**Міністерство освіти і науки України**

**Національний університет «Запорізька політехніка»**

Кафедра програмних засобів

**ЗВІТ**

Дисципліна «Фреймворки розробки програмного забезпечення»

Робота №8

Тема «Оформлення авторського свідоцтва на розроблену програму»

**Виконав варіант 19**

Студент КНТ-122 Онищенко О. А.

**Прийняли**

Викладач Зелік О. В.

2024

**Мета**: Засвоїти основні поняття про порядок та особливості оформлення авторського свідоцтва в Україні на розроблену програму, навчитися виділяти та представляти отримані результати.

**Завдання**: Розглянути розроблену програмну систему на предмет відповідності поставленому ТЗ. Оформити настанову щодо використання створеного програмного продукту, що має складатися з наступних елементів:

* Призначення програми;
* Структура системи;
* Функціонування системи;
* Структура та органіація даних;
* Основні функції;
* Інтерфейсні засоби;
* Вимоги до програмного та апаратного забезпечення;
* Методика роботи користувача з системою;

Оформити заяву про реєстрацію авторського права на твір. Оформити договір між роботодавцем та авторами комп’ютерної програми. Створити презентацію для представлення результатів розробленого продукту.

Настанова щодо використання

Призначення програми

Програма призначена для купівлі та продажу об’єктів нерухомості.

Структура системи

Система поділена на два рівні: видимий користувачеві (фронтенд) та невидимий користувачеві (бекенд).

Рівень фронтенду реалізовано мовою C# і фреймворком WPF. Елементи інтерфейсу додано викорстовуючи існуючі віджети WPF зі зміненими стилями.

На рівні бекенду система розподілена на два шари: функції доступні фронтенду (моделі представлення) та класи й методи взаємодії з даними (моделі). Фунції шару моделі представлення використовуються елементами інтерфейсу, а у самих функціях виконується доступ до рівню моделей та допоміжних класів.

Рівень фронтенду:

* LoginPage
* ProfilePage

Рівень бекенду:

* Database
* Estate
* EstateKind
* Meeting
* MeetingScore
* MeetingStatus
* User
* UtilityFunctions
* UtilityVariables

Кожен пункт рівню фронтенду поділено на два файли: типу XAML та типу CS. Файл першого типу визначає вигляд інтерфейсу користувача, файл другого типу виконує роль шару моделі представлення і з’єднує інтерфейс користувача із рівнем моделей.

Функціонування системи

Система починає роботу із запуску сторінку входу. На ній користувач має ввести ім’я та увійти у свій кабінет. Після входу користувачеві доступна сторінка профілю, на якій по вкладках представлені елементи системи.

Вкладки основної сторінки програми:

* Домівка (Home) надає інформація про особисті дані користувача: ім’я, баланс, статус.
* Ринок (Available Estate) показує доступні для придбання оголошення.
* Наявні оголошення (Owned Estate) показує об’єкти нерухомості у наявності користувача.
* Вхідні зустрічі (Incoming Meetings) показує список зустрічей для оголошень користувача.
* Вихідні зустрічі (Outgoing Meetings) виводить список зустічей, призначених користувачем.

Структура та органіація даних

Дані програми зберігаються на базі MySQL та організовані у три таблиці: user, estate, meeting.

Таблиця user має поля:

* name (текстове поле): ім’я користувача
* admin (булеве значення): статус менеджера
* balance (цілочисельне значення): баланс

Таблиця estate має поля:

* owner\_id (цілочисельне значення): зовнішній ключ до ідентифікатора користувача власника нерухомості
* title (текстове поле): назва нерухомості
* kind (текстове поле з обмеженим вибором): тип нерухомості, може бути Home (приватний будинок), Flat (квартира у будинку), New (квартира у новобудові)
* price (цілочисельне значення): вартість нерухомості

Таблиця meeting має поля:

* sender\_id (цілочисельне значення): зовнішній ключ до ідентифікатора користувача замовника зустрічі
* target\_id (цілочисельне значення): зовнішній ключ до ідентифікатора об’єкта нерухомості який переглядається
* status (текстове поле з обмеженим вибором): статус зустрічі, може бути Wait (зустріч чекає на опрацювання власником оголошення), Done (зустріч відбулася), Skip (власник оголошення відмовився від зустрічі)
* score (текстове поле з обмеженим вибором): оцінка користувача переглядача оголошення, може бути Bad (не сподобалося), Okay (нормально), Fine (сподобалося)

Кожен запис усіх таблиць має поле унікального ідентифікатора запису id типу int (ціле число).

