How to develop a chat system? The SoulMate Chat project

Version2

Overview

Ok. In the first version, we developed a quiz-based system to allow "matching souls" to chat with each other. We used a Signal-R-based web chat system for this.

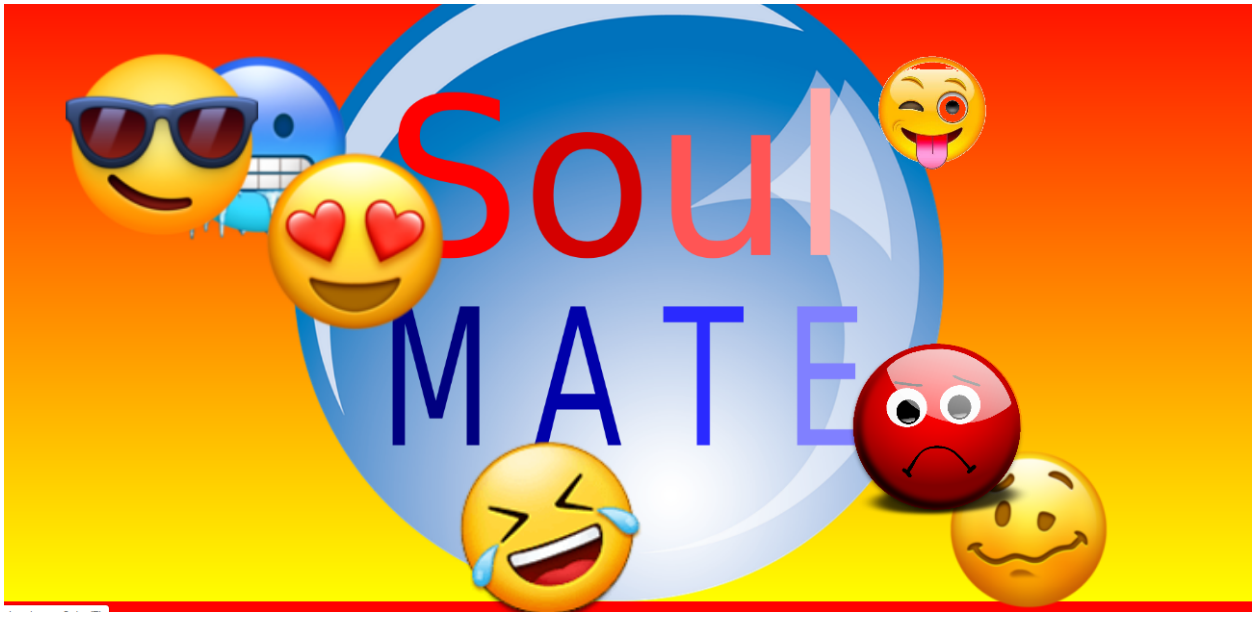
Now we will develop the same chat but very differently. First, we will not use the signal-R chat but we will develop our chat server instead. So here, we develop everything from scratch.

What we will try to achieve is a system where users can register and find a unique ‘matching soul’ by trying to find the unique combination of Emojis that will reveal a single user.

