VacationProject application
Internship project – Technical documentation

Table of Contents

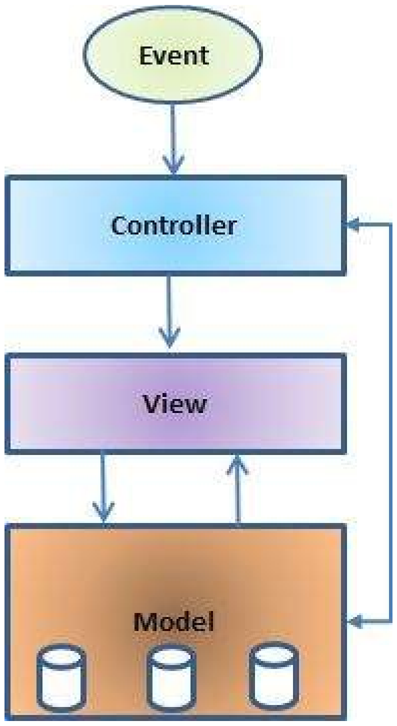
General description of MVC architecture

The goal of our project was the implementation of software solution for the provision of annual leave. The entire application is done in *.NET* technology, based on MVC architecture, using entity framework and code first approach.

Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A *ModelViewController* pattern is made up of the following three parts:

- **Model** - The lowest level of the pattern which is responsible for maintaining data.
- **View** - This is responsible for displaying all or a portion of the data to the user.
- **Controller** - Software Code that controls the interactions between the Model and View.

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.



The model

The model is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself.

The view

A presentation of data in a particular format, triggered by a controller's decision to present the data. They are script based templating systems like JSP, ASP, PHP and very easy to integrate with AJAX technology.

The controller

The controller is responsible for responding to user input and perform interactions on the data model objects. The controller receives the input, it validates the input and then performs the business operation that modifies the state of the data model.

The application Vacation Project is made in the programming tool Microsoft Visual Studio, targeted framework .NET Framework 4.5.2.

VacationProject architecture

Web application is organized and consists of four projects:

- VacationProject
- VacationProject.DAL
- VacationProject.Model
- VacationProject.Service

VacationProject.Model

Designing Data Model

For creating and accessing Users and Roles dana, access model classes User.cs and Role.cs have been created.

```
public class User
    {
        public int UserId { get; set; }
        [Required]
        public String Username { get; set; }
        [Required]
        public String Email { get; set; }
        [Display(Name ="Password")]
        public String Password { get; set; }
        [Display(Name = "First Name")]
        public String FirstName { get; set; }
        [Display(Name = "Last Name")]
        public String LastName { get; set; }
        [Display(Name = "User is active")]
        public Boolean IsActive { get; set; }
        [Display(Name = "Create Date")]
        public DateTime CreateDate { get; set; }
        [Display(Name ="Vacation Days")]
        public int VacationDays { get; set; }
        public virtual ICollection<Role> Roles { get; set; }
        public virtual ICollection<VacationApproving>
                    VacationApprovings{ get; set; }
        public virtual ICollection<VacationRequest>
                    VacationRequests { get; set; }
    }
}
public class Role
    {
        public int RoleId { get; set; }
        [Required]
        public string RoleName { get; set; }
        public string Description { get; set; }
        public virtual ICollection<User> Users { get; set; }
    }
```

For creating vacation request and sending requests to administrators for approval, classes VacationApproving.cs and VacationRequests.cs have been created.

```
public class VacationApproving
    {
        public int ID { get; set; }
        public VacationRequest VacationRequest { get; set; }
        public User Aprover { get; set; }
        public bool? Answer { get; set; }
    }
public enum Status { Approved, Reject, Pending,Invalid}
    public class VacationRequest
    {
        public VacationRequest()
        {
            ApprovingsFromManagers = new List<VacationApproving>();
        }
        public int ID { get; set; }
        public User User { get; set; }
        [Display(Name = "Begin date")]
        public DateTime BeginDate { get; set; }
        [Display(Name = "End date")]
        public DateTime EndDate { get; set; }
        public List<VacationApproving>
                   ApprovingsFromManagers { get; set; }
        [Display(Name = "Status")]
        public Status Status { get; set; }
        public int WorkingDays { get; set; }
    }
```

VacationProject.DAL

Using Entity Framework code first approach, DataContext is implemented, having User and Role entities with its relational mapping details. First we need to configure many to many relationship between User and Roles entites with Fluent API overriding OnModelCreating method:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{

modelBuilder.Conventions.Remove<OneToManyCascadeDeleteConvention>();
            modelBuilder.Entity<User>()
            .HasMany(u => u.Roles)
            .WithMany(r => r.Users)
            .Map(m =>
            {
                m.ToTable("UserRoles");
                m.MapLeftKey("UserId");
                m.MapRightKey("RoleId");
            });
            modelBuilder.Entity<VacationRequest>()
                                .HasRequired<User>(u => u.User)
                                .WithMany(u=>u.VacationRequests)
                                .WillCascadeOnDelete(false);
            modelBuilder.Entity<VacationApproving>()
                                .HasRequired<User>(s => s.Aprover)
                                .WithMany(s => s.VacationApprovings);
}
```

After configuring many to many relationship we specify Entites in DbSet.

