```
namespace BLL.Concrete
{
    public class UserSecurityService : EmailService, IUserSecurityService
    {
        public UserSecurityService(IUserRepository userRepository, IDbContextScopeFactory 
dbContextScopeFactory) : base(userRepository, dbContextScopeFactory) { }
        public UserSecurityService(IUserRepository userRepository, IDbContextScopeFactory 
dbContextScopeFactory, string emailRegularExpression) : base(userRepository, dbContextScopeFactory, 
emailRegularExpression) { }

        #region IUserSecurityService

        public bool ValidateUser(string email, string password, IEqualityComparer<string> 
passwordComparer)
        {
            UserExceptionsHelper.GetEmailExceptions(email, this.emailValidationRegex);
            UserExceptionsHelper.GetPasswordExceptions(password);
            bool result = false;
            using (var context = dbContextScopeFactory.CreateReadOnly())
            {
                var user = this.repository.GetUserByEmail(email);
                if (user != null && user.IsApproved)
                {
                    result = passwordComparer.Equals(user.Password, password);
                }
            }
            return result;
        }

        public bool ChangePassword(string email, string oldPassword, string newPassword, 
IEqualityComparer<string> passwordComparer)
        {
            UserExceptionsHelper.GetEmailExceptions(email, this.emailValidationRegex);
            UserExceptionsHelper.GetPasswordExceptions(oldPassword, "oldPassword");
            UserExceptionsHelper.GetPasswordExceptions(newPassword, "newPassword");
            bool result = false;
            using (var context = dbContextScopeFactory.Create())
            {
                var user = this.repository.GetUserByEmail(email);
                if (user != null)
                {
                    if (passwordComparer.Equals(user.Password, oldPassword))
                    {
                        user.Password = newPassword;
                        this.repository.Update(user);
                        result = true;
                    }
                }
                context.SaveChanges();
            }
            return result;
        }

        #endregion
    }
}
```

```
namespace BLL.Concrete
{
    public class UserQueryService : EmailService, IUserQueryService
    {
        public UserQueryService(IUserRepository userRepository, IDbContextScopeFactory
dbContextScopeFactory) : base(userRepository, dbContextScopeFactory) { }
        public UserQueryService(IUserRepository userRepository, IDbContextScopeFactory
dbContextScopeFactory, string emailRegularExpression) : base(userRepository, dbContextScopeFactory,
emailRegularExpression) { }

        #region IUserQueryService

        public User GetUser(string id)
        {
            UserExceptionsHelper.GetIdExceptions(id);
            User result = null;
            using (var context = dbContextScopeFactory.CreateReadOnly())
            {
                var user = this.repository.GetUser(id);
                if (user != null)
                {
                    result = user.ToBll();
                }
            }
            return result;
        }
        public User GetUserByEmail(string email)
        {
            UserExceptionsHelper.GetEmailExceptions(email, this.emailValidationRegex);
            User result = null;
            using (var context = dbContextScopeFactory.CreateReadOnly())
            {
                var user = this.repository.GetUserByEmail(email);
                if (user != null)
                {
                    result = user.ToBll();
                }
            }
            return result;
        }
        public IEnumerable<User> GetAllUsers()
        {
            IEnumerable<User> result = new List<User>();
            using (var context = dbContextScopeFactory.CreateReadOnly())
            {
                var users = this.repository.GetAllUsers();
                if (users.Count() != 0)
                {
                    result = users.Select(u => u.ToBll());
                }
            }
            return result;
        }

        #endregion


        public Dialog GetUserDilog(string userId, int dialogId)
        {
            Dialog result = null;
            using (var context = dbContextScopeFactory.CreateReadOnly())
            {
                result = this.repository.GetUserDilog(userId,dialogId).ToBll();
            }
            return result;
        }


        public IEnumerable<Dialog> GetUserDilogs(string userId)
        {
            List<Dialog> result = new List<Dialog>();
            using (var context = dbContextScopeFactory.CreateReadOnly())
            {
                var pResult = this.repository.GetUserDilogs(userId).Select(d => d.ToBll()).ToList();
