страницата за резервация):

public class ReservationForm : ViewModelBase

{

private string \_usrName;

private string \_fName;

private string \_lName;

private string \_htlName;

private string \_rmType;

private DateTimeOffset \_from;

private DateTimeOffset \_to;

private int \_rmCount;

private ObservableCollection<string> \_types;

private string \_error;

private ICommand \_btnReserve;

private bool \_canExecute;

public static string DB\_PATH = Path.Combine(Path.Combine(ApplicationData.Current.LocalFolder.Path, "db.sqlite"));

#region errors

const string datesPopulationError = "Dates are not accurately populated";

const string noRoomsAvailableError = "No rooms available";

const string noRoomsSelectedError = "Select more than 0 rooms!";

#endregion

#region getters and setters

public string Error

{

get

{

return \_error;

}

set

{

\_error = value;

NotifyPropertyChanged();

}

}

public string UserName

{

get

{

return \_usrName;

}

set

{

\_usrName = value;

NotifyPropertyChanged();

}

}

public string FName

{

get

{

return \_fName;

}

set

{

\_fName = value;

NotifyPropertyChanged();

}

}

public string LName

{

get

{

return \_lName;

}

set

{

\_lName = value;

NotifyPropertyChanged();

}

}

public string HtlName

{

get

{

return \_htlName;

}

set

{

\_htlName = value;

NotifyPropertyChanged();

}

}

public string RoomType

{

get

{

return \_rmType;

}

set

{

\_rmType = value;

NotifyPropertyChanged();

}

}

public DateTimeOffset From

{

get

{

return \_from;

}

set

{

\_from = value;

NotifyPropertyChanged();

}

}

public DateTimeOffset To

{

get

{

return \_to;

}

set

{

\_to = value;

NotifyPropertyChanged();

}

}

public int RmCount

{

get

{

return \_rmCount;

}

set

{

\_rmCount = value;

NotifyPropertyChanged();

}

}

public ObservableCollection<string> Types

{

get

{

return \_types;

}

set

{

\_types = value;

NotifyPropertyChanged();

}

}

public ICommand BtnReserve

{

get

{

return \_btnReserve ?? (\_btnReserve = new DelegateCommand(() => saveToDb(), \_canExecute));

}

}

#endregion

public void getUser()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Users> listUsr = Db\_helper.ReadAllUsers();

foreach (Users usr in listUsr)

{

if (UserName.Equals(usr.userName))

{

FName = usr.fName;

LName = usr.lName;

}

}

}

public void getHotel()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Hotel> listHtl = Db\_helper.ReadAllHotels();

ObservableCollection<RoomType> listRmT = Db\_helper.ReadAllRoomTypes();

Types = new ObservableCollection<string>();

foreach (Hotel hotel in listHtl)

{

if (HtlName.Equals(hotel.name))

{

foreach (RoomType rmT in listRmT)

{

if (rmT.hotel\_id == hotel.id)

{

RoomType = rmT.type;

Types.Add(RoomType);

}

}

}

}

}

public void getData()

{

getUser();

getHotel();

}

public int getHtlId()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Hotel> listHtl = Db\_helper.ReadAllHotels();

int id = 0;

foreach (Hotel hotel in listHtl)

{

if (HtlName.Equals(hotel.name))

{

id = hotel.id;

}

}

return id;

}

public int getRmId()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<RoomType> listRmT = Db\_helper.ReadAllRoomTypes();

int htlId = getHtlId();

int rmId = 0;

foreach (RoomType rmT in listRmT)

{

if ((rmT.hotel\_id == htlId) && (rmT.type.Equals(RoomType)))

{

rmId = rmT.id;

}

}

return rmId;

}

public int getUsrId()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Users> listUsr = Db\_helper.ReadAllUsers();

int user\_id = 0;

foreach (Users usr in listUsr)

{

if (UserName.Equals(usr.userName))

{

user\_id = usr.id;

}

}

return user\_id;

}

public Reservations reservation()

{

Reservations res = new Reservations();

int remaingngCnt = getRemainingRmCnt();

if (RmCount < remaingngCnt && RmCount > 0)

{

res.room\_id = getRmId();

res.user\_id = getUsrId();

if (From >= DateTime.Now && To > From)

{

res.dateFrom = From;

res.dateTo = To;

Error = null;

}

else

{

Error = datesPopulationError;

}

res.count = RmCount;

}

else if (RmCount > remaingngCnt)

{

Error = noRoomsAvailableError;

}

else if (RmCount <= 0)

{

Error = noRoomsSelectedError;