```
 public DbSet<User> Users { get; set; }
 public DbSet<Role> Roles { get; set; }
 public DbSet<VacationRequest> VacationRequests { get; set; }
 public DbSet<VacationApproving> VacationApprovings { get; set; }
```

Security

Because standard types of authentication do not meet our requirements, authentication mechanism has been modified to create a custom solution. A user context has principal which represents the identity and roles for that user. A user is authenticated by its identity and assigned roles to a user determine about authorization or permission to access resources.

Custom Authentication

ASP.NET provides IPrincipal and IIdentity interfaces to represents the identity and role for a user. So a custom solution has been created by evaluating the IPrincipal and IIdentity interfaces which are bound to the HttpContext as well as the current thread.

```
 public class CustomPrincipal:IPrincipal
    {
        public IIdentity Identity { get; private set; }
        public bool IsInRole(string role)
        {
            if (roles.Any(r => role.Contains(r)))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        public CustomPrincipal(string Username)
        {
            this.Identity = new GenericIdentity(Username);
        }
        public int UserId { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string[] roles { get; set; }
    }
```

After creating CustomPrincipal object has been implemented into the thread's currentPrinciple property and into the HttpContext's User property to accomplish custom authentication and authorization process.

ASP.NET forms authentication occurs after IIS authentication is completed.

The FormsAuthentication class creates the authentication cookie automatically when SetAuthCookie() or RedirectFromLoginPage() methods are called. The value of authentication cookie contains a string representation of the encrypted and signed FormsAuthenticationTicket object.

FormsAuthenticationTicket object has been created by specifying the cookie name, version of the cookie, directory path, issue date of the cookie, expiration date of the cookie, whether the cookie should be persisted, and user-defined data.

User defined data is passed into FormAuthenticationTicket with object created from CustomPrincipalSerializeModel Class.

```
public class CustomPrincipalSerializeModel
    {
        public int UserId { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string[] roles { get; set; }
    }
```

For implementing this functionality Newtonsoft.JSON plugin has been used, installed from Nuget packet manager. User information is passed to CustomPrincipalSerializeModel object, and with Json.Convert.SerializeObject method we created User data that we passed into FormAuthenticationTicket.

This functionality is implemented in AccountService, AddCookie method.

```
CustomPrincipalSerializeModel serializeModel = new
CustomPrincipalSerializeModel();
serializeModel.UserId = user.UserId;
serializeModel.FirstName = user.FirstName;
serializeModel.LastName = user.LastName;
serializeModel.roles = roles;
string userData = JsonConvert.SerializeObject(serializeModel);
FormsAuthenticationTicket authTicket = new FormsAuthenticationTicket(1,
user.Username, DateTime.Now, DateTime.Now.AddMinutes(15),false,userData);
```

Now, this ticket has been encrypted by using the Encrypt method FormsAuthentication class as given below:

```
string encTicket = FormsAuthentication.Encrypt(authTicket);
```

Encrypted ticket is passed into HttpCookie as given below:

```
HttpCookie faCookie = new HttpCookie(FormsAuthentication.FormsCookieName,
                                                                encTicket);
```

This functionality is implemented in Account service method:

```
public HttpCookie AddCookie(User user)
```

Now this Cookie has been used to authenticate Current user. In Global.asax file method Application_PostAuthenticateRequest has been implemented to read Cookie with user data, decrypt cookie, deserialize user data from cookie, and set user data values to current user.

```
protected void Application_PostAuthenticateRequest(Object sender, EventArgs e)
    {
     HttpCookie authCookie =
       Request.Cookies[FormsAuthentication.FormsCookieName];
            if (authCookie != null)
            {
                FormsAuthenticationTicket authTicket =
                   FormsAuthentication.Decrypt(authCookie.Value);
                CustomPrincipalSerializeModel serializeModel =
JsonConvert.DeserializeObject<CustomPrincipalSerializeModel>(authTicket.UserData);
                CustomPrincipal newUser = new
                CustomPrincipal(authTicket.Name);
                newUser.UserId = serializeModel.UserId;
                newUser.FirstName = serializeModel.FirstName;
                newUser.LastName = serializeModel.LastName;
                newUser.roles = serializeModel.roles;
                HttpContext.Current.User = newUser;
            }
        }
```

Custom Authorization

ASP.NET MVC provides Authorization filter to authorize a user. This filter can be applied to an action, a controller, or even globally. This filter is based on AuthorizeAttribute class. This filter has been customized by overriding OnAuthorization() method in CustomAuthorizeAttribute class that implements AuthorizeAttribute class.

