# ПРИЛОЖЕНИЕ А

Листинг основных программных модулей

/// <summary>

/// Work with files

/// </summary>

/// <param name="imageService"></param>

/// <param name="veteranService"></param>

/// <param name="userManager"></param>

public FilesController(IImageService imageService, VeteranService veteranService, IUserService userService)

{

\_imageService = imageService;

\_veteranService = veteranService;

\_userService = userService;

}

/// <summary>

/// Get report excell file

/// </summary>

/// <param name="searchVeteranModel"></param>

/// <returns></returns>

[Route("api/files")]

[AllowAnonymous]

public IHttpActionResult GetReportExcellFiles([FromUri]SearchVeteranModel searchVeteranModel)

{

List<Veteran> searchVeterans;

if (searchVeteranModel == null)

{

searchVeterans = \_veteranService.GetAll().ToList();

}

else

{

searchVeterans = \_veteranService.SearchVeterans(searchVeteranModel).Pagination((searchVeteranModel.Page - 1) \* searchVeteranModel.Size, searchVeteranModel.Size).ToList(); ;

}

var veteranMappings = Mapper.Map<IEnumerable<Veteran>, IEnumerable<VeteranMapping>>(searchVeterans);

var fileName = ExcellParser.GenerateReport(veteranMappings.ToList());

var generateAbsolutePath = GenerateAbsolutePath(fileName);

return Ok(new

{

PathToFile = generateAbsolutePath

});

}

/// <summary>

/// Upload a image files

/// </summary>

/// <returns></returns>

[Route("api/files")]

public async Task<IHttpActionResult> Post()

{

if (!Request.Content.IsMimeMultipartContent())

{

return BadRequest();

}

string root = HttpContext.Current.Server.MapPath("~/Content/Files/");

var provider = new MultipartMemoryStreamProvider();

await Request.Content.ReadAsMultipartAsync(provider);

var imageUrl = GenerateAbsolutePath(HttpContext.Current.Request.ApplicationPath + ConfigurationSettingsModule.GetItem("PathImages"));

if (!Directory.Exists(root))

{

Directory.CreateDirectory(root);

}

Dictionary<string, string> errors = new Dictionary<string, string>();

List<ImageReference> imageFilesVeterans = \_imageService.SaveImages(provider, root, ref errors);

var imageVeterans = imageFilesVeterans.Select(file => new ImageReference

{

ImageOriginal = imageUrl + @"/" + file.ImageOriginal,

ThumbnailImage = imageUrl + @"/" + file.ThumbnailImage

}).ToList();

return Ok(new

{

Images = imageVeterans,

Errors = errors

});

}

/// <summary>

/// Upload excell file on server

/// </summary>

/// <returns></returns>

/// <exception cref="HttpResponseException"></exception>

[Route("api/files/uploadExcell")]

public async Task<IHttpActionResult> PostUploadExcellFiles()

{

if (!Request.Content.IsMimeMultipartContent())

{

return BadRequest();

}

var provider = new MultipartMemoryStreamProvider();

string path = null;

Tuple<byte[], HttpContent> fileArrayAndFileHttpContent = await GetFileArrayAndFileHttpContentFromProvider(provider);

var filename = GetFilename(fileArrayAndFileHttpContent.Item2);

filename += Guid.NewGuid() + ".xlsx";

path = Path.Combine(HttpContext.Current.Server.MapPath("~" + ConfigurationSettingsModule.GetItem("Temp")), filename);

if (!Directory.Exists(HttpContext.Current.Server.MapPath("~" + ConfigurationSettingsModule.GetItem("Temp"))))

{

Directory.CreateDirectory(

HttpContext.Current.Server.MapPath("~" + ConfigurationSettingsModule.GetItem("Temp")));

}

using (FileStream fs = new FileStream(path, FileMode.Create))

{

fs.Write(fileArrayAndFileHttpContent.Item1, 0, fileArrayAndFileHttpContent.Item1.Length);

