Meetup Command Line Application

Bogazici University 2022

M. Semih Celek, 2018100075

2022-01-08

Contents

Introduction

System Request:

Business Needs: There is a need for new socials media application that focuses on creating meetups and creating events. This application aims to fulfill this need.

Business Values: The new social media application database system will enable users to create meetups and create posts. Also it will have business values throughout advertisement.

Objective

Aim of the project is to create a new social media application enabling people to organize events. This events can be at both for online, or physical. Also application will enable its users to communicate on meetup basis so that people can stay in touch.

Application Features

Meetup Module

Users can create meetup instances and invite other people to their meetups. Meetups are small group of users who shares a common feed, communication space.

Business rules are listed as;

- Only Users can create Meetups.
- Meetups have their admins.
- Meetup's feed can only be seen by its members, not member User cannot see the inside of a Meetup.
- Users can attend a meetup by its link and can quit anytime.
- Only admins have privilege to manage users(add, subtract)

Post Module

Users can create their posts like any other platform Posts are the simple way of user the share their ideas. Post consist of *Title, Content, Date, User Reference and ifExist a Meetup Refernce, Comments Ref*.

Business rules are listed as;

- Only Users can create posts.
- Posts can be posted on both meetup events or publicly.
- Posts that are published on meetup instances can only be viewed by those who attends to the meetup.
- Posts can only be edited by it's user.
- Only User's friends or meetup associates can comment the post, others can't comment.

Repo Link: https://github.com/semihcelek/MeetupApplication

Searching Module

Searching Module can be used to search both Users, Meetups and Posts. Users of search module can search every instance of the meetup and user by entering its id.

Application Interface

Meetup Application is built on command line interface. It offers limited amount of interaction by its nature. In this application We have 1 main menu where we can the entity(User, Meetup, Post) which we want to manipulate and we have 1 sub menu for each entity. We have 5 main operations on each entity. These operations are fundamental create, read, read-all, update and delete operations.

Screen Shots of Application

Main Menu:

```
Welcome to the meetup application, Please select an action
|--------------------------------------------------------|
Press (U) For All User Operations.
Press (M) For All Meetup Operations.
Press (P) For All Post Operations.
Press (Q) For Exit.


▯
```

User Sub-Menu:

```
uUser Operations are listed as:


|--------------------------------|
Press (A) for finding all users.
Press (S) for searching an users with Id.
Press (C) for creating an new user.
Press (U) for updating an user
Press (D) for deleting an users with Id.
```

Meetup Command Line Application

Application

Command line application is written with C# multi-purposed, typed, programming language. Application persist its data with sql database. In this project Mysql database is implemented although other databases can be easily integrated by implementing DataAccess interfaces.

Application consist of 3 major parts, Models, Services, and View.

- Model is where we define data schemas for objects. In the project we have User Model, Meetup Model, and Post Model.

- Services is where we manipulate and iterate the data. It's also where we persist the data by using database, the project uses Mysql database.

- View is where we present data. In the project, command line interface is used.

In the project we have users that interacts with events.

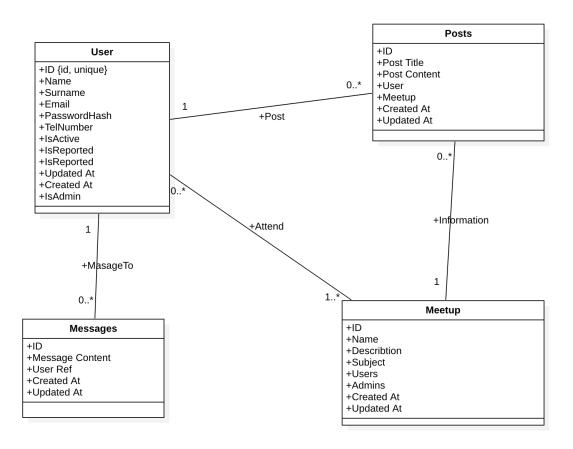This interaction can be represented with an Uml diagram.