                foreach(var item in pResult)
                {
                    List<User> users = new List<User>();
                    foreach(var user in item.Users)
                    {
                        var validUser = this.GetUser(user.Id);

                        users.Add( new User { Id = validUser.Id, Profile = validUser.Profile});
                    }
                    result.Add(new Dialog { Id = item.Id, Users = users});
                }
            }
            return result;
        }
    }
}

namespace BLL.Concrete
{
    public class UserCreationService : EmailService, IUserCreationService
    {
        public UserCreationService(IUserRepository userRepository, IDbContextScopeFactory
dbContextScopeFactory) : base(userRepository, dbContextScopeFactory) { }
        public UserCreationService(IUserRepository userRepository, IDbContextScopeFactory
dbContextScopeFactory, string emailRegularExpression) : base(userRepository, dbContextScopeFactory,
emailRegularExpression) { }

        #region IUserCreationService

        public User CreateUser(string email, string password, bool isApproved)
        {
            UserExceptionsHelper.GetEmailExceptions(email, this.emailValidationRegex);
            UserExceptionsHelper.GetPasswordExceptions(password);
            this.CreateUser(email, password, isApproved, DateTime.Now);
            using (var context = dbContextScopeFactory.CreateReadOnly())
            {
                return this.repository.GetUserByEmail(email).ToBll();
            }
        }
        public bool DeleteUser(string email)
        {
            UserExceptionsHelper.GetEmailExceptions(email, this.emailValidationRegex);
            bool result = false;
            using (var context = dbContextScopeFactory.Create())
            {
                var user = this.repository.GetUserByEmail(email);
                if (user != null)
                {
                    this.repository.Delete(user);
                    result = true;
                }
                context.SaveChanges();
            }
            return result;
        }
        public void UpdateUser(User user)
        {
            UserExceptionsHelper.GetIdExceptions(user.Id);
            using (var context = dbContextScopeFactory.Create())
            {
                var dalUser = this.repository.GetUser(user.Id);
                if (dalUser != null)
                {
                    dalUser.IsApproved = user.IsApproved;
                    this.repository.Update(dalUser);
                }
                context.SaveChanges();
            }
        }

        private void CreateUser(string email, string password, bool isApproved, DateTime createDate,
string roleName = "user")
        {
            User result = new User
            {
                Email = email,
                IsApproved = isApproved,
                CreateDate = DateTime.Now,

            };
            using (var context = dbContextScopeFactory.Create())
            {
                this.repository.Add(result.ToDal(password));
                this.repository.AddUserRole(email, roleName);
                context.SaveChanges();
            }
        }

        #endregion
    }
}

namespace BLL.Concrete
{
    public class UserRolesQueryService : EmailService, IUserRolesQueryService
    {
        public UserRolesQueryService(IUserRepository userRepository, IDbContextScopeFactory
dbContextScopeFactory) : base(userRepository, dbContextScopeFactory) { }
        public UserRolesQueryService(IUserRepository userRepository, IDbContextScopeFactory
dbContextScopeFactory, string emailRegularExpression) : base(userRepository, dbContextScopeFactory,
emailRegularExpression) { }

        #region IUserRolesQueryService

        public string[] GetRolesForUser(string email)
        {
            UserExceptionsHelper.GetEmailExceptions(email, this.emailValidationRegex);
            List<string> result = new List<string>();
            using (var context = dbContextScopeFactory.CreateReadOnly())
            {
                var user = this.repository.GetUserByEmail(email);
                if (user != null)
                {
                    result = user.Roles.Value.Select(r => r.Name).ToList();
                }
            }
            return result.ToArray();
        }
        public string[] GetUsersInRole(string roleName)
        {
            RoleExceptionsHelper.GetNameExceptions(roleName);
            List<string> result = new List<string>();
            using (var context = dbContextScopeFactory.CreateReadOnly())
            {
                var usersInRole = this.repository.GetUsersInRole(roleName);
                if (usersInRole.Count() != 0)
                {
                    result = usersInRole.Select(u => u.Email).ToList();
                }
