**Міністерство освіти і науки України**

**Національний університет «Запорізька політехніка»**

Кафедра програмних засобів

**ЗВІТ**

Дисципліна «Фреймворки розробки програмного забезпечення»

Робота №4

Тема «Розроблення базових модулів програмної системи»

**Виконав варіант 19**

Студент КНТ-122 Онищенко О. А.

**Прийняли**

Викладач Зелік О. В.

2024

Мета

Навчитися аналізувати технічне завдання, предметну область, на основі проведеного аналізу й дослідження виявляти першочергові для реалізації функції програмного забезпечення та реалізовувати їх.

Завдання

Виконати аналіз ТЗ і розробленої архітектури системи та виділити першочергові функції, які має виконувати система.

Розробити програму, що реалізує першочергові функції, які має виконувати система. На даному етапі потрібно розробити консольний додаток (або кілька консольних додатків за необхідності). Реалізація програми виконується за допомогою обраних на поперендьому етапі засобів розробки. Реалізована програма має відповідати нормам програмування, тобто має бути виключена надлишковість програмного коду тощо, що має забезпечеуватися за допомогою використання принципів об’єктно-орієнтованого програмування: поліморфізм, інкапсуляція, спадкування.

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, шгляхом виконання на пристроях з різними апараними характеристиками, а також під керуванням різних операційних систем або версій операційних систем. Резульатами тестування є швидкість роботи програми, вимоги до ресурсів, а також коректність отримуваних результатів. Обов’язково мають бути створені модульні тести для перевірки результатів роботи за передбаченими ТЗ функціями прогарми.

Виконати аналіз отриманих результатів тестування. У процесі аналізу отриманих результатів має бути порівняно результати, отримані під керуванням різних операційних систем (або їх версій) та на різних пристроях.

Виконання

1 Код

Фрагмент коду 1.1 – Основний код програми

using MySql.Data.MySqlClient;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace four\_source

{

public class User

{

public int ID { get; set; }

public string Name { get; set; }

public int Admin { 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 class Meeting

{

public int ID { get; set; }

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

public string Status { get; set; } = "Wait";

public User Sender { get; set; }

public Estate Target { get; set; }

}

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 static class Query

{

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

}

public class Session

{

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

public User Client { get; set; }

public string getUserStatusString()

{

string status = "Manager";

if (this.Client.Admin == 0)

{

status = "Client";

}

return status;

}

}

public class Helper

{

public bool checkEstateKind(string kind){

if (kind!=EstateKind.Home && kind!=EstateKind.Flat && kind!=EstateKind.New) {

return false;

}

return true;

}

public bool checkMeetingStatus(string status){

if (status!=MeetingStatus.Wait && status!=MeetingStatus.Done && status!=MeetingStatus.Skip){

return false;

}

return true;

}

public bool checkMeetingScore(string score){

if (score!=MeetingScore.Bad && score!=MeetingScore.Okay && score!=MeetingScore.Fine){

return false;

}

return true;

}

}

public class Database

{

MySqlConnection connection;

string query;

MySqlCommand command;

MySqlDataReader reader;

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

public User getUserByName(string userName)

{

query = $"SELECT id,name,admin FROM user WHERE name='{userName}';";

command = new MySqlCommand(query, connection);

reader = command.ExecuteReader();

while (reader.Read())

{

var user = new User();

user.ID = reader.GetInt32(0);

user.Name = reader.GetString(1);

user.Admin = reader.GetInt32(2);

reader.Close();

return user;

}

reader.Close();

return null;

}

public User getUserById(int id)

{

query = $"SELECT id,name,admin FROM user WHERE id={id};";

command = new MySqlCommand(query, connection);

reader = command.ExecuteReader();

while (reader.Read())

{

var user = new User();

user.ID = reader.GetInt32(0);

user.Name = reader.GetString(1);

user.Admin = reader.GetInt32(2);

reader.Close();

return user;

}

reader.Close();

return null;

}

public User createUser(string userName, int admin = 0)

{

query = $"INSERT INTO user (name,admin) VALUES ('{userName}',{admin});";

command = new MySqlCommand(query, connection);

command.ExecuteNonQuery();

query = Query.LastCreatedID;

command = new MySqlCommand(query, connection);

reader = command.ExecuteReader();

var user = new User();

while (reader.Read())

{

user.ID = reader.GetInt32(0);

user.Name = userName;

user.Admin = admin;

}

reader.Close();

return user;

}

public void updateUser(User user)