```
public override void OnAuthorization(AuthorizationContext filterContext)
```

Now keys can be added in appSettting section of web.config file to set custom authorization for users in role.

```
<appSettings>
        <add key="RolesConfigKey" value="Admin" />
         <add key="UsersConfigKey" value="22,33" />
```

```
 </appSettings>
```

Applying CustomAuthorize attribute

To make secure your admin or user pages, controllers have been decorated with CustomAuthorize attribute and specify the uses or roles to access admin and user pages.

```
[CustomAuthorize(RolesConfigKey = "UsersConfigKey")]

[CustomAuthorize(RolesConfigKey = "RolesConfigKey")]
```

User Authentication

A user will be authenticated if IsAuthenticated property returns true. For authenticating a user you can use one of the following two ways:

1. Thread.CurrentPrincipal.Identity.IsAuthenticated
2. HttpContext.Current.User.Identity.IsAuthenticated

Base View Page for accessing Current User

To access User data in all views, abstract class BaseViewPage has been implemented as shown below:

```
public abstract class BaseViewPage:WebViewPage

   {

        public virtual new CustomPrincipal User

        {

            get { return base.User as CustomPrincipal; }

        }

    }

         public abstract class BaseViewPage<TModel> : WebViewPage<TModel>

    {

        public virtual new CustomPrincipal User

        {

            get { return base.User as CustomPrincipal; }
            }
```

Register this class with in the \Views\Web.config as base class for all views as given below:

