Pass Task 8.2 Securing your enterprise application – Source code

IMyUserBO

```
public interface IMyUserBO
    {
        [OperationContract]
        List<MyUserDto> GetAllUsers();
        [OperationContract]
        void CreateUser(MyUserDto userDto);
        [OperationContract]
        void UpdateUser(MyUserDto userDto);
        [OperationContract]
        MyUserDto GetUserById(string id);
        [OperationContract]
        void DeleteUser(string userId);

        [OperationContract]
        List<string> ResetPassword(ResetPasswordRequestDto resetPasswordRequest);
        [OperationContract]
        List<string> PrecheckForResetPassword(ResetPasswordRequestDto resetPasswordRequest);

        [OperationContract]
        MyUserDto ValidateLogin(string username, byte[] password);
    }
```

MyUserBO

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]
    public class MyUserBO : IMyUserBO
    {
        private IMyUserDao _myUserDao;
        private UserMaintenanceService _userService;

        public MyUserBO()
        {
            _myUserDao = new MyUserDao();

            // initialize user service to listen to the MSMQ
            _userService = UserMaintenanceService.Instance;
        }

        public void CreateUser(MyUserDto userDto)
        {
            var myUser = Mapping.Map<MyUser, MyUserDto>(userDto);
            _myUserDao.Create(myUser);
        }

        public void DeleteUser(string userId)
        {
            var myUser = new MyUser()
            {
                UserID = userId
            };
            _myUserDao.Delete(myUser);
        }

        public void DeleteUser(MyUserDto userDto)
        {
            var myUser = Mapping.Map<MyUser, MyUserDto>(userDto);
            _myUserDao.Delete(myUser);
        }

        public List<MyUserDto> GetAllUsers()
        {
            var allUser = _myUserDao.GetAll();
            var mappingResult = Mapping.Map<IEnumerable<MyUserDto>, IEnumerable<MyUser>>(allUser);
```

```
            return mappingResult.ToList();
        }

        public MyUserDto GetUserById(string id)
        {
            var user = _myUserDao.GetById(id);
            if (user != null)
            {
                var mappingResult = Mapping.Map<MyUserDto, MyUser>(user);
                return mappingResult;
            }
            return null;
        }

        /// <summary>
        /// PT3.2 Implementation
        /// </summary>
        /// <param name="resetPasswordRequest"></param>
        /// <returns></returns>
        public List<string> ResetPassword(ResetPasswordRequestDto resetPasswordRequest)
        {
            List<string> errors = new List<string>();

            var user = _myUserDao.GetById(resetPasswordRequest.UserId);
            if (user == null)
                errors.Add("User is not found");

            if (!user.Password.Equals(resetPasswordRequest.OldPassword))
                errors.Add("Your password doesn't match");

            if (errors.Count > 0)
                return errors;

            user.Password = resetPasswordRequest.NewPassword;
            _myUserDao.Update(user);

            return errors;
        }

        public void UpdateUser(MyUserDto userDto)
        {
            var myUser = Mapping.Map<MyUser, MyUserDto>(userDto);
            _myUserDao.Update(myUser);
        }


        public List<string> PrecheckForResetPassword(ResetPasswordRequestDto resetPasswordRequest)
        {
            List<string> errors = new List<string>();
            // Reset password validation:
            //var user = _myUserDao.GetById(resetPasswordRequest.UserId);
            //if (user == null)
            //    errors.Add("User is not found");

            //if (!user.SecAns.Equals(resetPasswordRequest.SecAns))
            //    errors.Add("Your security answer is invalid");

            return errors;
        }


        public MyUserDto ValidateLogin(string username, byte[] encryptedPassword)
        {
            var user = _myUserDao.GetByUserName(username);
            if (user == null)
                return null;

            // check for valid password
            var userPassword = EncryptionService.EncryptPassword(user.Password);
            if(Encoding.UTF8.GetString(encryptedPassword)== Encoding.UTF8.GetString(userPassword))
            {
                var dto = new MyUserDto()
                {
                    UserID = user.UserID,
```

```
                    Address = user.Address,
                    Email = user.Email,
                    Name = user.Name,
                    Role = user.Role,
                    Tel = user.Tel
                };

                return dto;
            }

            return null;
        }
    }
```

MyUserDto

```
    /// <summary>
    /// Data transfer object doesn't contain sensitive information
    /// </summary>
    [Serializable]
    [DataContract]
    public class MyUserDto
    {
        [DataMember]
        public string UserID { get; set; }
        [DataMember]
        public string Name { get; set; }
        [DataMember]
        public string Email { get; set; }
        [DataMember]
        public string Tel { get; set; }
        [DataMember]
        public string Address { get; set; }
        [DataMember]
        public string Role { get; set; }
    }
```

EncryptionService

```
    public class EncryptionService
    {
        public static byte[] EncryptPassword(string password)
        {
            byte[] hashBytes = Encoding.UTF8.GetBytes(password);