}

return res;

}

public void saveToDb()

{

Reservations reserve;

reserve = reservation();

DateTimeOffset fr = reserve.dateFrom;

DateTimeOffset to = reserve.dateTo;

int usrId = reserve.user\_id;

int rmId = reserve.room\_id;

int cnt = reserve.count;

if (Error == null)

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

Db\_helper.InsertReservation(new Reservations(fr, to, usrId, rmId, cnt));

}

else if (Error != null)

{

ReservationForm rf = new ReservationForm();

}

}

public int countReservations()

{

int count = 0;

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Reservations> listReserv = Db\_helper.ReadAllReservations();

int rmId = getRmId();

DateTimeOffset fr = From;

DateTimeOffset to = To;

foreach (Reservations r in listReserv)

{

if (r.room\_id == rmId && ((r.dateFrom.Equals(fr)) ||

(r.dateFrom < fr && r.dateTo > fr) || (r.dateFrom < to && r.dateTo > to)))

{

count++;

}

}

return count;

}

public int getAllCnt()

{

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<RoomType> listRmT = Db\_helper.ReadAllRoomTypes();

int allRooms;

int rmId = getRmId();

RoomType room = new RoomType();

foreach (RoomType rmT in listRmT)

{

if (rmT.id == rmId)

{

room = rmT;

}

}

allRooms = room.count;

return allRooms;

}

public int getRemainingRmCnt()

{

int result;

int allRm = getAllCnt();

int currReserved = countReservations();

result = allRm - currReserved;

return result;

}

public ReservationForm()

{

\_canExecute = true;

}

}

* View Model MapViewModel(свързва се със страницата за картата):

public class MapViewModel : ViewModelBase

{

private string \_usrName;

private string \_htlName;

private string \_address;

private string \_myAddress;

public string UserName

{

get

{

return \_usrName;

}

set

{

\_usrName = value;

NotifyPropertyChanged();

}

}

public string HotelName

{

get

{

return \_htlName;

}

set

{

\_htlName = value;

NotifyPropertyChanged();

}

}

public string Address

{

get

{

return \_address;

}

set

{

\_address = value;

NotifyPropertyChanged();

}

}

public void getLocation()

{

string location = null;

DatabaseHelperClass Db\_helper = new DatabaseHelperClass();

ObservableCollection<Hotel> listHotels = Db\_helper.ReadAllHotels();

foreach (Hotel hotel in listHotels)

{

if (HotelName.Equals(hotel.name))

{

location = hotel.city + ", " + hotel.address;

}

}

Address = location;

}

}

* View MapView(code behind на страницата с картата):

public sealed partial class MapView : Page

{

public MapView()

{

this.InitializeComponent();

}

protected async override void OnNavigatedTo(NavigationEventArgs e)

{

base.OnNavigatedTo(e);

string[] parameters = new string[2];

parameters = (string[])e.Parameter;

var vm = new MapViewModel();

vm.UserName = parameters[0];

vm.HotelName = parameters[1];

vm.getLocation();

this.DataContext = vm;

string address = getAddress();

var results = await MapLocationFinder.FindLocationsAsync(address, null);

double lat = 0.0;

double lon = 0.0;

if (results.Status == MapLocationFinderStatus.Success)

{

lat = results.Locations[0].Point.Position.Latitude;

lon = results.Locations[0].Point.Position.Longitude;

}

BasicGeoposition position = new BasicGeoposition()

{

Latitude = lat,

Longitude = lon

};

Geopoint gPoint = new Geopoint(position);

MapIcon mapIcon1 = new MapIcon();

mapIcon1.Location = gPoint;

mapIcon1.NormalizedAnchorPoint = new Point(0.5, 1.0);

mapIcon1.Title = "Your Hotel";

mapIcon1.ZIndex = 0;

// Add the MapIcon to the map.

mapControl.MapElements.Add(mapIcon1);

// Center the map over the POI.

mapControl.Center = gPoint;

mapControl.ZoomLevel = 14;

}

public string getAddress()

{

string adr;

adr = address.Text;

return adr;

}

private async void Directions\_Click(object sender, RoutedEventArgs e)

{

Geopoint begin = null;

Geopoint end = null;

foreach (MapIcon item in mapControl.MapElements)

{

if (item.Title.Equals("You are here"))

{

begin = item.Location;

}

else if (item.Title.Equals("Your Hotel"))

{

end = item.Location;

}

}

try

{

MapRouteFinderResult routeResult = await MapRouteFinder.GetDrivingRouteAsync(begin, end);

if (routeResult.Status == MapRouteFinderStatus.Success)

{

// Use the route to initialize a MapRouteView.