{

query = $"UPDATE user SET name='{user.Name}',admin={user.Admin} WHERE id={user.ID};";

command = new MySqlCommand(query, connection);

command.ExecuteNonQuery();

}

public List<Estate> getAllEstates()

{

query=$"SELECT id,owner\_id,title,kind FROM estate;";

command=new MySqlCommand(query,connection);

reader=command.ExecuteReader();

List<Estate> estates=new List<Estate>();

List<int> 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);

estates.Add(estate);

}

reader.Close();

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

var owner = this.getUserById(owners[i]);

estates[i].Owner = owner;

}

return estates;

}

public List<Estate> getAvailableEstates(User user)

{

query=$"SELECT id,owner\_id,title,kind FROM estate WHERE owner\_id!={user.ID};";

command=new MySqlCommand(query,connection);

reader=command.ExecuteReader();

List<Estate> estates=new List<Estate>();

List<int> 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);

estates.Add(estate);

}

reader.Close();

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

var owner = this.getUserById(owners[i]);

estates[i].Owner = owner;

}

return estates;

}

public Estate getEstateById(int id)

{

query = $"SELECT id,owner\_id,title,kind FROM estate WHERE id={id};";

command = new MySqlCommand(query, connection);

reader = command.ExecuteReader();

var estate = new Estate();

int ownerId = 0;

while (reader.Read())

{

estate.ID=reader.GetInt32(0);

ownerId=reader.GetInt32(1);

estate.Title=reader.GetString(2);

estate.Kind=reader.GetString(3);

}

reader.Close();

estate.Owner=this.getUserById(ownerId);

return estate;

}

public List<Estate> getEstatesByOwnerId(int id)

{

query = $"SELECT id,title,kind FROM estate WHERE owner\_id={id};";

command = new MySqlCommand(query, connection);

reader = command.ExecuteReader();

var estates = new List<Estate>();

while (reader.Read())

{

var estate=new Estate();

estate.ID=reader.GetInt32(0);

estate.Title=reader.GetString(1);

estate.Kind=reader.GetString(2);

estates.Add(estate);

}

reader.Close();

foreach (var estate in estates){

estate.Owner=this.getUserById(id);

}

return estates;

}

public void updateEstate(Estate estate)

{

query = $"UPDATE estate SET owner\_id={estate.Owner.ID},title='{estate.Title}',kind='{estate.Kind}' WHERE id={estate.ID};";

command = new MySqlCommand(query, connection);

command.ExecuteNonQuery();

}

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

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

command = new MySqlCommand(query, connection);

command.ExecuteNonQuery();

query = Query.LastCreatedID;

command = new MySqlCommand(query, connection);

reader = command.ExecuteReader();

var estate = new Estate();

while (reader.Read())

{

estate.ID = reader.GetInt32(0);

estate.Title=title;

estate.Kind=kind;

estate.Owner=owner;

}

reader.Close();

return estate;

}

public void deleteEstate(int id)

{

query = $"DELETE FROM estate WHERE id={id};";

command = new MySqlCommand(query, connection);

command.ExecuteNonQuery();

}

public Meeting createMeeting(User sender,Estate target)

{

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

command = new MySqlCommand(query, connection);

command.ExecuteNonQuery();

query = Query.LastCreatedID;

command = new MySqlCommand(query, connection);

reader = command.ExecuteReader();

var meeting = new Meeting();

while (reader.Read())

{

meeting.ID = reader.GetInt32(0);

meeting.Score=null;

meeting.Status=MeetingStatus.Wait;

meeting.Sender=sender;

meeting.Target=target;

}

reader.Close();

return meeting;

}

public List<Meeting> getAllMeetings(){

query=$"SELECT id,sender\_id,target\_id,score,status FROM meeting;";

command=new MySqlCommand(query,connection);

reader=command.ExecuteReader();

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++){

var sender=this.getUserById(senders[i]);

var target=this.getEstateById(targets[i]);

meetings[i].Sender=sender;

meetings[i].Target=target;

}

return meetings;

}

public List<Meeting> getOutgoingMeetings(User sender){

// get all meetings where sender is sender user

var allMeetings=this.getAllMeetings();

var filteredMeetings=new List<Meeting>();

foreach (var meeting in allMeetings){

if (meeting.Sender.ID==sender.ID){

filteredMeetings.Add(meeting);

}

}

return filteredMeetings;

}

public List<Meeting> getIncomingMeetings(User owner){

// get all meetings where target estate owner is owner user

var allMeetings=this.getAllMeetings();

var filteredMeetings=new List<Meeting>();

foreach (var meeting in allMeetings){

if (meeting.Target.Owner.ID==owner.ID){

filteredMeetings.Add(meeting);