Figure 1: Uml Diagram

Creating Database

Lets start with creating database;

```
1  create database meetupdb;
2
3  use meetupdb;
```

After that create user table;

```
1   create table users
2   (
3       id           int unsigned primary key AUTO_INCREMENT,
4       name         char(80)         not null,
5       surname      char(80)         null,
6       email        char(120)        not null unique,
7       passwordHash char(255)        not null,
8       isActive     bool default true  not null,
9       isAdmin      bool default false not null
10  );
11
12  create table meetups
13  (
14      id           int unsigned primary key AUTO_INCREMENT,
15      name         char(80)  not null,
16      description  char(255) null,
17      subject      char(120) not null unique,
18      createdAt    timestamp not null,
19      updatedAt    timestamp null
20  );
21
22  create table meetupMembership
23  (
24      user    int unsigned not null,
25      meetup  int unsigned not null,
26      primary key (user, meetup),
27      constraint meetupMembership_users_fk foreign key users_fk (user)
            references users (id),
28      constraint meetupMembership_meetups_fk foreign key meetups_fk (
            meetup) references meetups (id)
29  );
30
31  create table posts
32  (
33      id          int unsigned primary key auto_increment,
34      title       char(60)     not null unique,
35      content     text         not null,
36      author      int unsigned not null,
37      meetupPost  int unsigned null,
38      createdAt   timestamp    not null,
39      updatedAt   timestamp    null,
```

```
40        constraint author_fk foreign key users_fk (author) references users
            (id),
41        constraint meetupPost_fk foreign key meetup_fk (meetupPost)
            references meetups (id)
42
43  );
```

Then we create other tables(Meetups, Posts, etc...)

Creating Data Model Class

Let's dive in to the project. First we create the UserModel class to represent the user data model;

UserModel.cs

```
 1  private int _id; // we define private variables.
 2
 3  public int Id     // then define public getters and
 4                    //      setters for each field.
 5  {
 6      get => _id;
 7      set => _id = value;
 8  }
 9
10  private string _name;
11
12  public string Name
13  {
14      get => _name;
15      set => _name = value;
16  }
17  ...
```

Then we define constructor for the class for instantiating user object.

```
 1  public UserModel(int id, string name, string surname, string email,
        string passwordHash, string telNumber)
 2  {
 3      _id = id;
 4      _name = name;
 5      _surname = surname;
 6      _email = email;
 7      _passwordHash = passwordHash;
 8      _telNumber = telNumber;
 9  }
```

Creating Services

After creating model class for each data models, we continue with creating services. Services are required for interacting with data. But before dive into the Service we should take quick glance at the IUserDbAccess interface.

We are going to create an interface for user service in order extend the modularity of our application.

UserService going to use an interface for interacting with data access layer. In the interface we create functions for identifying the extend of database actions.

```
1  public interface IUserDbAccess
2  {
3      List<UserModel> FindAll();
4
5      UserModel FindOne(int id);
6      void Create(UserModel user);
7      void Update(UserModel user);
8      void Delete(int id);
9  }
```

Then we create UserService class with using IUserDbAccess interface;

```
1   public class UserService
2   {
3       private IUserDbAccess _dbInstance; // we define private instance of
4           dbInstance.
5       public UserService(IUserDbAccess dbInstance)
6       {
7           _dbInstance = dbInstance; // then we invert the dependency
8                                     // by passing it at the
                                         constructor.
9       }
10
11      public void GetAllUsers()
12      {
13          _dbInstance.FindAll();   // take attention we use the function
14                                   // descriptions from the
                                        interface.
15      }
16
17      public void FindOne(int id)
18      {
19          _dbInstance.FindOne(id);
20      }
21
22      public void CreateUser(string name, string surname, string email,
            string password, string telNumber)
23      {
```

```
24          var user = new UserModel(name, surname, email, password,
                telNumber);
25          _dbInstance.Create(user);
26      }
27
28      public void UpdateUser(string name, string surname, string email,
            string password, string telNumber)
29      {
30          var user = new UserModel(name, surname, email, password,
                telNumber);
31          _dbInstance.Update(user);
32      }
33
34      public void DeleteUser(int userId)
35      {
36          _dbInstance.Delete(userId);
37      }
38  }
```

We have created the UserService, let's implement the Mysql Database.