}

try

{

ExcellParser excellParser = new ExcellParser(path);

var veteranMappings = excellParser.GetVeterans();

foreach (var veteranMapping in veteranMappings)

{

var veteranBindingModel = Mapper.Map<VeteranMapping, VeteranBindingModel>(veteranMapping);

var listParsedUrls = UrlParser.Parse(veteranMapping.UrlImages);

veteranBindingModel.Images = UrlParser.DownloadFromUrls(listParsedUrls);

var veteran = Mapper.Map<VeteranBindingModel, Veteran>(veteranBindingModel);

veteran.User = \_userService.GetById(User.Identity.GetUserId());

var googleMapsService = new GoogleMapsService(string.Empty);

var latLng = googleMapsService.GetLatLng(veteran.BirthPlace);

veteran.Latitude = latLng.Latitude;

veteran.Longitude = latLng.Longitude;

\_veteranService.Add(veteran);

}

}

catch (Exception exception)

{

var resp = new HttpResponseMessage(HttpStatusCode.InternalServerError)

{

Content = new StringContent("Problems with parsing the file"),

ReasonPhrase = "File Not Parsed"

};

\_logger.Error(exception);

throw new HttpResponseException(resp);

}

finally

{

if (File.Exists(path))

{

File.Delete(path);

}

}

return Ok();

}

/// <summary>

/// Crop a image by coordinates

/// x,y,widh,heigh

/// </summary>

/// <returns></returns>

[Route("api/files/cropImage")]

public async Task<IHttpActionResult> CropImage([FromUri]SelectionImageBindingModel selectionImage)

{

var source = new Bitmap(HostingEnvironment.MapPath(selectionImage.UrlImage.ToRelativePath()));

Rectangle section = new Rectangle(new Point(selectionImage.X, selectionImage.Y), new Size(selectionImage.Width, selectionImage.Height));

var cropImage = \_imageService.CropImage(source, section);

var imageReference = \_imageService.SaveImage(cropImage);

return Ok(imageReference);

}

[System.Web.Http.NonAction]

public string GenerateAbsolutePath(string virtualPath)

{

return HttpContext.Current.Request.Url.Scheme +

"://"

+ HttpContext.Current.Request.Url.Authority + virtualPath;

}

private static string GetFilename(HttpContent file)

{

var filename = file.Headers.ContentDisposition.FileName.Replace("\"", string.Empty);

return filename;

}

private async Task<Tuple<byte[], HttpContent>> GetFileArrayAndFileHttpContentFromProvider(MultipartMemoryStreamProvider provider)

{

await Request.Content.ReadAsMultipartAsync(provider);

var file = provider.Contents[0];

byte[] fileArray = file.ReadAsByteArrayAsync().Result;

return new Tuple<byte[], HttpContent>(fileArray, file);

}

}

[Authorize(Roles = "User")]

public class VeteranController : ApiController

{

private readonly IVeteranService \_veteranService;

private readonly IUserService \_userService;

private readonly IImageVeteranService \_imageVeteranService;

public VeteranController(IVeteranService veteranService, IUserService userService, IImageVeteranService imageVeteranService)

{

\_veteranService = veteranService;

\_userService = userService;

\_imageVeteranService = imageVeteranService;

}

/// <summary>

/// Get a veteran by Id

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

[Route("api/veteran/{id}")]

[ResponseType(typeof(VeteranViewModel))]

[AllowAnonymous]

public IHttpActionResult Get(int id)

{

var veteran = \_veteranService.GetById(id);

if (veteran == null)

{

return NotFound();

}

veteran.Views++;

\_veteranService.SaveArticle();

var veteranViewModel = Mapper.Map<Veteran, VeteranViewModel>(veteran);

return Ok(veteranViewModel);

}

/// <summary>

/// Get veterans by conditional or get all veterans

/// </summary>

/// <param name="searchVeteranModel"></param>

/// <returns></returns>

[Route("api/veteran")]

[ResponseType(typeof(VeteranViewModel))]

[AllowAnonymous]

public IHttpActionResult Get([FromUri]SearchVeteranModel searchVeteranModel)