Програма підключається до бази даних через доєднувач MySQL і опрацьовує дані у класи моделей: User, Estate, Meeting.

Клас даних User має поля:

* ID (ціле число): унікальний ідентифікатор користувача
* Name (рядок): ім’я користувача
* Admin (ціле число): позначка менеджера, за замовчуванням Нема
* Balance (ціле число): поточний баланс, за замовчуванням 0

Клас даних Estate має поля:

* ID (ціле число): унікальний ідентифікатор об’єкта
* Owner (об’єкт User): користувач власник нерухомості
* Title (рядок): назва
* Kind (рядок): тип
* Price (ціле число): ціна, за замовчуванням 0

Клас даних Meeting має поля:

* ID (ціле число): унікальний ідентифікатор зустрічі
* Sender (об’єкт User): користувач переглядач оголошення
* Target (об’єкт Estate): об’єкт нерухомості на перегляд
* Score (рядок): оцінка зустрічі, за замовчуванням Unrated
* Status (рядок): статус зустрічі, за замовчуванням Wait

Доєднувач MySQL працює з базою даних та використовує класи даних для створення об’єктів з прочитаної інформації. Клас роботи з базою даних Database реалізує базові операції з даними: додавання, читання, редагування, видалення.

Програма також містить статичні класи з константними назвами для полів, які мають обмежений вибір: MeetingScore, EstateKind, MeetingStatus. Такі класи необхідні для уникнення помилок написання константних значень та для увімкнення підказок у редакторі коду. Класи є статичними і не потребують оголошення об’єктів їхнього типу бо всі дані в них є незмінними.

Клас MeetingScore містить такі дані:

* Bad (рядок): значення Bad
* Okay (рядок): значення Okay
* Fine (рядок): значення Fine

Клас MeetingStatus містить такі дані:

* Wait (рядок): значення Wait
* Done (рядок): значення Done
* Skip (рядок): значення Skip

Клас EstateKind містить такі дані:

* Home (рядок): значення Home
* Flat (рядок): значення Flat
* New (рядок): значення New

Програма містить класи для допоміжних функцій та змінних: UtilityFunctions та UtilityVariables.

Клас UtilityFunctions містить методи:

* checkEstateKind(kind): перевірити значення типу нерухомості; приймає введене значення типу як параметр та повертає булевий результат на виході
* checkMeetingStatus(status): перевірити значення статусу зустрічі; приймає введене значення статусу як параметр та повертає булевий результат на виході
* checkMeetingScore(score): перевірити значення оцінки зустрічі; приймає введене значення оцінки як параметр та повертає булевий результат на виході

Клас UtilityVariables містить змінну:

* connectionString (рядок): сформований рядок ідентифікаційної інформації для з’єднання з базою даних

Основні функції

Інтерфейсні засоби

Вимоги до програмного та апаратного забезпечення

Методика роботи користувача з системою

Коли користувач взаємодіє з системою через графічний інтерфейс, програма опрацьовує запити (події) та виконує функції через шар моделі представлення. Методи шару моделі представлення виконують запити до бази даних через клас моделі Database, отримані дані передають на інтерфейс який виводить оновлену інформацію користувачеві після взаємодії.

Основне меню програми розділене на вкладки, кожна з яких має дані. При переході на нову вкладку дані на ній оновлюються моделлю представлення. При отриманні результатів від бази даних викорастані засоби LINQ для специфічного фільтрування даних

Заява про реєстрацію авторського права

Договір між роботодавцем та автором

Фрагмент коду програми

public class Database {

string query;

MySqlCommand command;

MySqlDataReader reader;

MySqlConnection connection;

public const string LastCreatedID="SELECT LAST\_INSERT\_ID();";

// SYSTEM

public Database(MySqlConnection connection) { this.connection = connection; }

public MySqlDataReader read(string query){

command=new MySqlCommand(query,connection);

return command.ExecuteReader();

}

public void write(string query){

command = new MySqlCommand(query, connection);

command.ExecuteNonQuery();

}

// CREATE

public User createUser(string name, int admin = 0, int balance=0) {

write($"INSERT INTO user (name,admin,balance) VALUES ('{name}',{admin},{balance});");

reader=read(LastCreatedID);

var user = new User();

while (reader.Read()) {

user.ID = reader.GetInt32(0);

user.Name = name;

user.Admin = admin;

user.Balance=balance;

}

reader.Close();

return user;

}

public Estate createEstate(string title, string kind, User owner, int price) {

write($"INSERT INTO estate (title,kind,owner\_id,price) VALUES ('{title}','{kind}',{owner.ID},{price});");

reader=read(LastCreatedID);

var estate = new Estate();

while (reader.Read()) {

estate.ID = reader.GetInt32(0);

estate.Title = title;

estate.Kind = kind;

estate.Owner = owner;

estate.Price=price;