```
<system.web.webPages.razor>
   <pages pageBaseType="VacationProject.ViewModel.BaseViewPage">
```

VacationProject.Service

In VacationProject application are implement architecture which contain layers. Service layer is one of them. It implements like ddl and its full name is: VacationProject.Service. It contains five service, one exception and four classes for work with security.

List of services:

- AccountService
- EmailService
- UserService
- VacationApprovingService
- VacationRequestService

Exception:

- SomethingWentWrongException

List of class that work with security:

- BaseViewPage
- CustomAuthorizeAttribute
- CustomPrincipal
- CustomPrincipalSerializeModel

List of services

Main purpose of services are providing result from data base.

Account service

Account service is implemented in AccountService.cs class, and it is collection of methods used for exchanging data between Project files.

AccountService depends on UserService and with those services we supply controllers with information. First namespace System.Web.Helpers has been included, where Crypto Class has been used for encrypting and password verification.

List of methods:

- public string HashPassword(string password)
- public string GetHashedPassword(string username)
- public string GetHashedPassword(int userID)
- public bool IsPasswordMatchWithHash(string password, string hashedPassword)
- public bool IsUserAuthenticated(string username, string password)
- public HttpCookie AddCookie(User user)
- public void DefinePassword(User editedUser)
- public void ChangePassword(User currentUser, string password)

Hashpassword

Returns an RFC 2898 hash value for the specified password.

Parameters:

Password Type:*System.String*
The password to generate a hash value for.

Return Value Type:*System.String*
The hash value for password as a base-64-encoded string.

The password hash is generated with the RFC 2898 algorithm using a 128-bit salt, a 256-bit subkey, and 1000 iterations. The format of the generated hash bytestream is {0x00, salt, subkey}, which is base-64 encoded before it is returned.

GetHashedPassword

Overloaded methods both returns hashed password for the specified user, by Username or UserID.

Parameters

Username Type:*System.String*

UserID Type:*System.Int32*

Return Value Type:*System.String*
Return hashed password from database for user with specified parameters

IsPasswordMatchWithHash

Determines whether the specified RFC 2898 hash and password are a cryptographic match.

Parameters:

HashedPassword Type:*System.String*
The previously-computed RFC 2898 hash value as a base-64-encoded string.

Password Type:*System.String*
The plaintext password to cryptographically compare with hashedPassword.

Return Value Type:*System.Boolean*
True if the hash value is a cryptographic match for the password; otherwise, false.

IsUserAuthenticated

Determines wheter the specified user with username and password exists in database

Parameters:

username, password Type:*System.String*
The username and password for specified user

Return Value Type:*System.Boolean*
True if the hash value is a cryptographic match for the password; otherwise, false.

AddCookie

Method used for creating cookie with user defined data.

Parameters:

User Type:User User class

Return Value Type:*System.Web.HttpCookie*

DefinePassword

Method used for creating password for user.

Parameters:

editedUser Type:User User class

Return Value Type: void

Collect password user provided in view, generate hash password from input password, activate inactive user, and save edited user data into database.

ChangePassword

Method that provides changing of current user password.

Parameters:

CurrentUser Type:User User class

Password Type:*System.String*
Password for current user

Return Value Type: void

Collect password current user provided in view, generate hash password from input password, activate inactive user, and save edited user data into database.

User service

User service is service used for managing basic CRUD operations for User model, and manipulating with user data in database. It depends on VacationProject.DAL and VacationProject.Model.

List of methods:

public int GetVacationDaysForUser(string username)

Method that returns remaining vacation days for user with specified username

public List<User> GetAllAdmins()

Method that returns list of users with administrative privileges

public List<User> GetListOfSelectedManagers(ValueProviderResult userName)

Method that returns list of selected administrators for approving vacation request

public User GetUserByUsername(string username)

Method that returns User with specified username

public User GetUserByID(int UserID)

Method that returns User with specified ID

public User GetActiveUserByUsername(string username)

Method that returns active User with specified username

public string[] GetUserRoles(User user)

Method that returns array of roles for the specified User

public List<VacationApproving> Initiliaze(int id)

Method that returns list of VacationApproving that is sent to administrator for approval. Input parameter is approver id.

public List<VacationApproving> GetAprovingsbyName(string name)

Method that returns list of VacationApproving that is sent to administrator for approval. Input parameter is approver username

public List<Role> GetAllRoles()

Method that returns list of roles from database

public void CreateUser(User user,FormCollection form)

Method that create new user in database. Input parameters are user and form fields with user information.

public void SaveUser(User user,User editedUser)

Method that save edited user

public List<User> GetAllUsers()

Method that returns list of all active users including user roles from database

public User AddDays(int id)

Method that add 20 days to

public void DeleteUser(User user)