            }
            return result.ToArray();
        }
        public bool IsUserInRole(string email, string roleName)
        {
            UserExceptionsHelper.GetEmailExceptions(email, this.emailValidationRegex);
            RoleExceptionsHelper.GetNameExceptions(roleName);
            bool result = false;
            using (var context = dbContextScopeFactory.CreateReadOnly())
            {
                var user = this.repository.GetUserByEmail(email);
                if (user != null)
                {
                    result = user.Roles.Value.Where(r => r.Name == roleName).Count() > 0;
                }
            }
            return result;
        }

        #endregion
    }
}

namespace MvcUI.Providers
{

    public class MvcUIMembershipProvider : MembershipProvider
    {
        private string providerDescription = "";

        private string emailRegularExpression = @"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}";
        private bool enablePasswordReset = true;
        private int minRequiredPasswordLength = 6;
        private int minRequiredNonalphanumericCharacters = 0;
        private string passwordStrengthRegularExpression = string.Empty;

        private IUserQueryService userQueryService;
        private IUserCreationService userCreationService;
        private IUserSecurityService userSecurityService;

        public MvcUIMembershipProvider() : this(

(IUserQueryService)System.Web.Mvc.DependencyResolver.Current.GetService(typeof(IUserQueryService)),

(IUserCreationService)System.Web.Mvc.DependencyResolver.Current.GetService(typeof(IUserCreationService))
,

(IUserSecurityService)System.Web.Mvc.DependencyResolver.Current.GetService(typeof(IUserSecurityService))
            ) { }

        public MvcUIMembershipProvider(IUserQueryService userQueryService,IUserCreationService
userCreationService, IUserSecurityService userSecurityService)
```

```
        {
            if(userQueryService == null)
            {
                throw new System.ArgumentNullException("userQueryService", "User query service is 
null.");
            }
            if (userCreationService == null)
            {
                throw new System.ArgumentNullException("userCreationService", "User creation service is 
null.");
            }
            if (userSecurityService == null)
            {
                throw new System.ArgumentNullException("userSecurityService", "User security service is 
null.");
            }
            this.userQueryService = userQueryService;
            this.userCreationService = userCreationService;
            this.userSecurityService = userSecurityService;
        }

        #region Added

        public bool IsDuplicateEmail(string email)
        {
            bool result = false;
            if (IsValidEmail(email))
            {
                var user = this.userQueryService.GetUserByEmail(email);
                result = user != null;
            }
            return result;
        }

        #endregion

        #region Overridden

        #region Filds

        public override string ApplicationName { get; set; }

        public override bool RequiresUniqueEmail
        {
            get { return true; }
        }
        public override bool RequiresQuestionAndAnswer
        {
            get { return false; }
        }
        public override bool EnablePasswordRetrieval
        {
            get { return false; }
        }
        public override int PasswordAttemptWindow
        {
            get { return -1; }
        }
        public override int MaxInvalidPasswordAttempts
        {
            get { return -1; }
        }
        public override MembershipPasswordFormat PasswordFormat
        {
            get { return MembershipPasswordFormat.Hashed; }
        }

        public override bool EnablePasswordReset
        {
            get { return this.enablePasswordReset; }
        }
        public override int MinRequiredNonAlphanumericCharacters
        {
            get { return this.minRequiredNonalphanumericCharacters; }
        }
        public override int MinRequiredPasswordLength
        {
            get { return this.minRequiredPasswordLength; }
        }
        public override string PasswordStrengthRegularExpression
        {
            get { return this.passwordStrengthRegularExpression; }
        }

        public override void Initialize(string name, NameValueCollection config)
        {
            if (config == null)
            {
                throw new ArgumentNullException("config");
            }
            if (string.IsNullOrEmpty(name))
            {
                name = "DefaultMembershipProvider";
            }
            if (string.IsNullOrEmpty(config["description"]))
            {
                config.Remove("description");