}

reader.Close();

return estate;

}

public Meeting createMeeting(User sender,Estate target){

write($"INSERT INTO meeting (sender\_id,target\_id) VALUES ({sender.ID},{target.ID});");

reader=read(LastCreatedID);

var meeting = new Meeting();

while (reader.Read()) {

meeting.ID = reader.GetInt32(0);

meeting.Sender=sender;

meeting.Target=target;

}

reader.Close();

return meeting;

}

// READ

public List<User> getUsers(){

reader=read("SELECT id,name,admin,balance FROM user;");

var users=new List<User>();

while (reader.Read()) {

var user = new User();

user.ID = reader.GetInt32(0);

user.Name = reader.GetString(1);

user.Admin = reader.GetInt32(2);

user.Balance=reader.GetInt32(3);

users.Add(user);

}

reader.Close();

return users;

}

public User getUser(int id){

return getUsers().Where(u=>u.ID==id).ToList().ElementAtOrDefault(0);

}

public List<Estate> getEstates(){

reader=read("SELECT id,owner\_id,title,kind,price FROM estate;");

var estates=new List<Estate>();

var owners=new List<int>();

while (reader.Read()){

var estate=new Estate();

estate.ID=reader.GetInt32(0);

owners.Add(reader.GetInt32(1));

estate.Title=reader.GetString(2);

estate.Kind=reader.GetString(3);

estate.Price=reader.GetInt32(4);

estates.Add(estate);

}

reader.Close();

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

estates[i].Owner=getUser(owners[i]);

}

return estates;

}

public Estate getEstate(int id){

return getEstates().Where(e=>e.ID==id).ToList().ElementAtOrDefault(0);

}

public List<Meeting> getMeetings(){

reader=read("SELECT id,sender\_id,target\_id,score,status FROM meeting;");

var meetings=new List<Meeting>();

var senders = new List<int>();

var targets = new List<int>();

while (reader.Read()) {

var meeting = new Meeting();

meeting.ID = reader.GetInt32(0);

senders.Add(reader.GetInt32(1));

targets.Add(reader.GetInt32(2));

meeting.Score = reader.GetString(3);

meeting.Status = reader.GetString(4);

meetings.Add(meeting);

}

reader.Close();

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

meetings[i].Sender = getUser(senders[i]);

meetings[i].Target = getEstate(targets[i]);

}

return meetings;

}

public Meeting getMeeting(int id){

return getMeetings().Where(m=>m.ID==id).ToList().ElementAtOrDefault(0);

}

// UPDATE

public void updateUser(User user) {

write($"UPDATE user SET name='{user.Name}',admin={user.Admin},balance={user.Balance} WHERE id={user.ID};");

}

public void updateEstate(Estate estate) {

write($"UPDATE estate SET owner\_id={estate.Owner.ID},title='{estate.Title}',kind='{estate.Kind}',price={estate.Price} WHERE id={estate.ID};");

}

public void updateMeeting(Meeting meeting) {

write($"UPDATE meeting SET sender\_id={meeting.Sender.ID},target\_id={meeting.Target.ID},score='{meeting.Score}',status='{meeting.Status}' WHERE id={meeting.ID};");

}

// DELETE

public void deleteUser(int id) {

write($"DELETE FROM user WHERE id={id};");

}

public void deleteEstate(int id) {

write($"DELETE FROM estate WHERE id={id};");

}

public void deleteMeeting(int id) {

write($"DELETE FROM meeting WHERE id={id};");

}

}

Презентація

Додаток А – Код програми

А1 – Database.cs

using System.Collections.Generic;

using MySql.Data.MySqlClient;

using System.Linq;

namespace seven {

public class Database {

string query;

MySqlCommand command;

MySqlDataReader reader;

MySqlConnection connection;

public const string LastCreatedID="SELECT LAST\_INSERT\_ID();";

// SYSTEM

public Database(MySqlConnection connection){ this.connection = connection; }

public MySqlDataReader read(string query){

command=new MySqlCommand(query,connection);

return command.ExecuteReader();

}

public void write(string query){

command = new MySqlCommand(query, connection);

command.ExecuteNonQuery();

}

// CREATE

public User createUser(string name, int admin = 0, int balance=0){

write($"INSERT INTO user (name,admin,balance) VALUES ('{name}',{admin},{balance});");

reader=read(LastCreatedID);

var user = new User();

while (reader.Read()){

user.ID = reader.GetInt32(0);

user.Name = name;

user.Admin = admin;

user.Balance=balance;