With this method we set user status to inactive

public bool IsEmailInUse(string email)

Method that returns true if email is in use, otherwise returns false

public User GetUserByEmail(string email)

Method that returns User with specified email

public void makeAllRequestsInvalid(User user)

Method that change status of vacation requests to invalid for the specified user

public bool IsUsernameInUse(string username)

Method that returns true if username is in use, otherwise return false.

VacationApprovingService

VacationApprovingService have one method GetApprovingsForApprover.

public List<VacationApproving> GetApprovingsForApprover(User user)

The method returns list of VacationApproving entities, which one(or only) approver is user that is forwarded as parametar.

EmailServices

EmailService provide sending email messages. It contains four method(two of them are overloaded): VerificationMail(User user), VerificationMail(User user, int userID), OutcomeOfRequest(VacatioRequest vacatioRequest) and EncryptString(string username).

EncryptString(string username)

Get username and **return** its encrypted value as **string**. Its used in VerificationMail because of better security.

OutcomeOfRequest(VacationRequest vacationRequest)

Get vacationRequest entity as parameter and send mail to one that requires vacation its outcome, and cc of mail are all of managers – approvers. **Returns void.**

VerificationMail(User user)

This method is used when ForgotPassword action is called. Method get user entity as parameter and send him/her mail with link where he/she can reset password. **Returns void.**

VerificationMail(User user, int userId)

This method is used when new user is created. Method get user entity and new userId as parameter. Then its send mail to him/her with link where he/she can define new passoword. **Returns void.**

VacationRequestService

This service use for all CRUD operation for VacatioRequest entity. Contains nine methods: **GetAll(), GetAllRequestsForCurrentUser(string username), CalculateDays (VacatioRequest vacationRequest), SendRequestToManagers (VacationRequest vacationRequest, List<User> listOfManagers), CreatingOfRequest (VacationRequest vacationRequest, User applicant, int workingDays), FindVacationRequestByID(int id), CheckRemainingDays (VacationRequest vacationRequest, string username), AproveRequest(int id), RejectingRequest(int id, string name)**.

GetAll()

This method is used when we need all entities of VacationRequest. Returns List<VacatioRequest>.

GetAllRequestForCurrentUser(string username)

This method is used when we need entites of VacationRequest for user with given *username*. *Returns* List<VacationRequest>.

CalculateDays(VacationRequest vacationRequest)

This method is used for calculating working days. Get VacationRequest entity which contains BeginDate and EndDate. Returns int.

SendRequestToManagers (VacationRequest vacationRequest, List <User> listOfManagers)

This method method insert VacationApproving entities in data base for every user that is in listOfManagers. Returns void.

CreatingOfRequest (VacationRequest vacationRequest, User applicant, int workingDays)

This method write new entity into data base. Return void.

FindVacationRequestByID(int id)

This method find specific VacationRequest entity form data base with id that is forwarded as parameter. Returs VacationRequest.

CheckRemainingDays (VacationRequest vacationRequest, string username)

This method returns value of enum in case of result of calculation. Enum name is *StatusOfCountingDays* and have four values: *Valid, Invalid, LessDays and Yesterday. Yesterday* is got when one of days (begin or end date) is already past, *Invalid* when End date is before Begin Date, *LessDays* when entered number of days is bigger than day that user have right on, and *Valid* when everything is ok.

AproveRequest(int id)

This method used when one of (or only one) managers approves request. Parameter id is marks specific entity in data base. There is check if some of managers are waiting to approve or reject request. If that is situation, state of request stays *Panding,* otherwise it becomes *Approved.* Returns void.

RejectRequest(int id, stiring name)

This method used when one of (or only one) managers rejects request. User entity who approve is get by name parameter, and VacationRequest entity by id parameter. State of VacationRequest is set on Reject, and days that were "reserved" from applicant "gets back" to him. Return void.

Exceptions

SomethingWentWrong

This exception is located in VacationProject.Service. This is universal exception that is throw whenever appear problem with data base. Constructor overrides message of this exception.

VacationProject

Controllers

Asp.net MVC Controllers are responsible for controlling the flow of the application execution. When you make a request (means request a page) to MVC application, a controller is responsible for returning the response to that request. The controller can perform one or more actions. The controller action can return different types of action results to a particular request.

The Controller is responsible for controlling the application logic and acts as the coordinator between the View and the Model. The Controller receives an input from the users via the View, then processes the user's data with the help of Model and passes the results back to the View.

Vacation Project application logic is organized in five controllers:

- BaseController.cs
- AccountController.cs
- UsersController.cs
- VacationRequestsController.cs
- ErrorController.cs

Base Controller for accessing Current User

First a base controller has been created for accessing User data in all controllers, and Inherit all controllers from this base controller to access user information from the UserContext.