Implement Mysql Database

In first chapter we created our databases, In this chapter we are going to implement mysql database to the application.

First we need to add Mysql Connector package to the application

```
1       dotnet add package MySql.Data --version 8.0.27
```

After installing the package lets create MysqlDatabase class.

MysqlDatabe.cs

```csharp
public class MysqlDatabase
    {
        public readonly MySqlConnection MySqlConnection;
        private string connectionString =
            "server=127.0.0.1; database=meetupdb; uid=root; pwd=;";

        public MysqlDatabase()
        {
            try
            {
                MySqlConnection = new MySqlConnection(connectionString)
                    ;
                MySqlConnection.Open();
                Console.WriteLine("Trying to connect...");
            }
            catch (MySqlException e)
            {
                Console.WriteLine(e);
                throw;
            }
        }

        public void Dispose()
        {
            MySqlConnection.Close();
        }
    }
```

Now we are ready for integrating database access layer for UserService by implementing IUserDbAccess interface.

MysqlUserAccess.cs

```csharp
public class MysqlUserAccess : IUserDbAccess
    {
        private MySqlConnection _connection;

```

```
 5          public MysqlUserAccess(MySqlConnection dbInstance)
 6          {
 7              _connection = dbInstance;
 8          }
 9
10          public List<UserModel> FindAll()
11          {
12              List<UserModel> allUsers = new List<UserModel>();
13              const string findAllUsersSql = "select * from users;";
14              MySqlCommand command = new MySqlCommand(findAllUsersSql,
                     _connection);
15              MySqlDataReader reader = command.ExecuteReader();
16
17              Console.WriteLine("All Users are listed as:");
18              while (reader.Read())
19              {
20                  var user = new UserModel(Convert.ToInt32(reader[0]),
                         reader[1].ToString(), reader[2].ToString(),
21                          reader[3].ToString(), reader[4].ToString(), reader
                             [5].ToString());
22                  allUsers.Add(user);
23              }
24
25              foreach (var user in allUsers)
26              {
27                  Console.WriteLine($"{user.Name}, {user.Surname} has id
                         of {user.Id}");
28              }
29
30              reader.Close();
31              return allUsers;
32          }
33
34          public UserModel FindOne(int id)
35          {
36              UserModel user = null;
37              const string findUserByIdSql = "select * from users where
                     id =@id";
38              MySqlCommand command = new MySqlCommand(findUserByIdSql,
                     _connection);
39              command.Parameters.AddWithValue("@id", id);
40              MySqlDataReader reader = command.ExecuteReader();
41
42              Console.WriteLine("Single User is listed as:");
43              while (reader.Read())
44              {
45                  user = new UserModel(Convert.ToInt32(reader[0]), reader
                         [1].ToString(), reader[2].ToString(),
46                          reader[3].ToString(), reader[4].ToString(), reader
                             [5].ToString());
47                  Console.WriteLine(user.Id + "- " + user.Name + " " +
```

```
47              user.Surname + " " + user.TelNumber);
48          }
49
50
51          reader.Close();
52          return user;
53      }
54
55      public void Create(UserModel user)
56      {
57          const string insertUserSql =
58              "insert into users(name, surname, email, passwordHash,
                    isActive, isAdmin) values (@name, @surname, @email,
                    @passwordHash, true, false);";
59          try
60          {
61              MySqlCommand command = new MySqlCommand(insertUserSql,
                    _connection);
62
63              command.Parameters.AddWithValue("@name", user.Name);
64              command.Parameters.AddWithValue("@surname", user.
                    Surname);
65              command.Parameters.AddWithValue("@email", user.Email);
66              command.Parameters.AddWithValue("@passwordHash", user.
                    PasswordHash);
67
68              command.ExecuteNonQuery();
69          }
70          catch (Exception e)
71          {
72              Console.WriteLine(e);
73              throw;
74          }
75      }
76
77      public void Update(UserModel user)
78      {
79          const string updateUserSql =
80              "update users set name = @name, surname = @surname,
                    email = @email, passwordHash = @passwordHash where
                    id=@userId;";
81          try
82          {
83              MySqlCommand command = new MySqlCommand(updateUserSql,
                    _connection);
84
85              command.Parameters.AddWithValue("@name", user.Name);
86              command.Parameters.AddWithValue("@surname", user.
                    Surname);
87              command.Parameters.AddWithValue("@email", user.Email);
88              command.Parameters.AddWithValue("@passwordHash", user.
```

```
                            PasswordHash);
 89             command.Parameters.AddWithValue("@userId", user.Id);
 90
 91             command.ExecuteNonQuery();
 92         }
 93         catch (Exception e)
 94         {
 95             Console.WriteLine(e);
 96             throw;
 97         }
 98     }
 99
100     public void Delete(int id)
101     {
102         const string deleteUserSql = "delete from users where id =
                @userid";
103
104         try
105         {
106             MySqlCommand command = new MySqlCommand(deleteUserSql,
                    _connection);
107             command.Parameters.AddWithValue("@userid", id);
108
109             command.ExecuteNonQuery();
110         }
111         catch (Exception e)
112         {
113             Console.WriteLine(e);
114             throw;
115         }
116     }
117
118     public void AttendMeetup(MeetupModel meetup)
119     {
120         throw new NotImplementedException();
121     }
122
123     public void LeaveMeetup(MeetupModel meetup)
124     {
125         throw new NotImplementedException();
126     }
127 }
```

