- **User Controller**

```
namespace Controllers
{
    /// <summary>
    /// Controller class for the /api/user REST route
    /// </summary>
    [ApiController]
    [Route("api/[controller]")]
    public class UserController : ControllerBase
    {
        /// <summary>
        /// The user service that tests if an user exists
        /// </summary>
        private static IUser _userService = new UserService();

        /// <summary>
        /// Get method
        /// </summary>
        /// <returns>
        /// An IActionResult object containing the response code and a list
of users <para/>
        /// Status code - OK, if there are users<para/>
        /// Status code - NO CONTENT, if the there are no users
        /// </returns>
        [HttpGet]
        public IActionResult GetUsers()
        {
            List<UserEntity> users = _userService.GetUsers();

            if (users.Count != 0)
            {
                return Ok(users);
            }

            return NoContent();
        }

        /// <summary>
        /// Gets one user
        /// </summary>
        /// <param name="username">the user id</param>
        /// <returns>
        /// Status code - OK, if the user was found
        /// Status code - NOT FOUND, if the user wasn't found
        /// </returns>
        [HttpGet("{username}")]
        public IActionResult GetUser(string username)
        {
            UserEntity user = _userService.GetUser(username);
```

```
            if (user == null)
            {
                return NotFound();
            }

            return Ok(user);
        }

        /// <summary>
        /// Creates an user
        /// </summary>
        /// <param name="user">the user, taken from the body of the
request</param>
        /// <returns>Status code - CREATED</returns>
        [HttpPost]
        public IActionResult PostUser([FromBody] UserEntity user)
        {
            if (user == null)
            {
                return BadRequest();
            }

            _userService.CreateUser(user);
            return CreatedAtAction(nameof(GetUser), new { user = user },
user);
        }

        [HttpPut("{username}")]
        public IActionResult UpdateUser(string username, [FromBody]
UserEntity user)
        {
            if (_userService.UpdateUser(username, user))
            {
                return Ok();
            }

            return NotFound();
        }

        [HttpDelete("{username}")]
        public IActionResult DeleteUser(string username)
        {
            if (_userService.DeleteUser(username))
            {
                return Ok();
            }

            return NotFound();
        }
    }
}
```

- **User Interface**

```
namespace Interfaces
{
    public interface IUser
    {
        /// <summary>
        /// Method that returns an user based on the username provided
        /// </summary>
        /// <param name="username">username of the searched user</param>
        /// <returns>the searched user</returns>
        UserEntity GetUser(string username);

        /// <summary>
        /// Returns all the users stored
        /// </summary>
        /// <returns>the list of requested users</returns>
        List<UserEntity> GetUsers();

        /// <summary>
        /// Creates an user
        /// </summary>
        /// <param name="user">a user object filled with the required
properties</param>
        /// <returns>
        /// true if the user has been created<para/>
        /// false if the user wasn't created
        /// </returns>
        bool CreateUser(UserEntity user);

        /// <summary>
        /// Updates an user
        /// </summary>
        /// <param name="username">the id</param>
        /// <param name="user">the user entity</param>
        /// <returns>
        /// true if the user has been updated<para/>
        /// false if the user wasn't updated
        /// </returns>
        bool UpdateUser(string username, UserEntity user);

        /// <summary>
        /// Deletes an user
        /// </summary>
        /// <param name="username">the id</param>
        /// <returns>
        /// true if the user has been deleted<para/>
        /// false if the user wasn't deleted
        /// </returns>
        bool DeleteUser(string username);
    }
}
```