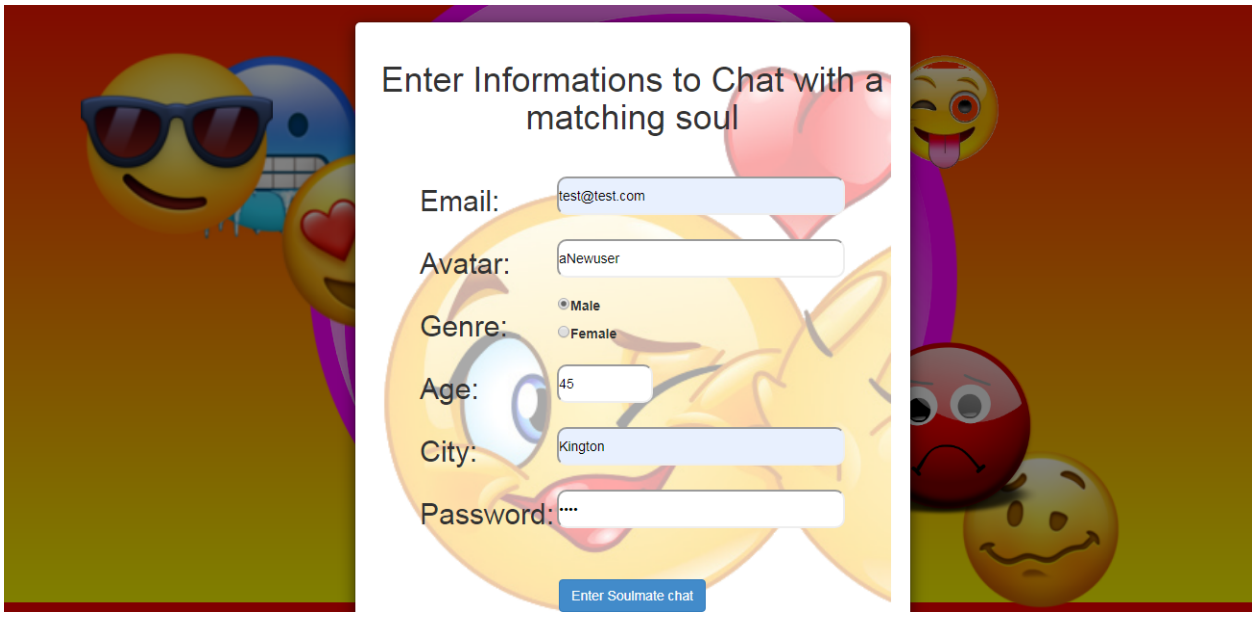
At start a user will register and will choose a unique combination of Emojis symbols. This will act as a 'code" to give him/her a very unique signature. The chances that someone else chooses that precise combination of symbols is almost zero.

Then the user can find its 'unique' matching soul by forming combinations of emojis. When the number of chat users having a "common intersection" with that sequence of symbols is equal to 1, the user can start chatting privately with the "soulmate". Of course, before that happens, an invitation must be sent by the user… and accepted by the other party.