{

IEnumerable<Veteran> veterans = null;

int allCount = 0;

if (searchVeteranModel == null)

{

veterans = \_veteranService.GetAll();

allCount = \_veteranService.GetAll().Count();

}

else

{

allCount = \_veteranService.SearchVeterans(searchVeteranModel).Count();

veterans = \_veteranService.SearchVeterans(searchVeteranModel).Pagination((searchVeteranModel.Page - 1) \* searchVeteranModel.Size, searchVeteranModel.Size).ToList();

}

var veteranBindingModels = Mapper.Map<IEnumerable<Veteran>, IEnumerable<VeteranViewModel>>(veterans);

return Ok(new

{

Items = veteranBindingModels,

TotalCount = allCount

});

}

/// <summary>

/// Add a veteran

/// </summary>

/// <param name="veteranBindingModel"></param>

/// <returns></returns>

[Route("api/veteran")]

[ResponseType(typeof(VeteranViewModel))]

public IHttpActionResult Post(VeteranBindingModel veteranBindingModel)

{

if (!ModelState.IsValid) return BadRequest(ModelState);

Veteran veteran = Mapper.Map<VeteranBindingModel, Veteran>(veteranBindingModel);

var userId = User.Identity.GetUserId();

if (userId == null)

{

return BadRequest();

}

var user = \_userService.GetById(userId);

veteran.User = user;

\_veteranService.Add(veteran);

var veteranViewModel = Mapper.Map<Veteran, VeteranViewModel>(veteran);

return Ok(veteranViewModel);

}

/// <summary>

/// Update a veteran

/// </summary>

/// <param name="veteranBindingModel"></param>

/// <returns></returns>

[Route("api/veteran")]

[ResponseType(typeof(VeteranViewModel))]

public IHttpActionResult Put([FromBody]VeteranBindingModel veteranBindingModel)

{

var veteran = \_veteranService.GetById(veteranBindingModel.Id);

var userId = User.Identity.GetUserId();

if (ModelState.IsValid && veteranBindingModel.Id == veteran.Id && userId == veteran.User.Id)

{

\_imageVeteranService.DeleteImages(veteran.Images);

Veteran mapVeteran = Mapper.Map<VeteranBindingModel, Veteran>(veteranBindingModel);

Mapper.Map<Veteran, Veteran>(mapVeteran, veteran);

\_veteranService.UpdateVeteran(veteran);

var veteranModified = Mapper.Map<Veteran, VeteranViewModel>(veteran);

return Ok(veteranModified);

}

return StatusCode(HttpStatusCode.NotModified);

}

/// <summary>

/// Delete a veteran by Id

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

[Route("api/veteran/{id}")]

[HttpDelete]

public IHttpActionResult Delete(int id)

{

var veteran = \_veteranService.GetById(id);

var userId = User.Identity.GetUserId();

if (veteran == null || veteran.Id != id || veteran.User.Id != userId) return BadRequest();

veteran.IsDeleted = true;

\_veteranService.SaveArticle();

return Ok();

}

}

[Authorize]

[RoutePrefix("api/Account")]

public class AccountController : BaseController

{

private readonly IUserService \_userService;

private const string LocalLoginProvider = "Local";

public AccountController(ApplicationUserManager userManager, IUserService userService)

{

\_userService = userService;

UserManager = userManager;

}

public ApplicationUserManager UserManager

{

get

{

var applicationUserManager = \_userManager ?? Request.GetOwinContext().GetUserManager<ApplicationUserManager>();

return applicationUserManager;

}

private set

{

\_userManager = value;

}

}

public ISecureDataFormat<AuthenticationTicket> AccessTokenFormat { get; private set; }

// GET api/Account/UserInfo

/// <summary>

/// Get user info

/// </summary>

/// <returns></returns>

[HostAuthentication(DefaultAuthenticationTypes.ExternalBearer)]

[Route("UserInfo")]

public UserInfoViewModel GetUserInfo()