```
protected virtual new CustomPrincipal User
        {
            get { return HttpContext.User as CustomPrincipal; }
        }
public class AccountController : BaseController
```

Account Controller

Account controller is responsible for security and authorization logic of Vacation Project application. Account controller in constructor create new instance of UserService, AccountService and EmailService.

To proccess user's input data from Views, following Actions have been implemented:

```
public ActionResult Index()
```

In this method, first check is whether the user is authenticated, and depending on this redirects the user to the user or admin page. If user is not authenticated, Index view (Login page) from Account controller is called.

```
[HttpPost]
public ActionResult Index(LoginViewModel model, string returnUrl = "")
```

In HttpPost method of Index controller action, LoginViewModel has been passed from Index view as input parameter, with all information needed to authenticate current user. With UserService, it has been checked if User with username passed from LoginViewModel exists in database. If user exists, another validation has been provided, with AccountService user with specified username and password authentication has been checked, and if true, cookie with user defined data is added, and depending on user roles users have been redirected to their pages.

```
public ActionResult LogOut()
```

Action used to log out user from Vacation Project application.

```
public ActionResult AccountRecovery()
[HttpPost]
```

```
 public ActionResult AccountRecovery(RecoveryPasswordViewModel
 model)
```

If user forget the password, account recovery action allows the users to reset their password. First, customer service is called to check whether the user has entered the email that exists in the database. If the user is identified, application send him verification email with the necessary information to change the password and performs the redirect to Login page.

```
 [HttpGet]
 public ActionResult DefinePassword(string code)
```

In get method of DefinePassword action parameter code is passed. Code parameter represents encrypted username of current user. With DecryptString method encrypted username is decrypted and passed to GetUserByName method of UserService to get user information.

```
 [HttpPost]
 public ActionResult DefinePassword
([Bind(Include="UserID,Password,RepeatPassword")] DefinePasswordViewModel
definePasswordViewModel)
```

When an administrator creates a new user, the user is automatically deactivated and receive via email a link to DefinePassword page to enter a new password. Action DefinePassword allows the user to set a password and after setting new password user automatically becomes active user of application. Input parameter of DefinePassword action is DefinePasswordViewModel.

```
 [HttpGet]
 public ActionResult ChangePassword()
 [HttpPost]
 public ActionResult ChangePassword(ChangePasswordViewModel model)
```

ChangePassword action allows user to change current password. ChangePasswordViewModel is passed as parameter to ChangePassword action method to provide information about current and new password.

In this action AccountService is called to check if current password hash matches with new password that user entered. If statement is true, AccountService method ChangePassword is called to set new password to user.

List of methods:

```
[HttpPost]
public bool CheckPassword(string oldPass)
```

Check password method returns true if old password matches with hashedpassword from database.