Then we implement all the method that indicated on the interface.

Creating View Controller for CLI

So far we are created Model and Service for our application, but we still haven't use the services we
actually created. In this part we are going to create Cli controller for the application.

Let's create A controller class for user, UserOperationsController class.

UserOperationsController.cs

```
 1  public class UserOperationsController
 2      {
 3          private UserService _userService; // we create _userService
                property.
 4
 5          public UserOperationsController(UserService userService)
 6          {
 7              _userService = userService; // then we pass it at the
                    constructor.
 8          }
 9
10          public void UserOperations()
11          {
12              Console.WriteLine("User Operations are listed as:\n");
13              Console.WriteLine("|------------------------------------|")
                    ;
14              Console.WriteLine(
15                  "Press (A) for finding all users.\n" +
16                  "Press (S) for searching an users with Id.\n" +
17                  "Press (C) for creating an new user.\n" +
18                  "Press (U) for updating an user \n" +
19                  "Press (D) for deleting an users with Id.\n"
20              );
21
22              var argumentSelection = Console.ReadKey();
23
24              switch (argumentSelection.Key)
25              {
26                  case ConsoleKey.A:
27                      _userService.GetAllUsers();
28                      break;
29
30                  case ConsoleKey.S:
31                      FindUser();
32                      break;
33
34                  case ConsoleKey.C:
35                      RegisterUser();
36                      break;
37
38                  case ConsoleKey.U:
39                      UpdateUser();
40                      break;
41
42                  case ConsoleKey.D:
43                      DeleteUser();
44                      break;
```

```
45
46                  case ConsoleKey.Backspace:
47                      return;
48              }
49          }
```

After that we create class for controlling the main menu;