}

reader.Close();

return user;

}

public Estate createEstate(string title, string kind, User owner, int price){

write($"INSERT INTO estate (title,kind,owner\_id,price) VALUES ('{title}','{kind}',{owner.ID},{price});");

reader=read(LastCreatedID);

var estate = new Estate();

while (reader.Read()){

estate.ID = reader.GetInt32(0);

estate.Title = title;

estate.Kind = kind;

estate.Owner = owner;

estate.Price=price;

}

reader.Close();

return estate;

}

public Meeting createMeeting(User sender,Estate target){

write($"INSERT INTO meeting (sender\_id,target\_id) VALUES ({sender.ID},{target.ID});");

reader=read(LastCreatedID);

var meeting = new Meeting();

while (reader.Read()){

meeting.ID = reader.GetInt32(0);

meeting.Sender=sender;

meeting.Target=target;

}

reader.Close();

return meeting;

}

// READ

public List<User> getUsers(){

reader=read("SELECT id,name,admin,balance FROM user;");

var users=new List<User>();

while (reader.Read()){

var user = new User();

user.ID = reader.GetInt32(0);

user.Name = reader.GetString(1);

user.Admin = reader.GetInt32(2);

user.Balance=reader.GetInt32(3);

users.Add(user);

}

reader.Close();

return users;

}

public User getUser(int id){

return getUsers().Where(u=>u.ID==id).ToList().ElementAtOrDefault(0);

}

public List<Estate> getEstates(){

reader=read("SELECT id,owner\_id,title,kind,price FROM estate;");

var estates=new List<Estate>();

var owners=new List<int>();

while (reader.Read()){

var estate=new Estate();

estate.ID=reader.GetInt32(0);

owners.Add(reader.GetInt32(1));

estate.Title=reader.GetString(2);

estate.Kind=reader.GetString(3);

estate.Price=reader.GetInt32(4);

estates.Add(estate);

}

reader.Close();

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

estates[i].Owner=getUser(owners[i]);

}

return estates;

}

public Estate getEstate(int id){

return getEstates().Where(e=>e.ID==id).ToList().ElementAtOrDefault(0);

}

public List<Meeting> getMeetings(){

reader=read("SELECT id,sender\_id,target\_id,score,status FROM meeting;");

var meetings=new List<Meeting>();

var senders = new List<int>();

var targets = new List<int>();

while (reader.Read()){

var meeting = new Meeting();

meeting.ID = reader.GetInt32(0);

senders.Add(reader.GetInt32(1));

targets.Add(reader.GetInt32(2));

meeting.Score = reader.GetString(3);

meeting.Status = reader.GetString(4);

meetings.Add(meeting);

}

reader.Close();

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

meetings[i].Sender = getUser(senders[i]);

meetings[i].Target = getEstate(targets[i]);

}

return meetings;

}

public Meeting getMeeting(int id){

return getMeetings().Where(m=>m.ID==id).ToList().ElementAtOrDefault(0);

}

// UPDATE

public void updateUser(User user){

write($"UPDATE user SET name='{user.Name}',admin={user.Admin},balance={user.Balance} WHERE id={user.ID};");

}

public void updateEstate(Estate estate){

write($"UPDATE estate SET owner\_id={estate.Owner.ID},title='{estate.Title}',kind='{estate.Kind}',price={estate.Price} WHERE id={estate.ID};");

}

public void updateMeeting(Meeting meeting){

write($"UPDATE meeting SET sender\_id={meeting.Sender.ID},target\_id={meeting.Target.ID},score='{meeting.Score}',status='{meeting.Status}' WHERE id={meeting.ID};");

}

// DELETE

public void deleteUser(int id){

write($"DELETE FROM user WHERE id={id};");

}

public void deleteEstate(int id){

write($"DELETE FROM estate WHERE id={id};");

}

public void deleteMeeting(int id){

write($"DELETE FROM meeting WHERE id={id};");

}

}

}

A2 – Estate.cs

namespace seven {

public class Estate {

public int ID { get; set; }

public User Owner { get; set; }

public string Title { get; set; }

public string Kind { get; set; }

public int Price { get; set; } = 0;

}

}

A3 – EstateKind.cs

A4 – LoginPage.xaml

A5 – LoginPage.xaml.cs

A6 – Meeting.cs

A7 – MeetingScore.cs

A8 – MeetingStatus.cs

A9 – ProfilePage.xaml

A10 – ProfilePage.xaml.cs

A11 – User.cs

A12 – UtilityFunctions.cs

A13 – UtilityVariables.cs