User Interface

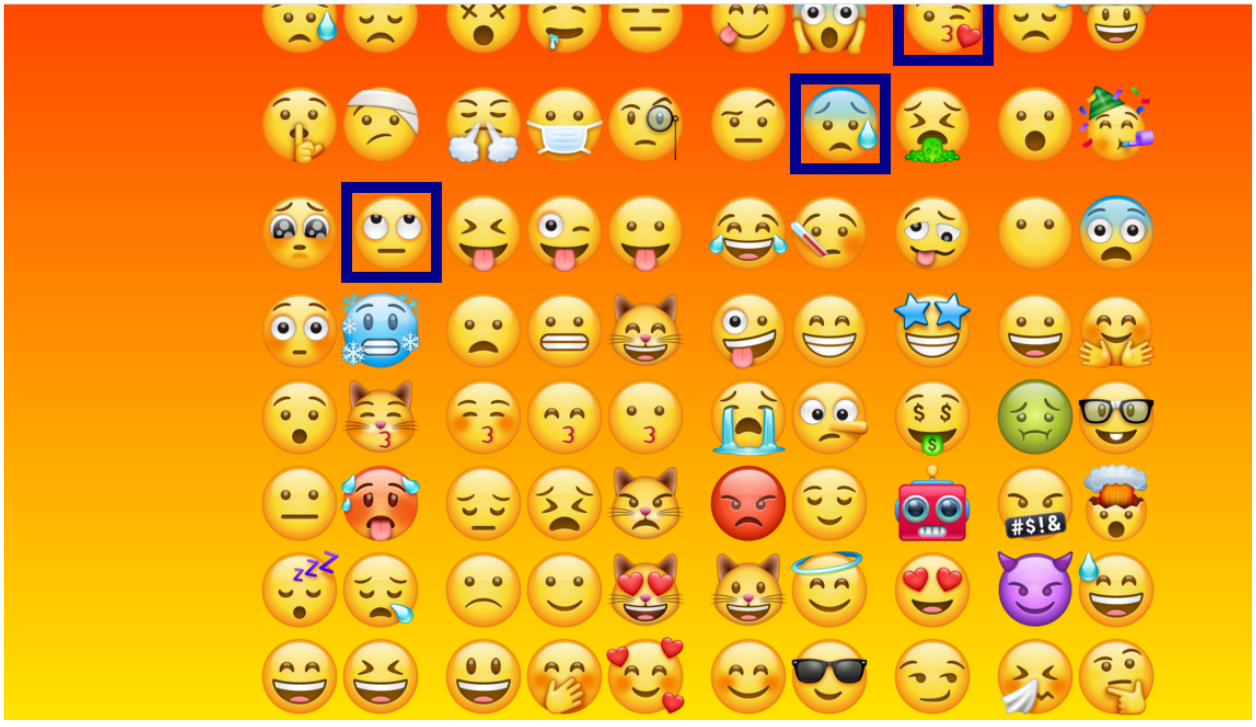


We designed the chat by adding a few fancy effects. The Emojis will shake and make various sounds when they are clicked on.



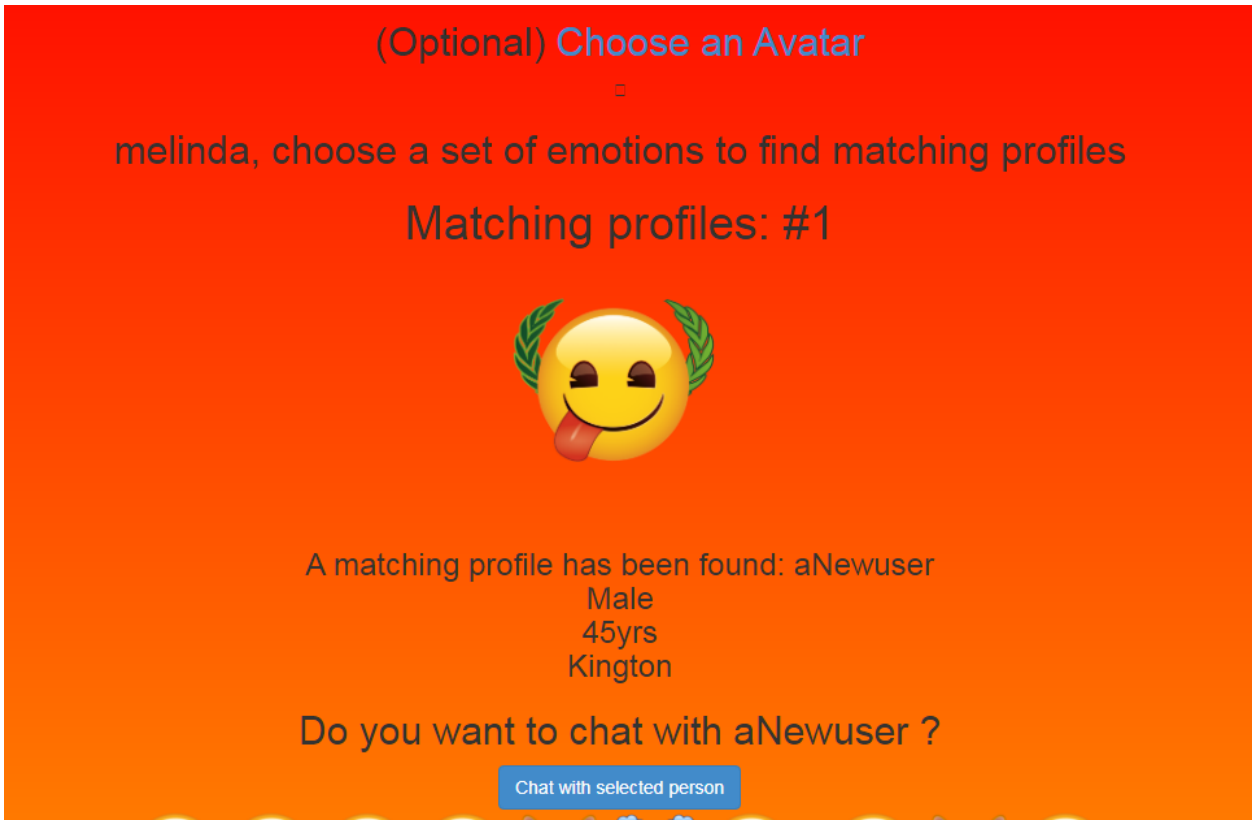
When you click on the blue sphere, a form allows the registration of new users.

