**Міністерство освіти і науки України**

**Національний університет «Запорізька політехніка»**

Кафедра програмних засобів

**ЗВІТ**

Дисципліна «Фреймворки розробки програмного забезпечення»

Робота №6

Тема «Реалізаія інноваційних функцій системи»

**Виконав варіант 19**

Студент КНТ-122 Онищенко О. А.

**Прийняли**

Викладач Зелік О. В.

2024

Мета

Навчитися виділяти інноваційні функції системи, виконувати їх аналіз та реалізовувати на практиці.

Завдання

Одержати пропозиції щодо розширення інноваійної функціональності програмного забезпечення та виконати аналіз отриманих пропозицій.

Одержати вихідні коди розробленого програмного забезпечення.

Реалізцувати інноваційні функції для розробленого програмного забезпечення.

Виконати тестування розробленого програмного забезпечення. У процесі тестування має обов’язково застосовуватись модульне тестування. Тестування має виконуватися за різноманітних умов роботи програми, тобто шляхом введення різних даних, шляхом виконання на пристроях з різними апаратними характеристиками, а також під керуванням різних операційних систем або версій операційних систем.

Задача

Додати ціну до оголошень та рахунок користувача.

Виконання

1 Процес

Для додавання ціни оголошення та балансу користувача потрібно модифікувати наявні моделі даних. Наразі вони виглядають так:

Фрагмент коду 1.1 – Поточний вигляд архітектури системи

user

id int primaryKey autoIncrement

name string

admin bool

estate

id int primaryKey autoIncrement

owner\_id int foreignKey <user.id>

title string

kind string [Home | Flat | New]

meeting

id int primaryKey autoIncrement

sender\_id int foreignKey <user.id>

target\_id int foreignKey <estate.id>

status string [Wait | Done | Skip]

score string [Bad | Okay | Fine]

Модифікувати архітектуру можна так:

Фрагмент коду 1.2 – Модифікована архітектура: доданий баланс користувача та ціна оголошення

user

id int primaryKey autoIncrement

name string

admin bool

int balance

estate

id int primaryKey autoIncrement

owner\_id int foreignKey <user.id>

title string

kind string [Home | Flat | New]

int price

meeting

id int primaryKey autoIncrement

sender\_id int foreignKey <user.id>

target\_id int foreignKey <estate.id>

status string [Wait | Done | Skip]

score string [Bad | Okay | Fine]

Для реалізації введених змін можна модифікувати структуру бази даних так:

Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Автоматично згенерований опис

Рисунок 1.1 – Модифікований об’єкт користувача із доданим балансом

Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Автоматично згенерований опис

Рисунок 1.2 – Модифікований об’єкт нерухомості із доданою ціною

Подальші зміни в коді наведені у розділі «2 Код».

2 Код

Фрагмент коду 2.1 – Основний код програми, редагований з попередньої роботи

using MySql.Data.MySqlClient;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace six

{

public class User {

public int ID { get; set; }

public string Name { get; set; }

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

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

}

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;

}

public class Meeting {

public int ID { get; set; }

public User Sender { get; set; }

public Estate Target { get; set; }

public string Score { get; set; } = "Unrated";

public string Status { get; set; } = MeetingStatus.Wait;

}

public static class EstateKind {

public const string Home = "Home";

public const string Flat = "Flat";

public const string New = "New";

}

public static class MeetingStatus {

public const string Wait = "Wait";

public const string Done = "Done";

public const string Skip = "Skip";

}

public static class MeetingScore {

public const string Bad = "Bad";

public const string Okay = "Okay";

public const string Fine = "Fine";

}

public class Session {

public bool Entered { get; set; } = false;

public User Client { get; set; }

}

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};");

}

}

public class Helper {

// CHECKERS

public bool checkEstateKind(string kind) {

return kind != EstateKind.Home && kind != EstateKind.Flat && kind != EstateKind.New ? false : true;

}

public bool checkMeetingStatus(string status) {

return status != MeetingStatus.Wait && status != MeetingStatus.Done && status != MeetingStatus.Skip ? false : true;

}

public bool checkMeetingScore(string score) {

return score != MeetingScore.Bad && score != MeetingScore.Okay && score != MeetingScore.Fine ? false : true;

}

// INPUT

public string getInputString(string hint){

Console.Write($"{hint}: ");

return Console.ReadLine();

}

public int getInputNumber(string hint){

var userInput=getInputString(hint);

try {

return int.Parse(userInput);

} catch { return -1; }

}

// FORMATTERS

public string getUserStatusString(User client) {

return client.Admin==0 ? "Client" : "Manager";

}

public string getEstateKindString(User client){

return client.Admin==0 ? $"Estate kind ({EstateKind.Home} or {EstateKind.Flat})" : $"Estate kind ({EstateKind.Home} or {EstateKind.Flat} or {EstateKind.New})";

}

// DISPLAYS

// Estate

public void showEstates(List<Estate> estates,string header=""){

if (header!=""){

Console.WriteLine($"{header} estates ({estates.Count})");