```
public string DecryptString(string encrString)
```

Decrypt string method returns decrypted string of encrString paramether.

Users Controller

Users controller is responsible for CRUD logic for user. Users controller in constructor create new instance of UserService, AccountService and VacationService.

List of actions:

public ActionResult Index()

Action used to pass to Index View list of vacation approvings pending for approval.

[CustomAuthorize(RolesConfigKey = "RolesConfigKey")]

public ActionResult Create()

[HttpPost]

[ValidateAntiForgeryToken]

[CustomAuthorize(RolesConfigKey = "RolesConfigKey")]

public ActionResult Create([Bind(Include = "UserId,Username,Email,Password,FirstName,LastName,IsActive,CreateDate,VacationDays")] User user, FormCollection form)

Action that is secured with CustomAuthorize attribute. Only administrators are allowed to create new users.

public ActionResult Edit(int? id)

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "UserId,Username,Email,FirstName,LastName,VacationDays")] User editedUser)

Action that is used to edit user information and save it to database.

[CustomAuthorize(RolesConfigKey = "RolesConfigKey")]

public ActionResult Delete(int? id)
[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)
Delete and DeleteConfirmed actions are secured with CustomAuthorizeAttribute class, and only application administrators can disable active user.

When user performs Delete and DeleteConfirmed action in UsersController, user is not deleted from database, only user status is changed to inactive.

[CustomAuthorize(RolesConfigKey = "RolesConfigKey")]

public ActionResult ListOfUsers()
Action ListOfUsers is secured with CustomAuthorizeAttribute class, and only application administrators can see list of active users.

Action ListOfUsers call UserService to retrieve list of all active users.

[CustomAuthorize(RolesConfigKey = "RolesConfigKey")]

public ActionResult DetailsOfUsers(int? id)
Action DetailsOfUsers is secured with CustomAuthorizeAttribute class, and only application administrators can see details for specified user. Id of user is passed to controller action and GetUserByID method from UserService is called to get details for specified user.

Methods:

public ActionResult AddDays(int? id)
Method for adding days to remaining vacation days.

[HttpPost]

public void Approvig(int? id)
In this method application call VacationService to approve pending vacation request.

[HttpPost]

public void RejectingOFRequest(int? id)
In this method application call VacationService to reject pending vacation request.

public bool UserIsAuthenticated()
Check if user is active and authenticated.

[HttpPost]

public bool CheckUsername(string username)
Method returns true if user with specified username exists in database, otherwise false.

[HttpPost]

public bool CheckEmail(string email)
Method returns true if user with specified email exists in database, otherwise false.

Vacation Requests Controller

Vacation Requests Controller is responsible for manipulating with Vacation Requests data. Vacation Requests Controller in constructor create new instance of UserService and VacationRequestsService.

List of methods:

public ActionResult Index()
Index action send list of Vacation requests to Index view of VacationRequests controller.

public ActionResult Create()
[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "ID,BeginDate,EndDate")] VacationRequest vacationRequest, FormCollection form)
Create Action is responsible for creating new Vacation Request. This action call private Action CreationOfRequest.

private ActionResult CreationOfRequest(VacationRequest vacationRequest, ValueProviderResult usernamesOfApprovers)

private ActionResult SendAndSaveRequest(VacationRequest vacationRequest, List<User> approvers, User currentUser, int workingDays)

public StatusOfCountingDays CheckRemainingDays(VacationRequest vacationRequest)

Views

For every controller in VacationProject, and for most of methods there is specific view. Here is a list of controllers and its views.

- AccountView
  - AccountRecovery
  - ChangePassword
  - DefinePassword
  - Index
  - Error
- ErrorPage
  - Shared
- _Layout
  - Users
- Create
- Delete
- DetailsOfUser
- Edit
- Index
- ListOfUsers
  - VacationRequest
- Create
- Index

Account

AccountRecovery

It's connected to AccountRecovery method in AcoountController, and it use RecoveryPasswordViewModel

ChangePassword

It's connected to ChangePassword method in AcoountController, and it use ChangePasswordViewModel

DefinePassword

It's connected to DefinePassword method in AcoountController, and it use DefinePasswordViewModel

I

It's connected to Index method in AcoountController which represent Login of user, and it use LoginViewModel

ErrorPage

It's connected to ErrorPage method in ErrorControlle and it use ErrorViewModel

Shared

_Layout

It's represent header and footer of View.

Users

Create

It's represent form for creating of user. It's connected to Create method of UserController, and it use User model.

D

It's represent form for deleting of user. It's connected to Delete method of UserController, and it use User model.

DetailsOf

It's represent form for reading values of user. It's connected to DetailsOfUser method of UserController, and it use User model.

Edit

It's represent form for editing values of user. It's connected to Edit method of UserController, and it use User model.

I

It's represent main page for admins. It's connected to Index method of UserController, and it use IEnumerable<VacationApproving>.

ListOfUsers

It's represent list of users page for admins. It's connected to ListOfUsers method of UserController, and it use Ienumerable<User>.

VacatioRequest

Create

It's represent form for creating of VacationRequest. It's connected to Create method of VacationRequestController, and it use VacationRequest model.

I

It's represent main page for users. It's connected to Index method of VacationRequestController, and it use Ienumerable<VacationRequest>.

Exceptions

In VacationProject application are implements some custom exceptions. Here is list of them:

- AccessDeiedException
- EmailExistException
- EnterNonPositiveNumberException
- ForbidenActionException
- FormNotValidExcpetion
- LessWorkingDaysException
- NotFoundException
- NotSelectedMenagersException
- NotSelectedRoleException
- UsernameInUseException
- SomethingWentWrongException

AccessDeniedException

This exception is located in VacationProject. It use mostly when someone try to do some action for which he/she is not authorize. Constructor overrides message of this exception.

EmailExistException

This exception is located in VacationProject. It use only in Create method in UserController when entered email for new user already exist. Constructor overrides message of this exception.

EnterNonPositiveNumberException

This exception is located in VacationProject. It use only in Create method in UserController when entered vacation days number is negative. Constructor overrides message of this exception.

ForbidenActionException

This exception is located in VacationProject. It use mostly when someone try to do some action for which he/she is not allowed to do. Constructor overrides message of this exception.

FormNotValidException

This exception is located in VacationProject. It use mostly when someone fill form on not valid way, or some of fields in for are not valid filled. Constructor overrides message of this exception.

LessWorkingDaysException

This exception is located in VacationProject. It only use in Create method in VacationRequestContoller, when someone ask for more vacation days than he/she have. Constructor overrides message of this exception.

NotFoundException

This exception is located in VacationProject. It use mostly when someone call methods without parametars or when data base return null. Constructor overrides message of this exception.

NotSelectedMenagersException

This exception is located in VacationProject. It only use in Create method in VacationRequestContoller, when someone try to create vacation request but not select managers for approve of that. Constructor overrides message of this exception.

NotSelectedRoleException

This exception is located in VacationProject. It only use in Create method in UserContoller, when someone try to create user but not select role for he/she. Constructor overrides message of this exception.

UsernameInUseException

This exception is located in VacationProject. It only use in Create method in UserContoller, when someone try to create user but entered username alredy exist. Constructor overrides message of this exception.

SomethingWentWrong

This exception is located in VacationProject.Service. This is universal exception that is throw whenever appear problem with data base. Constructor overrides message of this exception.

ViewModel

In VacationProject are used four ViewModels. They are AccountViewModel, BaseViewModel, DefinePasswordViewModel, ErrorViewModel.

AccountViewModel

This class contains three inner classes.

L

This view model is used in Index method in AccountController. Index method is login method. This model contains three properties: string, Username, string Password, bool RememberMe

Rec

This view model is used in AccountRecovery method in Account Controller. AccountRecovery is Forgot password functionality. Model contains one property: *string Email*.

ChangePa

This view model is used in ChangePassword method in AccountControler. Contains three property: *string OldPassword, string Password, string ConfirmPassword*...

ErrorViewModel

This ViewModel i*s general viewModel for all exceptions.* It have only one property: *string Message, and* it have usage in BaseContollerin OnExcpetion method.

DefinePasswordViewModel

This view model is used in DefinePassword method in AccountControler. Contains three property: string OldPassword, string Password, string ConfirmPassword...

Base View Page for accessing Current User

To access User data in all views, we created new abstract class BaseViewPage that implements WebViewPage class as shown below:

```
public abstract class BaseViewPage:WebViewPage
    {
        public virtual new CustomPrincipal User
        {
            get { return base.User as CustomPrincipal; }
        }
    }
    public abstract class BaseViewPage<TModel> : WebViewPage<TModel>
    {
        public virtual new CustomPrincipal User
        {
            get { return base.User as CustomPrincipal; }
        }
    }
```

Register this class with in the \Views\Web.config as base class for all views as given below:

```
<system.web.webPages.razor>
   <pages pageBaseType="VacationProject.ViewModel.BaseViewPage">
```

Front-end

The VacationProject application contains client-side implementation, which is done through JS libraries.

List of library that are used in application:

- _reference.js
- bootstrap-datetimepicker.js
- bootstrap-datetimepicker.min.js
- bootstrap.js
- bootstrap.min.js
- ddtf.js
- jquery.js
- jquery-1.12.4.js
- jquery-1.12.4.min.js
- jquery-3.1.1.js
- jquery-3.1.1.min.js
- jquery.validate.js
- jquery.validate.unobtusive.js
- modernizr-2.6.2.js
- moment.js
- moment.min.js
- respond.js
- respond.min.js
- SingUp.js

Custom library that are used in application:

- adminApprovingJS.js
- changePassword.js
- userCreating.js
- vacationRequest.js

adminApproving.js

This library is used in Index view of VacationRequestController, and contains two methods: ***ApprovingFunc()*** and ***RejectingFunc()***.

ApprovingFunc()

This method receives one parametar **<u>id</u>**, which represent id of VacationRequets that is approved. That id is send via ajax *<u>post</u>* call on "<u>/Users/Approvig/id</u>".

RejectingFunc()

This method receives one parametar **<u>id</u>**, which represent id of VacationRequets that is approved. That id is send via ajax *<u>post</u>* call on "<u>/Users/RejectingOFReques/id</u>".

changePassword.js

This library is used in <u>DefinePassword and ChangePassword view of AccountController</u>, and contains three methods: ***CheckOldPassword(), IsRepeatedPassEqualsToNew()*** *and* ***IsSame()***;

CheckOldPassword()

This method is called when onblur event is happened. Onblur is connected to OldPassword field in ChangePassword.cshtml. Event triggers displaying of loader, than method get entered value from OldPassword field, and via ajax post call on <u>"/Account/CheckPassword"</u> send value of OldPassword. Controller returns boolean result, and depending on the result, success function in ajax can do two things. If result equals to *true*, submit button becomes enable and border of input becomes green, in other case, submit button is disable and border is red. Complete function of ajax hides loader.

IsRepeatedPassEqualsToNew()

This methodis called when onblur event is happened. Onblur is connected to ConfirmPassword field in ChangePassword.cshtml. Method get values from

Password and ConfirmPassword fields and compare them. If equals border color of ConfirmPassword field becoumes green, in other case red.

IsSame()

This method is called when onblur event is happened. Onblur is connected to ConfirmPassword field in DefinePassword.cshtml. Method get values form Password and ConfirmPasswordFields and compare them. If equals border color of ConfirmPassword filed becomer green, in other case red and error message displays. Error message is simple <p> which initial state of display is none.

userCreating.js

This library is used in Create view of UserController and contains three methods: ***checkUsername(), checkEmail()*** and ***checkVacationDays().***

CheckUsername()

This method is called when onblur event is happened. Onblur is connected to Username field in Create.cshtml. Event triggers get of entered value from Username field, and via ajax post call on "/Users/CheckUsername?username=value" send value of Username. Controller returns boolean result, and depending on the result, success function in ajax can do two things. If result equals to *true*, submit button becomes enable and border of input becomes green, in other case, submit button is disable and border is red and error message shows. Error message is simple <p> which initial state of display is none.

CheckEmail()

This method is called when onblur event is happened. Onblur is connected to Email field in Create.cshtml. Event triggers get of entered value from Username field, and via ajax post call on "/Users/CheckEmail?email=value" send value of Email. Controller returns boolean result, and depending on the result, success function in ajax can do two things. If result equals to *true*, submit button becomes enable and border of input becomes green, in other case, submit button is disable and border is red and error message shows. Error message is simple <p> which initial state of display is none.

CheckVacationDays()

This method is called when onblur event is happened. Onblur is connected to VacationDays field in Create.cshtml. Event triggers get of entered value from Username field, and check if entered number is less than 0. If result equals to *false*, submit button becomes enable and border of input becomes green, in other case, submit button is disable and border is red and error message shows.

Error message is <p> which initial state of display is none.

vacationRequest.js

This library is used in Create view of VacationRequestController and contains one method: ***checkDays().***

CheckDays()

This metod is called when onblur event is happened. Onblur is connected to *BeginDate* and *EndDate* fields in Create.cshtml. Event triggers get of entered value from BeginDate and EndDate fields, and via ajax post call on "/VacationRequest/CheckRemainingDays" send object. Controller returns enum , and depending on the value, success function in ajax can do four things. If result equals to *"Yesterday"*,"LessDays" or "Invalid" submit button becomes disable and error message shows, in other case submit button becomes enable and error message hides. Error messages are simple <p> which initial state of display is none.