- **User Service**

```
namespace Services
{
    public class UserService : IUser
    {
        private static Dictionary<string, UserEntity> _users = new
Dictionary<string, UserEntity>();

        #region Public Methods

        /// <summary>
        /// Creates an user
        /// </summary>
        /// <param name="user">a user object filled with the required
properties</param>
        /// <returns>
        /// true if the user has been created<para/>
        /// false if the user wasn't created
        /// </returns>
        public bool CreateUser(UserEntity user)
        {
            if (!_users.ContainsKey(user.Username))
            {
                _users.Add(user.Username, user);
                return true;
            }

            return false;
        }

        /// <summary>
        /// Deletes an user
        /// </summary>
        /// <param name="username">the id</param>
        /// <returns>
        /// true if the user has been deleted<para/>
        /// false if the user wasn't deleted
        /// </returns>
        public bool DeleteUser(string username)
        {
            if (_users.ContainsKey(username))
            {
                _users.Remove(username);
                return true;
            }

            return false;
        }

        /// <summary>
        /// Method that returns an user based on the username provided
        /// </summary>
```

```
        /// <param name="username">username of the searched user</param>
        /// <returns>the searched user</returns>
        public UserEntity GetUser(string username)
        {
            if (_users.ContainsKey(username))
            {
                return _users[username];
            }

            return null;
        }

        /// <summary>
        /// Returns all the users stored
        /// </summary>
        /// <returns>the list of requested users</returns>
        public List<UserEntity> GetUsers()
        {
            return _users.Values.ToList();
        }

        /// <summary>
        /// Updates an user
        /// </summary>
        /// <param name="username">the id</param>
        /// <param name="user">the user entity</param>
        /// <returns>
        /// true if the user has been updated<para/>
        /// false if the user wasn't updated
        /// </returns>
        public bool UpdateUser(string username, UserEntity user)
        {
            if (_users.ContainsKey(username))
            {
                _users[username] = user;
                return true;
            }

            return false;
        }

        #endregion
    }
}
```

- **User Model**

```
namespace Models
{
    public enum Gender { MALE, FEMALE };

    public class UserEntity
    {
        [JsonProperty("username")]
        public string Username { get; set; }

        [JsonProperty("firstName")]
        public string FirstName { get; set; }

        [JsonProperty("lastName")]
        public string LastName { get; set; }

        [JsonProperty("email")]
        public string Email { get; set; }

        [JsonProperty("phone")]
        public string Phone { get; set; }

        [JsonProperty("password")]
        public string Password { get; set; }

        [JsonProperty("sex")]
        public Gender Sex { get; set; }

        [JsonProperty("city")]
        public string City { get; set; }

        [JsonProperty("country")]
        public string Country { get; set; }
    }
}
```

- **Abstract Factory**

```
namespace Commons
{
    /// <summary>
    /// Abstract class for building a factory design pattern
    /// </summary>
    public abstract class AbstractFactory
    {
        /// <summary>
        /// Returns a friend based on the close factor
        /// </summary>
        /// <param name="IsClose">"close" if he/she is close or "not
close"</param>
        /// <param name="data">the FriendEntity object</param>
        /// <returns>a new FriendEntity with the close factor
filled</returns>
        public abstract FriendEntity GetFriend(string IsClose , FriendEntity
data);
    }
}
```

- **Factory Builder**

```
namespace Commons
{
    /// <summary>
    /// Class that returns a factory based on the argument provided
    /// </summary>
   public class FactoryBuilder
    {
        /// <summary>
        /// Method that returns a factory
        /// </summary>
        /// <param name="type">the factory's type</param>
        /// <returns>A FriendFactory or null</returns>
        public static AbstractFactory GetFactory(string type)
        {
            if (type.Equals("friend"))
                return new FriendFactory();
            return null;
        }
    }
}
```

- **Friend Factory**

```
namespace Commons
{
    /// <summary>
    /// Implementation class for building a factory design pattern
    /// </summary>
    public class FriendFactory : AbstractFactory
    {
        /// <summary>
        /// Returns a friend based on the close factor
        /// </summary>
        /// <param name="IsClose">"close" if he/she is close or "not
close"</param>
        /// <param name="data">the FriendEntity object</param>
        /// <returns>a new FriendEntity with the close factor
filled</returns>
        public override FriendEntity GetFriend(string IsClose , FriendEntity
data)
        {
            bool check = false;
            if(IsClose.Equals("close"))
            {
                check = true;
            }
            return new FriendEntity { IsClose = check , Username =
data.Username };
        }
    }
}
```