Then the new user must select a unique sequence of Emojis. This will allow him/her to be found later on.

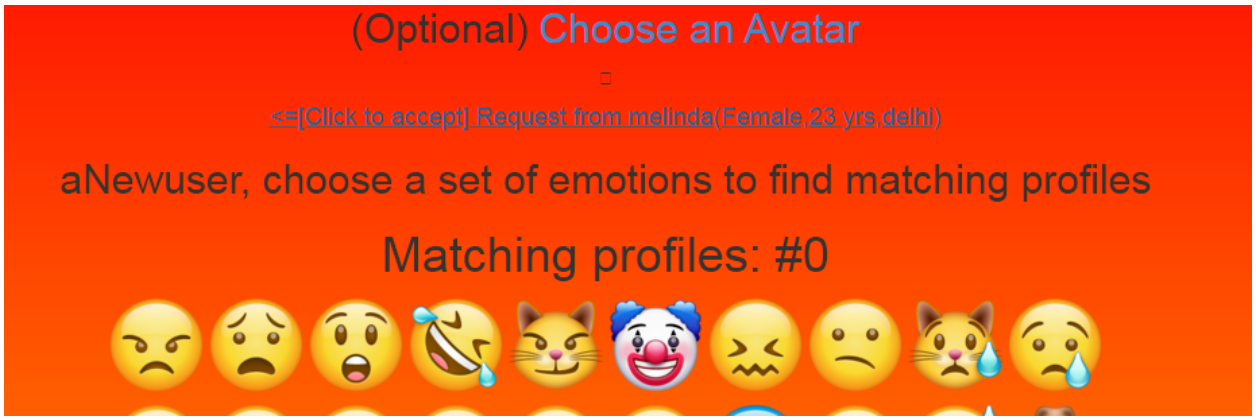


We put a lot of emojis in the chat so that there can be a wide range of possible choices.

Another user can try some combinations and find the corresponding unique matching profile:



Once a request has been accepted the two soulmates can chat with each other.



The chat system is on-to-one and there are a lot of emojis available!

Behind the hood…

Now we must explain how all that magic works and what kind of framework we used.

The database

That chat system is a typical example where you do not need at all a 'professional' database such as SQL databases like MySQL or MSSQL. Such databases are complicated to use and debug. They always require connection strings and can create network errors very easily.

Here we prefer to use a no-SQL approach. MongoDB would have been fined but it's also a big machine and we do not need it.

We simply use object storage: we serialize the objects and store them as flat files.

We will store the following objects:

```
public class ChatMessage : dataVault.dataVaultStorage
   {
       public String from;
```

```
        public String to;
        public String message;
        public string time;
        public int timestamp;
        public String room_id;
    }
```

```
public class ChatRequest : dataVault.dataVaultStorage
    {

        public String From;
        public String To;
        public String date_issued;
        public String date_accepted;
        public String date_refused;

        public Guid request_uid;

        public bool displayed;
        public bool accepted;
        public bool actual;

        public Member from_member;
        public Member to_member;

    }
```

And the chat user data:

```
public class Member : dataVault.dataVaultStorage
   {

       public String name;
       public String password;
       public String email;
       public String genre;
       public String city;
       public String emoticon1;
       public String emoticon2;
       public String emoticon3;
       public String emoticon4;
       public String emoticon5;
       public bool selected = false;
       public int age;
       public String avatar;
   }
```

This is all we need in terms of data storage! From that we can register a user and searching through the database for matching emoticon (emoji) profiles.