}

}

return filteredMeetings;

}

public void updateMeeting(int id, string score, string status, User sender, Estate target){

query = $"UPDATE meeting SET score='{score}',status='{status}',sender\_id={sender.ID},target\_id={target.ID} WHERE id={id};";

command = new MySqlCommand(query, connection);

command.ExecuteNonQuery();

}

public Meeting getMeetingById(int id)

{

query = $"SELECT id,score,status,sender\_id,target\_id FROM meeting WHERE id={id};";

command = new MySqlCommand(query, connection);

reader = command.ExecuteReader();

var meeting = new Meeting();

int senderId = 0;

int targetId = 0;

while (reader.Read())

{

meeting.ID=reader.GetInt32(0);

meeting.Score=reader.GetString(1);

meeting.Status=reader.GetString(2);

senderId=reader.GetInt32(3);

targetId=reader.GetInt32(4);

}

reader.Close();

meeting.Sender=this.getUserById(senderId);

meeting.Target=this.getEstateById(targetId);

return meeting;

}

public List<Meeting> getMeetingsByTargetId(int id)

{

query = $"SELECT id,score,status,sender\_id,target\_id FROM meeting WHERE target\_id={id};";

command = new MySqlCommand(query, connection);

reader = command.ExecuteReader();

var meetings = new List<Meeting>();

var senders=new List<int>();

while (reader.Read())

{

var meeting=new Meeting();

meeting.ID=reader.GetInt32(0);

meeting.Score=reader.GetString(1);

meeting.Status=reader.GetString(2);

senders.Add(reader.GetInt32(3));

}

reader.Close();

var target=this.getEstateById(id);

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

meetings[i].Target=target;

meetings[i].Sender=this.getUserById(senders[i]);

}

return meetings;

}

}

public class Program

{

static void Main(string[] args)

{

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

var connection = new MySqlConnection(connectionString);

var database = new Database(connection);

var helper=new Helper();

connection.Open();

var session = new Session();

while (true)

{

string choice;

int point;

if (!session.Entered)

{

Console.WriteLine("User name:");

var userName = Console.ReadLine();

User foundUser = database.getUserByName(userName);

if (foundUser!=null)

{

session.Client = foundUser;

}

else

{

session.Client = database.createUser(userName);

}

session.Entered=true;

}

else

{

const string menu = @"1 Edit profile

2 Buy estate

3 Sell estate

4 Edit estate

5 Remove estate

6 Schedule meeting

7 Rate meeting (Viewer)

8 Process meeting (Owner)";

Console.WriteLine(menu);

choice = Console.ReadLine();

try

{

point = int.Parse(choice);

}

catch (Exception e)

{

break;

}

Console.WriteLine();

if (point == 1)

{

// EDIT PROFILE

// read use details

// get input from user on what to change

// Request manager status

// update user object

string userStatus = session.getUserStatusString();

Console.WriteLine($"User name: {session.Client.Name}\nUser status: {userStatus}");

// show all owned property

var estates = database.getEstatesByOwnerId(session.Client.ID);

if (estates.Count > 0)

{

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

foreach (var estate in estates)

{

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

}

}

// show all incoming meetings

var incomingMeetings = database.getIncomingMeetings(session.Client);

if (incomingMeetings.Count > 0)

{

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

foreach (var meeting in incomingMeetings)

{

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

}

}

// show all outgoing meetings

var outgoingMeetings = database.getOutgoingMeetings(session.Client);

if (outgoingMeetings.Count > 0)

{

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

foreach (var meeting in outgoingMeetings)

{

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

}

}

string options = "1 Change name\n2 Change status";

Console.WriteLine(options);

choice = Console.ReadLine();

try

{

point = int.Parse(choice);

}

catch (Exception e)

{

continue;

}

Console.WriteLine();

if (point == 1)

{

Console.WriteLine("Enter new user name:");

var newName = Console.ReadLine();

session.Client.Name = newName;

database.updateUser(session.Client);

}

else if (point == 2)

{

if (session.Client.Admin == 1)

{

session.Client.Admin = 0;

}

else

{

session.Client.Admin = 1;

}

database.updateUser(session.Client);

}

}

else if (point == 2)

{

// BUY ESTATE

// read all available estate

// show all estate where owner is not user

// get input from user on what to buy

var estates=database.getAvailableEstates(session.Client);

if (estates.Count < 1)

{

Console.WriteLine("No estates available");

continue;