            SHA1 sha1 = new SHA1CryptoServiceProvider();
            return sha1.ComputeHash(hashBytes);
        }

        public static string MD5Hash(string plainText)
        {
            byte[] hash = MD5.Create().ComputeHash(Encoding.UTF8.GetBytes(plainText));
            return Encoding.UTF8.GetString(hash);
        }
    }
```

ResetPassword Method

```
[HttpPost]
        public HttpResponseMessage ResetPassword(ResetPasswordRequestViewModel resetPasswordModel)
        {
            var jsonMessage = new JsonResponseMessage();
            if (!ModelState.IsValid)
            {
                jsonMessage.Error("Invalid data", resetPasswordModel);
                return Request.CreateResponse(HttpStatusCode.BadRequest, jsonMessage);
            }

            try
            {
                var resetPasswordDto = new ResetPasswordRequestDto()
                {
                    OldPassword = resetPasswordModel.OldPassword,
```

```
                    UserId = resetPasswordModel.UserID,
                    NewPassword = resetPasswordModel.NewPassword
                };

                var errors = _myUserService.ResetPassword(resetPasswordDto);

                if (errors.Length > 0)
                {
                    jsonMessage.Error(string.Join("; ", errors));
                    return Request.CreateResponse(HttpStatusCode.BadRequest, jsonMessage);
                }

                jsonMessage.Success("Ok", resetPasswordModel);
                return Request.CreateResponse(HttpStatusCode.Accepted, jsonMessage);
            }
            catch (Exception ex)
            {
                jsonMessage.Error("Server error: " + ex.Message);
                return Request.CreateResponse(HttpStatusCode.InternalServerError, jsonMessage);
            }
        }
```

AccountController

```
public class AccountController : Controller
    {
        MyUserBOClient _myUserService = new MyUserBOClient("wsHttpBinding_IMyUserBO");

        [AllowAnonymous]
        public ActionResult Login()
        {
            if(User.Identity.IsAuthenticated)
            {
                if (User.IsInRole(Constants.RoleValue.Administrator))
                {
                    return RedirectToAction("Index", "Admin");
                }
                else if (User.IsInRole(Constants.RoleValue.Employee))
                {
                    return RedirectToAction("Index", "Employee");
                }
                else
                {
                    return RedirectToAction("Index", "Home");
                }
            }
            return View();
        }

        [HttpPost]
        [AllowAnonymous]
        public ActionResult Login(LoginViewModel vm, string returnUrl = "")
        {
            if (ModelState.IsValid)
            {
                try
                {
                    var userPrincipal = UserPrincipal.ValidateLogin(vm.UserName, vm.Password);
                    var userData = JsonConvert.SerializeObject(userPrincipal.SerializedData);

                    FormsAuthenticationTicket authTicket = new FormsAuthenticationTicket(
                        1,
                        userPrincipal.Identity.Name,
                        DateTime.Now,
                        DateTime.Now.AddMinutes(30),
                        false, //pass here true, if you want to implement remember me functionality
                        userData);

                    string encTicket = FormsAuthentication.Encrypt(authTicket);
                    HttpCookie faCookie = new HttpCookie(FormsAuthentication.FormsCookieName,
encTicket);
                    Response.Cookies.Add(faCookie);

                    if(returnUrl!="")
```

```
                    {
                        return Redirect(returnUrl);
                    }
                    else if (userPrincipal.IsInRole(Constants.RoleValue.Administrator))
                    {
                        return RedirectToAction("Index", "Admin");
                    }
                    else if (userPrincipal.IsInRole(Constants.RoleValue.Employee))
                    {
                        return RedirectToAction("Index", "Employee");
                    }
                    else
                    {
                        return RedirectToAction("Index", "Home");
                    }
                }
                catch (Exception)
                {
                    ModelState.AddModelError("", "Incorrect username and/or password");
                }
            }

            return View(vm);
        }

        [AllowAnonymous]
        public ActionResult LogOut()
        {
            FormsAuthentication.SignOut();
            return RedirectToAction("Login", "Account", null);
        }
    }
```

AuthorizeAttribute decorated on Controllers

```
[Authorize(Roles = "Administrator|Employee")]
    public class EmployeeController : Controller
    {

    }

[Authorize(Roles ="Administrator")]
    public class AdminController : Controller
    {
    }
```

UserIdentity

```
public class UserIdentity : IIdentity
    {
        private string _userID;
        private string _email;
        private string _fullName;
        private string _tel;
        private string _address;
        private string _role;

        #region IIdentity methods
        public string AuthenticationType
        {
            get
            {
                return "Custom Authentication";
            }
        }

        public bool IsAuthenticated
```

```
        {
            get
            {
                return true;
            }
        }

        public string Name
        {
            get
            {
                return _fullName;
            }
        }
        #endregion