MapRouteView viewOfRoute = new MapRouteView(routeResult.Route);

viewOfRoute.RouteColor = Colors.Yellow;

viewOfRoute.OutlineColor = Colors.Black;

// Add the new MapRouteView to the Routes collection

// of the MapControl.

mapControl.Routes.Add(viewOfRoute);

// Fit the MapControl to the route.

await mapControl.TrySetViewBoundsAsync(

routeResult.Route.BoundingBox,

null,

Windows.UI.Xaml.Controls.Maps.MapAnimationKind.None);

System.Text.StringBuilder routeInfoKm = new System.Text.StringBuilder();

System.Text.StringBuilder routeInfoTime = new System.Text.StringBuilder();

routeInfoTime.Append("Total estimated time = ");

routeInfoTime.Append(routeResult.Route.EstimatedDuration.TotalMinutes.ToString());

routeInfoTime.Append("min");

routeInfoKm.Append("\nTotal length = ");

routeInfoKm.Append((routeResult.Route.LengthInMeters / 1000).ToString());

routeInfoKm.Append("km");

infoKm.Text = routeInfoKm.ToString();

infoTime.Text = routeInfoTime.ToString();

}

}

catch (NullReferenceException ex)

{

err.Text = "Your location must be set!";

}

}

private async void btnMyLocation\_Click(object sender, RoutedEventArgs e)

{

err.Text = "";

var geoLocator = new Geolocator();

geoLocator.DesiredAccuracy = PositionAccuracy.High;

Geoposition pos = await geoLocator.GetGeopositionAsync();

double latitude = pos.Coordinate.Point.Position.Latitude;

double longitude = pos.Coordinate.Point.Position.Longitude;

BasicGeoposition position = new BasicGeoposition()

{

Latitude = latitude,

Longitude = longitude

};

Geopoint gPoint = new Geopoint(position);

MapIcon mapIcon1 = new MapIcon();

mapIcon1.Location = gPoint;

mapIcon1.NormalizedAnchorPoint = new Point(0.5, 1.0);

mapIcon1.Title = "You are here";

mapIcon1.ZIndex = 0;

// Add the MapIcon to the map.

mapControl.MapElements.Add(mapIcon1);

// Center the map over the POI.

mapControl.Center = gPoint;

mapControl.ZoomLevel = 14;

}

private void btnBack\_Click(object sender, RoutedEventArgs e)

{

string[] parameters = new string[2];

parameters[0] = usrName.Text;

parameters[1] = htlName.Text;

Frame.Navigate(typeof(HotelOverview), parameters);

}

private void btnSignOut\_Click(object sender, RoutedEventArgs e)

{

Frame.Navigate(typeof(NavigationPage));

}

private void btnHome\_Click(object sender, RoutedEventArgs e)

{

string usName = usrName.Text;

Frame.Navigate(typeof(UserData), usName);

}

}

* Свързване с базата данни и добавяне на информация при стартиране на приложението:

sealed partial class App : Application

{

public static string DB\_PATH = Path.Combine(Path.Combine(ApplicationData.Current.LocalFolder.Path, "db.sqlite"));

/// <summary>

/// Initializes the singleton application object. This is the first line of authored code

/// executed, and as such is the logical equivalent of main() or WinMain().

/// </summary>

public App()

{

this.InitializeComponent();

this.Suspending += OnSuspending;

if (!CheckFileExists("db.sqlite").Result)

{

using (var conn = new SQLiteConnection(new SQLite.Net.Platform.WinRT.SQLitePlatformWinRT(), DB\_PATH))

{

conn.CreateTable<Users>();

conn.CreateTable<Hotel>();

conn.CreateTable<Reservations>();

conn.CreateTable<RoomType>();

putData(conn);

}

}

}

public void putData(SQLiteConnection conn)

{

conn.Insert(new Hotel() {

id = 1,

name = "Troyan Plaza",

city = "Troyan",

address = "ul. P.R. Slavejkov 54",

phone = "0885100006",

stars = 4,

picPath = "/Images/TroyanPlaza.jpg",

description = " One of our bestsellers in Troyan!Located on the bank of the Beli Ossum River in the centre of Troyan, Troyan Plaza Hotel features a fitness and spa centre.It offers spacious rooms with free LAN access."

});

conn.Insert(new RoomType() {

id=1,

type="2 beds",

price=50.00,

count=20,

hotel\_id=1

});

……………

}