                config.Add("description", this.providerDescription);
            }
            base.Initialize(name, config);
            if (!string.IsNullOrEmpty(config["applicationName"]))
            {
                this.ApplicationName = config["applicationName"];
            }
            else
            {
                this.ApplicationName = GetDefaultAppName();
            }
            config.Remove("connectionStringName");
            if (config["enablePasswordReset"] != null)
            {
                this.enablePasswordReset = Convert.ToBoolean(config["enablePasswordReset"], 
CultureInfo.InvariantCulture);
            }
            if (config["minRequiredNonalphanumericCharacters"] != null)
            {
                this.minRequiredNonalphanumericCharacters = 
Convert.ToInt32(config["minRequiredNonalphanumericCharacters"], CultureInfo.InvariantCulture);
            }
            if (config["minRequiredPasswordLength"] != null)
            {
                this.minRequiredPasswordLength = Convert.ToInt32(config["minRequiredPasswordLength"], 
CultureInfo.InvariantCulture);
            }
            if (config["passwordStrengthRegularExpression"] != null)
            {
                this.passwordStrengthRegularExpression = config["passwordStrengthRegularExpression"];
            }
            if (config["emailRegularExpression"] != null)
            {
                this.emailRegularExpression = config["emailRegularExpression"];
            }
        }

        #endregion

        #region Methods

        public override MembershipUser GetUser(object providerUserKey, bool userIsOnline)
        {
            if (providerUserKey == null)
            {
                throw new System.ArgumentNullException("providerUserKey", "User key is null.");
            }
            User result = null;
            var user = this.userQueryService.GetUser(providerUserKey.ToString());
            if (user != null)
            {
                result = user.ToWeb();
            }
            return result;
        }
        public override string GetUserNameByEmail(string email)
        {
            return email;
        }
        public override MembershipUser GetUser(string username, bool userIsOnline)
        {
            User result = null;
            var user = this.userQueryService.GetUserByEmail(username);
            if (user != null)
            {
                result = user.ToWeb();
            }
            return result;
        }
        public override MembershipUserCollection GetAllUsers(int pageIndex, int pageSize, out int
totalRecords)
        {
            MembershipUserCollection result = new MembershipUserCollection();
            var allUsers = this.userQueryService.GetAllUsers();
            if (allUsers.Count() != 0)
            {
                foreach (var item in allUsers.Select(u => u.ToWeb()).ToList().Skip((pageIndex - 1) *
pageSize).Take(pageSize))
                {
                    result.Add(item);
                }
            }
            totalRecords = allUsers.Count();
            return result;
        }

        public override MembershipUser CreateUser(string username, string password, string email, string
passwordQuestion, string passwordAnswer, bool isApproved, object providerUserKey, out
MembershipCreateStatus status)
        {
            if (!IsValidPassword(password))
            {
                status = MembershipCreateStatus.InvalidPassword;
                return null;
            }
            if (!IsValidEmail(email))
            {
                status = MembershipCreateStatus.InvalidEmail;
                return null;
            }
            if (IsDuplicateEmail(email))
            {
                status = MembershipCreateStatus.DuplicateEmail;
                return null;
            }
            var result = this.userCreationService.CreateUser(email, Crypto.HashPassword(password), 
isApproved);
            status = MembershipCreateStatus.Success;
            return result.ToWeb();
        }
        public override bool DeleteUser(string username, bool deleteAllRelatedData)
        {
            return this.userCreationService.DeleteUser(username);
        }
        public override void UpdateUser(MembershipUser user)
        {
            this.userCreationService.UpdateUser(user.ToBll());
        }

        public override bool ValidateUser(string username, string password)
        {
            return this.userSecurityService.ValidateUser(username, password, new 
PasswordComparer(Crypto.VerifyHashedPassword));
        }
        public override bool ChangePassword(string username, string oldPassword, string newPassword)
        {
            return this.userSecurityService.ChangePassword(username, oldPassword, newPassword, new 
PasswordComparer(Crypto.VerifyHashedPassword));