{

ExternalLoginData externalLogin = ExternalLoginData.FromIdentity(User.Identity as ClaimsIdentity);

var roleUser = UserManager.GetRoles(User.Identity.GetUserId());

var user = \_userService.GetById(User.Identity.GetUserId());

return new UserInfoViewModel

{

Id = user.Id,

Email = user.Email,

Role = roleUser[0],

HasRegistered = externalLogin == null,

LoginProvider = externalLogin?.LoginProvider,

Image = user.Image.ToAbsolutPath(),

FirstName = user.FirstName,

LastName = user.LastName

};

}

// POST api/Account/Logout

/// <summary>

/// Logout a user

/// </summary>

/// <returns></returns>

[Route("Logout")]

public IHttpActionResult Logout()

{

Authentication.SignOut(CookieAuthenticationDefaults.AuthenticationType);

return Ok();

}

[Route("ManageInfo")]

public async Task<ManageInfoViewModel> GetManageInfo(string returnUrl, bool generateState = false)

{

IdentityUser user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user == null)

{

return null;

}

List<UserLoginInfoViewModel> logins = new List<UserLoginInfoViewModel>();

foreach (IdentityUserLogin linkedAccount in user.Logins)

{

logins.Add(new UserLoginInfoViewModel

{

LoginProvider = linkedAccount.LoginProvider,

ProviderKey = linkedAccount.ProviderKey

});

}

if (user.PasswordHash != null)

{

logins.Add(new UserLoginInfoViewModel

{

LoginProvider = LocalLoginProvider,

ProviderKey = user.UserName,

});

}

return new ManageInfoViewModel

{

LocalLoginProvider = LocalLoginProvider,

Email = user.UserName,

Logins = logins,

ExternalLoginProviders = GetExternalLogins(returnUrl, generateState)

};

}

/// <summary>

/// Change password a user

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

[Route("ChangePassword")]

public async Task<IHttpActionResult> ChangePassword(ChangePasswordBindingModel model)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

IdentityResult result = await UserManager.ChangePasswordAsync(User.Identity.GetUserId(), model.OldPassword,

model.NewPassword);

if (!result.Succeeded)

{

return GetErrorResult(result);

}

return Ok();

}

[Route("UpdateProfile")]

public IHttpActionResult UpdateProfile(UserInfoBindingModel userInfo)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var user = \_userService.GetById(User.Identity.GetUserId());

Mapper.Map<UserInfoBindingModel, User>(userInfo, user);

\_userService.UpdateUser(user);

return Ok();

}

/// <summary>

/// Set password

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

[Route("SetPassword")]

public async Task<IHttpActionResult> SetPassword(SetPasswordBindingModel model)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

IdentityResult result = await UserManager.AddPasswordAsync(User.Identity.GetUserId(), model.NewPassword);

if (!result.Succeeded)

{

return GetErrorResult(result);

}

return Ok();

}

/// <summary>

/// Add external login

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

[Route("AddExternalLogin")]

public async Task<IHttpActionResult> AddExternalLogin(AddExternalLoginBindingModel model)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

Authentication.SignOut(DefaultAuthenticationTypes.ExternalCookie);

AuthenticationTicket ticket = AccessTokenFormat.Unprotect(model.ExternalAccessToken);

if (ticket == null || ticket.Identity == null || (ticket.Properties != null

&& ticket.Properties.ExpiresUtc.HasValue

&& ticket.Properties.ExpiresUtc.Value < DateTimeOffset.UtcNow))

{

return BadRequest("External login failure.");

}

ExternalLoginData externalData = ExternalLoginData.FromIdentity(ticket.Identity);

if (externalData == null)

{

return BadRequest("The external login is already associated with an account.");

}

IdentityResult result = await UserManager.AddLoginAsync(User.Identity.GetUserId(),

new UserLoginInfo(externalData.LoginProvider, externalData.ProviderKey));

if (!result.Succeeded)

{

return GetErrorResult(result);

}

return Ok();

}

/// <summary>

/// Remove login

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

[Route("RemoveLogin")]

public async Task<IHttpActionResult> RemoveLogin(RemoveLoginBindingModel model)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