        #region Constructor
        // get user by user id
        public UserIdentity(MyUserDto user)
        {
            if (user == null) throw new Exception("User login failed");
            this._address = user.Address;
            this._email = user.Email;
            this._fullName = user.Name;
            this._role = user.Role;
            this._tel = user.Tel;
            this._userID = user.UserID;

        }
        #endregion

        #region Public attributes
        public string UserID
        {
            get { return _userID; }
        }

        public string Email
        {
            get { return _email; }
        }

        public string Tel
        {
            get { return _tel; }
        }

        public string Address
        {
            get { return _address; }
        }

        public string Role
        {
            get { return _role; }
        }
        #endregion
    }
```

UserPrincipal

```
public class UserPrincipal : IPrincipal
```

```
    {
        private UserIdentity _identity;
        private List<string> _roleList;
        MyUserBOClient _clientProxy = ServiceFactory.CreateUserBoClient();

        MyUserDto _serializedData;

        public IIdentity Identity
        {
            get
            {
                return _identity;
            }
        }

        public bool IsInRole(string role)
        {
            if(role.Contains("|"))
            {
                var checkRoleList = role.Split("|".ToCharArray());
                foreach(var roleItem in checkRoleList)
                {
                    if (_roleList.Contains(roleItem)) return true;
                }
                return false;
            }
            else if(role.Contains("&"))
            {
                var checkRoleList = role.Split("&".ToCharArray());
                foreach (var roleItem in checkRoleList)
                {
                    if (!_roleList.Contains(roleItem)) return false;
                }
                return true;
            }
            return _roleList.Contains(role);
        }

        public MyUserDto SerializedData
        {
            get
            {
                return _serializedData;
            }
        }

        public UserPrincipal(MyUserDto userData)
        {
            _serializedData = userData;
            _identity = new UserIdentity(userData);
            _roleList = _identity.Role.Split(",".ToCharArray()).ToList();
        }

        public UserPrincipal(string userId)
        {
            var userData = _clientProxy.GetUserById(userId);
            _identity = new UserIdentity(userData);
        }

        public static UserPrincipal ValidateLogin(string userId, string password)
        {
            var cyperPass = EncryptionService.EncryptPassword(password);
```

```
            var clientProxy = ServiceFactory.CreateUserBoClient();

            var userData = clientProxy.ValidateLogin(userId, cyperPass);

            if(userData == null)
            {
                throw new Exception("Your username and password is invalid");
            }
            return new UserPrincipal(userData);
        }


    }
```

ResetPasswordViewModel

```
public class ResetPasswordRequestViewModel
    {
        [Required]
        public string UserID { get; set; }
        public string Name { get; set; }
        public string OldPassword { get; set; }
        public string NewPassword { get; set; }
        public string ConfirmPassword { get; set; }
    }
```

LoginViewModel

```
public class LoginViewModel
    {
        public string UserName { get; set; }
        public string Password { get; set; }
    }
```

Login.cshtml

```
@model SwinSchool.WebUI.Models.LoginViewModel
@{
    ViewBag.Title = "Login";
}

<h3>Login to SECURE</h3>
<hr />
@using (Html.BeginForm())
{
    <div class="form form-horizontal">
        <div class="form-group">
            @Html.LabelFor(model => model.UserName, new { @class = "control-label
col-xs-3" })

            <div class="col-xs-9">
                @Html.TextBoxFor(model => model.UserName, new { @class = "form-
control"})
                @Html.ValidationMessageFor(model => model.UserName)
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.Password, new { @class = "control-label
col-xs-3" })

            <div class="col-xs-9">
                @Html.PasswordFor(model => model.Password, new { @class = "form-
control"})
```

```
                @Html.ValidationMessageFor(model => model.Password)
            </div>
        </div>
        <div class="form-group">
            <div class="col-xs-3"></div>
            <div class="col-xs-9">
                <input type="submit" value="Login" class="btn btn-primary" />
            </div>
        </div>
    </div>

}
```

Web.config

```
<system.web>
    <authentication mode="Forms">
      <forms loginUrl="~/Account/Login" timeout="2880" />
    </authentication>
</system.web>
```

Global.asax

```
protected void Application_PostAuthenticateRequest(Object sender, EventArgs e)
        {
            HttpCookie authCookie =
Request.Cookies[FormsAuthentication.FormsCookieName];
            if (authCookie != null)
            {

                FormsAuthenticationTicket authTicket =
FormsAuthentication.Decrypt(authCookie.Value);

                MyUserDto serializeModel =
JsonConvert.DeserializeObject<MyUserDto>(authTicket.UserData);
                if (serializeModel == null)
                {
                    FormsAuthentication.SignOut();
                    return;
                }
                UserPrincipal newUser = new UserPrincipal(serializeModel);
                HttpContext.Current.User = newUser;
            }

        }
```

References:

(1) BitBucket source code: https://bitbucket.org/werynguyen/swinschool/src/4c090c6cef0ddb9b49a1e5f23c0886d9975ed88b/?at=PT82