        }

        #endregion

        #endregion

        #region Not supported

        public override int GetNumberOfUsersOnline()
        {
            throw new System.NotSupportedException();
        }
        public override string GetPassword(string username, string answer)
        {
            throw new System.NotSupportedException("Password retrieval is not supported.");
        }
        public override string ResetPassword(string username, string answer)
        {
            throw new System.NotSupportedException("Password generation is not supported.");
        }
        public override bool ChangePasswordQuestionAndAnswer(string username, string password, string 
newPasswordQuestion, string newPasswordAnswer)
        {
            throw new System.NotSupportedException("Question and answer are not supported.");
        }
        public override bool UnlockUser(string userName)
        {
            throw new System.NotSupportedException("Locking of users is not supported.");
        }
        public override MembershipUserCollection FindUsersByEmail(string emailToMatch, int pageIndex, 
int pageSize, out int totalRecords)
```

```
        {
            throw new System.NotSupportedException("Not unique emails are not supported.");
        }
        public override MembershipUserCollection FindUsersByName(string usernameToMatch, int pageIndex,
int pageSize, out int totalRecords)
        {
            throw new System.NotSupportedException("Username are equal to email. Not unique emails are
not supported.");
        }

        #endregion

        #region Private methods

        private static string GetDefaultAppName()
        {
            try
            {
                string applicationVirtualPath = HostingEnvironment.ApplicationVirtualPath;
                if (string.IsNullOrEmpty(applicationVirtualPath))
                {
                    applicationVirtualPath = Process.GetCurrentProcess().MainModule.ModuleName;
                    int index = applicationVirtualPath.IndexOf('.');
                    if (index != -1)
                    {
                        applicationVirtualPath = applicationVirtualPath.Remove(index);
                    }
                }
                if (string.IsNullOrEmpty(applicationVirtualPath))
                {
                    return "/";
                }
                return applicationVirtualPath;
            }
            catch (Exception)
            {
                return "/";
            }
        }
        private static ConnectionStringSettings GetConnectionString(string connectionstringName)
        {
            if (string.IsNullOrEmpty(connectionstringName))
            {
                throw new System.ArgumentNullException("connectionstringName", "Connectionstring name is
null.");
            }
            ConnectionStringSettings settings =
ConfigurationManager.ConnectionStrings[connectionstringName];
            if (settings == null)
            {
                throw new System.InvalidOperationException("Configuration manager returned null.");
            }
            return settings;
        }

        bool IsValidPassword (string password)
        {
            bool result = true;
            if (string.IsNullOrWhiteSpace(password))
            {
                result = false;
            }
            if (password.Length < this.MinRequiredPasswordLength)
            {
```

```
            result = false;
        }
        if (this.MinRequiredNonAlphanumericCharacters > 0)
        {
            int num = 0;
            for (int i = 0; i < password.Length; i++)
            {
                if (!char.IsLetterOrDigit(password[i]))
                {
                    num++;
                }
            }
            if (num < this.MinRequiredNonAlphanumericCharacters)
            {
                result = false;
            }
        }
        if (!String.IsNullOrWhiteSpace(this.passwordStrengthRegularExpression))
        {
            Regex regex = new Regex(this.passwordStrengthRegularExpression);
            if (!regex.IsMatch(password))
            {
                result = false;
            }
        }
        return result;
    }

    bool IsValidEmail(string email)
    {
        bool result = true;
        if (string.IsNullOrWhiteSpace(email))
        {
            result = false;
        }
        if (!IsEmail(email))
        {
            result = false;
        }
        return result;
    }
    private bool IsEmail(string email)
    {
        bool result = true;
        if (!String.IsNullOrWhiteSpace(this.emailRegularExpression))
        {
            Regex regex = new Regex(this.emailRegularExpression);
            result = regex.IsMatch(email);
        }
        return result;
    }

    #endregion
```