matching

Our matching will be mostly performed by a class that will look at the combinations of emoticons:

```
   public class combination
       {

           List<int> L = new List<int>();
           //build the C(k,n) k-uples of the n integers {0,... ,n-1}
           public static List<List<int>> getCombinations(int k, int n)
           {
               List<List<int>> L_ = new List<List<int>>();

               //should never happen
               if ((k>n)||(k == 0)||(n==0))
```

```
                return L_;


            if (k == 1)
            {

                for (int i = 0; i < n; i++)
                {
                    L_.Add(new List<int>() { i+1 });
                }
                    return L_;
            }


            //get it from k-1,n-1 and k,n

            L_.AddRange(getCombinations(k, n - 1));

            List<List<int>> L__ = getCombinations(k - 1, n - 1);
            List<List<int>> L2__ = new List<List<int>>();

            foreach (List<int> Z in L__)
            {
                Z.Add(n);
                L2__.Add(Z);
            }

            L_.AddRange(L2__);
            return L_;
        }

    }

    public static void printListofLists(List<List<int>> L_)
    {
        foreach (List<int> Z in L_)
        {
            foreach (int i in Z)
            {
                Console.Out.Write("{"+i+"}");

                if(i!=Z.Last())
                    Console.Write(",");
```

```
                }

                Console.Out.WriteLine();
             }
        }

        public static bool Equals_(int[] a,int[] b)
        {

            if (a.Length != b.Length)
                return false;

            int[] a_ = a.OrderBy(s1 => s1).ToArray();
            int[] b_ = a.OrderBy(s1 => s1).ToArray();

            for (int i=0;i<a_.Length;i++)
            {

                if (a_[i] != b_[i])
                    return false;

            }

            return true;
        }
    }
```

These routines perform the matching itself:

```
public static List<Member> matchingProfiles(List<String> list)
        {


            if (list == null)
                return null;

            List<Member> Z = new List<Member>();

            int n = list.Count;

            if (n == 0)
                return Z;
```

```
        Z= MatchingProfiles(list[0]);

        for (int i = 1; i < n; i++)
        {

            List<Member> A = MatchingProfiles(list[i]);

            Z = Z.Intersect(A).ToList();

        }




        return Z;

    }

    public static List<Member> MatchingProfiles(String img)
    {
        int n = 0;

        List<Member> Z = new List<Member>();
        init();

        var x = L.FindAll(s1 => s1.emoticon1 == img);

            if (x != null)
            n += x.Count;

        Z.AddRange(x);

        x = L.FindAll(s1 => s1.emoticon2 == img);

        if (x != null)
            n += x.Count;

        Z.AddRange(x);

        x = L.FindAll(s1 => s1.emoticon3 == img);

        if (x != null)
```

```
                n += x.Count;

            Z.AddRange(x);

            x = L.FindAll(s1 => s1.emoticon4 == img);

            if (x != null)
                n += x.Count;


            Z.AddRange(x);

            x = L.FindAll(s1 => s1.emoticon5 == img);

            if (x != null)
                n += x.Count;


            Z.AddRange(x);

            return Z;

        }
```

Chat system

These routines will allow us to maintain the chat in itself:

```
  public static List<ChatMessage> PumpMessages(String roomid , String
recipient)
        {

            List<ChatMessage> L =
dataVault.DataVault.getAllObjects<ChatMessage>(new
dataVault.DataVaultKey());

            List<ChatMessage> L0=L.FindAll(S1 => (S1.room_id.ToString() ==
roomid)&&(S1.to==recipient)&&(S1.deleted==false));

            L.Clear();

            foreach(ChatMessage CM in L0)
            {


dataVault.DataVault.deleteObject<ChatMessage>(CM.vault_id.ToString(), new
dataVault.DataVaultKey());

            }

            return L0;

        }



        internal static List<ChatRequest> getNewChatrequests(string email)
        {

            List<ChatRequest> LCR = new List<ChatRequest>();

            LCR = dataVault.DataVault.getAllObjects<ChatRequest>(new
dataVault.DataVaultKey());

            return  LCR.FindAll(s1 => ((s1.displayed ==
false)&&(s1.To==email)&&(s1.deleted==false)));
        }

        public List<ChatMessage> getMessagesOrderedBydate(String To, String
From)
            {
```

```
            List<String> L = new List<String>();

           List<ChatMessage> cm=
dataVault.DataVault.getAllObjects<ChatMessage>(new
dataVault.DataVaultKey());

            List<ChatMessage> cm2 = cm.FindAll(s1 =>
((s1.deleted==false)&&(s1.from == From) && (s1.to ==
To))).OrderByDescending(s1=>s1.timestamp).ToList();

            cm.Clear();

            return cm2;


            }
```

Conclusion

We explained how to develop an original chat in C# and ASP.NET. The source code is provided for free use and without any restrictions … or guarantees. You can download it here.
We hope that you enjoyed the tutorial and that you found some useful inspiration for your own ASP.NET applications!