}

foreach (var e in estates) {

Console.WriteLine($"{e.ID}. {e.Title} of kind {e.Kind} price {e.Price} owned by {e.Owner.Name}");

}

}

public bool showOwnedEstates(Database db,User user){

var ownedEstates=db.getEstates().Where(e=>e.Owner.ID==user.ID).ToList();

if (ownedEstates.Count<1){

Console.WriteLine("No owned estates");

return false;

} else {

showEstates(ownedEstates,"Owned");

return true;

}

}

public bool showAvailableEstates(Database db,User user){

var availableEstates=db.getEstates().Where(e=>e.Owner.ID!=user.ID).ToList();

if (availableEstates.Count<1){

Console.WriteLine("No available estates");

return false;

} else {

showEstates(availableEstates,"Available");

return true;

}

}

// Meeting

public void showMeetings(List<Meeting> meetings,string header=""){

if (header!=""){

Console.WriteLine($"{header} meetings ({meetings.Count})");

}

foreach (var m in meetings) {

Console.WriteLine($"{m.ID}. For {m.Target.Title} by {m.Sender.Name} to {m.Target.Owner.Name} rated {m.Score} status {m.Status}");

}

}

public bool showIncomingMeetings(Database db,User user){

var incomingMeetings=db.getMeetings().Where(m=>m.Target.Owner.ID==user.ID).ToList();

if (incomingMeetings.Count<1){

Console.WriteLine("No incoming meetings");

return false;

} else {

showMeetings(incomingMeetings,"Incoming");

return true;

}

}

public bool showOutgoingMeetings(Database db,User user){

var incomingMeetings=db.getMeetings().Where(m=>m.Sender.ID==user.ID).ToList();

if (incomingMeetings.Count<1){

Console.WriteLine("No outgoing meetings");

return false;

} else {

showMeetings(incomingMeetings,"Outgoing");

return true;

}

}

}

public class Program

{

static void Main(string[] args)

{

string connectionString = "uid=root;pwd=1313;host=localhost;port=3306;database=fr\_data";

var connection = new MySqlConnection(connectionString);

connection.Open();

var database = new Database(connection);

var session = new Session();

var helper = new Helper();

while (true) {

if (!session.Entered) {

var name=helper.getInputString("User name");

User foundUser = database.getUsers().Where(u=>u.Name==name).ToList().ElementAtOrDefault(0);

session.Entered = true;

if (foundUser != null) {

session.Client = foundUser;

} else {

session.Client = database.createUser(name);

}

continue;

}

int point=helper.getInputNumber("1 Edit profile\n2 Buy estate\n3 Sell estate\n4 Edit estate\n5 Remove estate\n6 Schedule meeting\n7 Rate meeting (Viewer)\n8 Process meeting (Owner)\n9 Delete meeting\n");

if (point==-1){

break;

}

// EDIT PROFILE

if (point == 1) {

string status = helper.getUserStatusString(session.Client);

Console.WriteLine($"Name: {session.Client.Name}\nStatus: {status}\nBalance: {session.Client.Balance}");

helper.showOwnedEstates(database,session.Client);

helper.showIncomingMeetings(database,session.Client);

helper.showOutgoingMeetings(database,session.Client);

point=helper.getInputNumber("1 Change name\n2 Change status\n3 Delete profile\n4 Change balance\n");

if (point==-1){

continue;

}

if (point == 1) {

var newName=helper.getInputString("New user name");

session.Client.Name = newName;

} else if (point == 2) {

if (session.Client.Admin == 1) {

session.Client.Admin = 0;

} else {

session.Client.Admin = 1;

}

} else if (point==4){

point=helper.getInputNumber("New balance");

if (point==-1){

continue;

}

session.Client.Balance=point;

}

database.updateUser(session.Client);

if (point==3){

database.deleteUser(session.Client.ID);

session.Entered=false;

break;

}

}

// BUY ESTATE

else if (point == 2) {

if (helper.showAvailableEstates(database,session.Client)==false){

continue;

}

point=helper.getInputNumber("Estate ID to buy");

if (point==-1){

continue;

}

var estate = database.getEstate(point);

if (estate==null){

Console.WriteLine("Estate not found");

continue;

} else if (session.Client.Balance<estate.Price){

Console.WriteLine("Not enough money to buy this estate");

continue;

}

estate.Owner = session.Client;

session.Client.Balance-=estate.Price;

database.updateEstate(estate);

}

// SELL ESTATE

else if (point == 3) {

var title=helper.getInputString("Title");

var kind=helper.getInputString(helper.getEstateKindString(session.Client));

var price=helper.getInputNumber("Price (default 0)");

if (price==-1){

price=0;

}

if (helper.checkEstateKind(kind) == false) {

Console.WriteLine("Wrong estate kind, please select from a list");

continue;