}

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

foreach(var estate in estates)

{

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

}

Console.WriteLine("Enter estate ID to buy:");

choice = Console.ReadLine();

int estateId;

try

{

estateId = int.Parse(choice);

}

catch (Exception e)

{

continue;

}

// buy estate here

var selectedEstate = database.getEstateById(estateId);

selectedEstate.Owner = session.Client;

database.updateEstate(selectedEstate);

}

else if (point == 3)

{

// SELL ESTATE

// get all the estate details

// add new estate to database

Console.WriteLine("Estate title:");

var title=Console.ReadLine();

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

Console.WriteLine($"Estate kind ({EstateKind.Home} or {EstateKind.Flat} or {EstateKind.New})");

}

else{

Console.WriteLine($"Estate kind ({EstateKind.Home} or {EstateKind.Flat})");

}

var kind=Console.ReadLine();

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;

}

var estate = database.createEstate(title, kind, session.Client);

}

else if (point == 4)

{

// EDIT ESTATE

// get all estate by owner

// get estate id from user

// get option to edit in estate

// update estate in a database

var estates = database.getEstatesByOwnerId(session.Client.ID);

foreach (var e in estates)

{

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

}

Console.WriteLine("Select estate ID to edit:");

var estateIdString = Console.ReadLine();

int estateId;

try{

estateId=int.Parse(estateIdString);

} catch (Exception e){

continue;

}

var estate=database.getEstateById(estateId);

Console.WriteLine("1 Title\n2 Kind");

choice=Console.ReadLine();

try{

point=int.Parse(choice);

} catch (Exception e){

continue;

}

if (point==1){

Console.Write("Enter new estate title please: ");

var newTitle=Console.ReadLine();

estate.Title=newTitle;

}

else if (point==2){

if (session.Client.Admin==0){

Console.Write($"Enter new kind please ({EstateKind.Home} or {EstateKind.Flat} or {EstateKind.New}): ");

} else {

Console.Write($"Enter new kind please ({EstateKind.Home} or {EstateKind.Flat}): ");

}

var newKind = Console.ReadLine();

estate.Kind = newKind;

}

database.updateEstate(estate);

}

else if (point == 5)

{

// REMOVE ESTATE

// show all owned estates

// get estate id to delete

var estates = database.getEstatesByOwnerId(session.Client.ID);

foreach (var e in estates)

{

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

}

Console.WriteLine("Select estate ID to delete:");

var estateIdString = Console.ReadLine();

int estateId;

try

{

estateId = int.Parse(estateIdString);

}

catch (Exception e)

{

continue;

}

database.deleteEstate(estateId);

}

else if (point == 6)

{

// SET MEETING

// show all estates that user can buy

// get estate id from user

// schedule new meeting for that estate

var estates = database.getAvailableEstates(session.Client);

if (estates.Count < 1)

{

Console.WriteLine("No estates available");

continue;

}

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

foreach (var e in estates)

{

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

}

Console.Write("Enter estate ID to schedule meeting for: ");

choice = Console.ReadLine();

int estateId;

try{

estateId = int.Parse(choice);

} catch (Exception e) {

continue;

}

var estate=database.getEstateById(estateId);

database.createMeeting(session.Client,estate);

}

else if (point == 7)

{

// RATE MEETING FOR VIEWER

// get all meetings where sender\_id==client and status is Done

// get meeting rating from user

// update meeting with rating

var foundMeetings = database.getOutgoingMeetings(session.Client);

var meetings = new List<Meeting>();

foreach (var m in foundMeetings)

{

if (m.Status == MeetingStatus.Done)

{

meetings.Add(m);

}

}

if (meetings.Count < 1)

{

Console.WriteLine("No meetings to rate");

continue;

}

foreach (var m in meetings)

{

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

}

Console.Write("Enter meeting ID to rate please: ");

var response = Console.ReadLine();

int meetingId;

try

{

meetingId = int.Parse(response);

} catch (Exception e)

{

continue;

}

Console.Write($"Enter meeting rating ({MeetingScore.Bad} or {MeetingScore.Okay} or {MeetingScore.Fine}): ");

var meetingScore = Console.ReadLine();

if (helper.checkMeetingScore(meetingScore) == false)

{

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

continue;

}

var meeting = database.getMeetingById(meetingId);

meeting.Score = meetingScore;

database.updateMeeting(meeting.ID,meeting.Score,meeting.Status,meeting.Sender,meeting.Target);