CommanlineController.cs

```
 1   public class CommandLineController
 2       {
 3           private readonly UserOperationsController
                 _userOperationsController;
 4           private readonly MeetupOperationsController
                 _meetupOperationsController;
 5           private readonly PostOperationsController
                 _postOperationsController;
 6
 7           public CommandLineController(MeetupService meetupService,
 8               UserService userService, PostService postService)
 9           {
10               _meetupOperationsController = new
                     MeetupOperationsController(meetupService);
11               _userOperationsController = new UserOperationsController(
                     userService);
12               _postOperationsController = new PostOperationsController(
                     postService);
13           }
14
15
16           public void InitializeCommandLine()
17           {
18               Console.WriteLine("Welcome to the meetup application,
                     Please select an action");
19               Console.WriteLine("
                     |-------------------------------------------------------|
                     ");
20
21               Console.WriteLine(
22                   "Press (U) For All User Operations.\n" +
23                   "Press (M) For All Meetup Operations.\n" +
24                   "Press (P) For All Post Operations.\n" +
25                   "Press (Q) For Exit.\n"
26               );
27
28               var argumentSelection = Console.ReadKey();
29
30               while (argumentSelection.Key != ConsoleKey.Q)
31               {
32                   switch (argumentSelection.Key)
```

```
33                    {
34                        case ConsoleKey.U:
35                            _userOperationsController.UserOperations();
36                            break;
37                        case ConsoleKey.M:
38                            _meetupOperationsController.MeetupOperations();
39                            break;
40                        case ConsoleKey.P:
41                            _postOperationsController.PostOperations();
42                            break;
43                    }
44                }
45            }
46        }
```

Building The Application

In the Main program, we pass all the dependencies for classes, then we initialize with command line method.

Program.cs

```
1  internal static class Program
2      {
3          private static void Main(string[] args)
4          {
5              var sqlConnection = new MysqlDatabase();
6
7              var userService = new UserService(new MysqlUserAccess(
                     sqlConnection.MySqlConnection));
8              var meetupService = new MeetupService(new MysqlMeetupAccess
                     (sqlConnection.MySqlConnection));
9              var postService = new PostService(new MysqlPostAccess(
                     sqlConnection.MySqlConnection));
10
11             var commandLine = new CommandLineController(meetupService,
                     userService, postService);
12             commandLine.InitializeCommandLine();
13         }
14      }
```

Example Output from Program

```
1  Trying to connect...
2  Welcome to the meetup application, Please select an action
3  |-------------------------------------------------------|
4  Press (U) For All User Operations.
```

```
 5  Press (M) For All Meetup Operations.
 6  Press (P) For All Post Operations.
 7  Press (Q) For Exit.
 8
 9  // u is pressed
10  User Operations are listed as:
11
12  |--------------------------------|
13  Press (A) for finding all users.
14  Press (S) for searching an users with Id.
15  Press (C) for creating an new user.
16  Press (U) for updating an user
17  Press (D) for deleting an users with Id.
18
19  //a is pressed
20
21  All Users are listed as:
22  Enes, Sucuk has id of 1
23  Ege, Eroglu has id of 2
24  deniz, doygun has id of 4
25  hamdi, cakici has id of 5
26
27  User Operations are listed as:
28
29  |--------------------------------|
30  Press (A) for finding all users.
31  Press (S) for searching an users with Id.
32  Press (C) for creating an new user.
33  Press (U) for updating an user
34  Press (D) for deleting an users with Id.
35
36  // c is pressed
37  In order to create an user please enter a name
38  Salih
39  please enter a surname
40  Salkim
41  please enter a email
42  salih@mail.com
43  please enter a password
44  passwordsuper
45  please enter a telNumber
46  53420005911
47
48  User Operations are listed as:
49
50  |--------------------------------|
51  Press (A) for finding all users.
52  Press (S) for searching an users with Id.
53  Press (C) for creating an new user.
54  Press (U) for updating an user
55  Press (D) for deleting an users with Id.
```

```
56
57  // D is pressed
58
59  Caution!, you're deleting an user this can't be recovered.
60  Please enter an user id to delete
61  5
62  // deletes user with id of 5.
```

Conclusion

With this .Net, C# Application, we have successfully implemented core concepts of the Meetup Application and we persist the data with solid database implementation. In this application we have created command line interface to utilize operations, in the future, we can replace the command line interface with more concrete frontend application make application more accessible and use this project as rest backend api.