} else if (kind == EstateKind.New && session.Client.Admin == 0) {

Console.WriteLine($"Estate of kind {EstateKind.New} may be added only by managers");

continue;

}

database.createEstate(title, kind, session.Client, price);

}

// EDIT ESTATE

else if (point == 4) {

if (helper.showOwnedEstates(database,session.Client)==false){

continue;

}

point=helper.getInputNumber("Estate ID to edit");

if (point==-1){

continue;

}

var estate = database.getEstate(point);

if (estate==null){

Console.WriteLine("Estate not found");

continue;

}

point = helper.getInputNumber("1 Title\n2 Kind\n3 Price\n");

if (point==-1){

continue;

}

if (point == 1) {

var newTitle=helper.getInputString("New estate title");

estate.Title = newTitle;

} else if (point == 2) {

var kind=helper.getInputString(helper.getEstateKindString(session.Client));

if (helper.checkEstateKind(kind) == false) {

Console.WriteLine("Wrong estate kind, please select from a list");

continue;

}

estate.Kind = kind;

} else if (point==3){

var price=helper.getInputNumber("Price (default 0)");

if (price==-1){

price=0;

}

estate.Price=price;

}

database.updateEstate(estate);

}

// REMOVE ESTATE

else if (point == 5) {

if (helper.showOwnedEstates(database,session.Client)==false){

continue;

}

point=helper.getInputNumber("Estate ID to remove");

if (point==-1){

continue;

}

database.deleteEstate(point);

}

// SET MEETING

else if (point == 6) {

if (helper.showAvailableEstates(database,session.Client)==false){

continue;

}

point=helper.getInputNumber("Estate ID to schedule a meeting for");

if (point==-1){

continue;

}

var estate = database.getEstate(point);

if (estate==null){

Console.WriteLine("Estate not found");

continue;

}

database.createMeeting(session.Client, estate);

}

// RATE MEETING FOR VIEWER

else if (point == 7) {

if (helper.showOutgoingMeetings(database,session.Client)==false){

continue;

}

point = helper.getInputNumber("Meeting ID to rate");

if (point==-1){

continue;

}

var meeting=database.getMeeting(point);

if (meeting==null){

Console.WriteLine("Meeting not found");

continue;

}

var score=helper.getInputString($"Meeting rating ({MeetingScore.Bad} or {MeetingScore.Okay} or {MeetingScore.Fine})");

if (helper.checkMeetingScore(score) == false) {

Console.WriteLine("Incorrect score, please select from the list");

continue;

}

meeting.Score = score;

database.updateMeeting(meeting);

}

// PROCESS MEETING FOR OWNER

else if (point == 8)

{

point=helper.getInputNumber("Would you like to see incoming meetings for all estate or only for selected?\n1 All\n2 Selected\n");

if (point==-1){

continue;

}

if (point == 1) {

if (helper.showIncomingMeetings(database,session.Client)==false){

continue;

}

} else if (point==2) {

if (helper.showOwnedEstates(database,session.Client)==false){

continue;

}

point=helper.getInputNumber("Estate ID to see meetings for");

if (point==-1){

continue;

}

if (helper.showIncomingMeetings(database,session.Client)==false){

continue;

}

}

point=helper.getInputNumber("Meeting ID to change status for");

if (point==-1){

continue;

}

var meeting = database.getMeeting(point);

if (meeting==null){

Console.WriteLine("Meeting not found");

continue;

}

var status=helper.getInputString($"New meeting status ({MeetingStatus.Done} or {MeetingStatus.Skip} or {MeetingStatus.Wait})");

if (helper.checkMeetingStatus(status) == false) {

Console.WriteLine("Incorrect meeting status, please select from the list");

continue;

}

meeting.Status = status;

database.updateMeeting(meeting);

}

// DELETE MEETING

else if (point==9){

if (helper.showOutgoingMeetings(database,session.Client)==false){

continue;

}

point=helper.getInputNumber("Meeting ID to remove");

if (point==-1){

continue;

}

database.deleteMeeting(point);

}

}

connection.Close();

}

}

}

3 Результати

Тестування програми виконано за кількома сценаріями:

1. Новий користувач реєструється, поповнює баланс
2. Користувач входить, змінює баланс
3. Користувач входить, купує оголошення

Результати виконання кожного зі сценаріїв наведено нижче.

2.1 Сценарій 1 – Новий користувач поповнює баланс

Зображення, що містить текст, знімок екрана, документ, Шрифт

Автоматично згенерований опис

Рисунок 2.1 – Користувач реєструється, поповнює баланс

2.2 Сценарій 2 – Новий користувач змінює баланс

Зображення, що містить текст, знімок екрана, Шрифт, документ

Автоматично згенерований опис Рисунок 2.2 – Користувач входить, змінює баланс

2.3 Сценарій 3 – Новий користувач купує оголошення

Зображення, що містить текст, знімок екрана, Шрифт, документ

Автоматично згенерований опис

Рисунок 2.3 – Користувач входить, купує оголошення