}

else if (point==8){

// PROCESS MEETING FOR OWNER

// see for all estate or only for selected

Console.WriteLine("Would you like to see incoming meetings for all estate or only for selected?");

Console.WriteLine("1 All\n2 Selected");

choice = Console.ReadLine();

try

{

point = int.Parse(choice);

} catch (Exception e)

{

continue;

}

Console.WriteLine();

if (point == 1)

{

var incomingMeetings = database.getIncomingMeetings(session.Client);

if (incomingMeetings.Count > 0)

{

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

foreach (var m in incomingMeetings)

{

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

}

} else

{

Console.WriteLine("No meetings found");

}

} else

{

var estates = database.getEstatesByOwnerId(session.Client.ID);

foreach (var e in estates)

{

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

}

Console.Write("Please enter estate ID to see meetings for: ");

var estateIdString = Console.ReadLine();

int estateId;

try

{

estateId = int.Parse(estateIdString);

} catch (Exception e)

{

continue;

}

var incomingMeetings = database.getMeetingsByTargetId(estateId);

var estate = database.getEstateById(estateId);

if (incomingMeetings.Count > 0)

{

Console.WriteLine($"Incoming meetings for {estate.Title} ({incomingMeetings.Count})");

foreach (var m in incomingMeetings)

{

Console.WriteLine($"{m.ID}. By {m.Sender.Name} rated {m.Score} status {m.Status}");

}

}

else

{

Console.WriteLine("No meetings found");

}

}

Console.Write("Enter meeting ID to change status for please: ");

choice = Console.ReadLine();

int meetingId;

try

{

meetingId = int.Parse(choice);

} catch (Exception e)

{

continue;

}

var meeting = database.getMeetingById(meetingId);

Console.Write($"Please enter new meeting status ({MeetingStatus.Done} or {MeetingStatus.Skip} or {MeetingStatus.Wait}): ");

var status= Console.ReadLine();

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

{

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

continue;

}

meeting.Status = status;

database.updateMeeting(meeting.ID, meeting.Score, meeting.Status, meeting.Sender, meeting.Target);

}

else

{

break;

}

}

Console.WriteLine();

}

connection.Close();

}

}

}

2 Результати

Тестування програми виконано за кількома сценаріями:

1. Створення нового користувача, додавання нового оголошення, зміна назви оголошення.
2. Використання існуючого користувача, придбання новоствореного оголошення
3. Використання існуючого користувача, зміна деталей придбаного оголошення.
4. Використання існуючого користувача, видалення оголошення.
5. Використання новоствореного користувача, перегляд доступних оголошень, призначення зустрічі на оголошення.
6. Використання користувача-власника оголошення, опрацювання зустрічі: зміна статусу на «Переглянуто».
7. Використання новоствореного користувача, оцінка зустрічі.

Деталі виконання кожного зі сценаріїв наведено нижче.

2.1 Сценарій 1 – Новий користувач створює та змінює оголошення

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

Автоматично згенерований описРисунок 2.1 – Вхід користувача, створення оголошення

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

Автоматично згенерований опис

Рисунок 2.2 – Зміна даних оголошення, перегляд результатів

2.2 Сценарій 2 – Існуючий користувач купує нове оголошення

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

Автоматично згенерований опис

Рисунок 2.3 – Вхід існуючого користувача, купівля нового оголошення та перегляд його у профілі

2.3 Сценарій 3 – Існуючий користувач змінює деталі оголошення

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

Автоматично згенерований опис

Рисунок 2.4 – Вхід користувача, зміна деталей оголошення, перегляд його у профілі

2.4 Сценарій 4 – Існуючий користувач видаляє оголошення

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

Автоматично згенерований опис

Рисунок 2.5 – Вхід користувача, зміна деталей оголошення, перегляд наявних у профілі

2.5 Сценарій 5 – Новий користувач призначає зустріч на оголошення

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

Автоматично згенерований опис

Рисунок 2.6 – Вхід користувача, призначення зустрічі, перегляд у профілі

2.6 Сценарій 6 – Користувач-власник змінює статус зустрічі

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

Автоматично згенерований опис

Рисунок 2.7 – Вхід користувача, зміна статусу зустрічі, перегляд у профілі

2.7 Сценарій 7 – Новий користувач змінює оцінку зустрічі

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

Автоматично згенерований опис

Рисунок 2.8 – Вхід користувача, зміна оцінки зустрічі, перегляд у профілі

3 Аналіз результатів

По виконанню програми благодаттю Господа нашого Ісуса Христа було виявлено що застосунок працює коректно і всі функції виконуються успішно: Алилуя!