IdentityResult result;

if (model.LoginProvider == LocalLoginProvider)

{

result = await UserManager.RemovePasswordAsync(User.Identity.GetUserId());

}

else

{

result = await UserManager.RemoveLoginAsync(User.Identity.GetUserId(),

new UserLoginInfo(model.LoginProvider, model.ProviderKey));

}

if (!result.Succeeded)

{

return GetErrorResult(result);

}

return Ok();

}

/// <summary>

/// Get external login

/// </summary>

/// <param name="provider"></param>

/// <param name="error"></param>

/// <returns></returns>

[OverrideAuthentication]

[HostAuthentication(DefaultAuthenticationTypes.ExternalCookie)]

[AllowAnonymous]

[Route("ExternalLogin", Name = "ExternalLogin")]

public async Task<IHttpActionResult> GetExternalLogin(string provider, string error = null)

{

if (error != null)

{

return Redirect(Url.Content("~/") + "#error=" + Uri.EscapeDataString(error));

}

if (!User.Identity.IsAuthenticated)

{

return new ChallengeResult(provider, this);

}

ExternalLoginData externalLogin = ExternalLoginData.FromIdentity(User.Identity as ClaimsIdentity);

if (externalLogin == null)

{

return InternalServerError();

}

if (externalLogin.LoginProvider != provider)

{

Authentication.SignOut(DefaultAuthenticationTypes.ExternalCookie);

return new ChallengeResult(provider, this);

}

User user = await UserManager.FindAsync(new UserLoginInfo(externalLogin.LoginProvider,

externalLogin.ProviderKey));

bool hasRegistered = user != null;

if (hasRegistered)

{

Authentication.SignOut(DefaultAuthenticationTypes.ExternalCookie);

ClaimsIdentity oAuthIdentity = await user.GenerateUserIdentityAsync(UserManager,

OAuthDefaults.AuthenticationType);

ClaimsIdentity cookieIdentity = await user.GenerateUserIdentityAsync(UserManager,

CookieAuthenticationDefaults.AuthenticationType);

AuthenticationProperties properties = ApplicationOAuthProvider.CreateProperties(user.UserName);

Authentication.SignIn(properties, oAuthIdentity, cookieIdentity);

}

else

{

IEnumerable<Claim> claims = externalLogin.GetClaims();

ClaimsIdentity identity = new ClaimsIdentity(claims, OAuthDefaults.AuthenticationType);

Authentication.SignIn(identity);

}

return Ok();

}

/// <summary>

/// Get external logins

/// </summary>

/// <param name="returnUrl"></param>

/// <param name="generateState"></param>

/// <returns></returns>

[AllowAnonymous]

[Route("ExternalLogins")]

public IEnumerable<ExternalLoginViewModel> GetExternalLogins(string returnUrl, bool generateState = false)

{

IEnumerable<AuthenticationDescription> descriptions = Authentication.GetExternalAuthenticationTypes();

List<ExternalLoginViewModel> logins = new List<ExternalLoginViewModel>();

string state;

if (generateState)

{

const int strengthInBits = 256;

state = RandomOAuthStateGenerator.Generate(strengthInBits);

}

else

{

state = null;

}

foreach (AuthenticationDescription description in descriptions)

{

ExternalLoginViewModel login = new ExternalLoginViewModel

{

Name = description.Caption,

Url = Url.Route("ExternalLogin", new

{

provider = description.AuthenticationType,

response\_type = "token",

client\_id = Startup.PublicClientId,

redirect\_uri = new Uri(Request.RequestUri, returnUrl).AbsoluteUri,

state = state

}),

State = state

};

logins.Add(login);

}

return logins;

}

/// <summary>

/// Register a user

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

[AllowAnonymous]

[Route("Register")]

public async Task<IHttpActionResult> Register(RegisterBindingModel model)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

User user = new User()

{

UserName = model.Email,

Email = model.Email,

Image = model.Image.ToRelativePath()

};

IdentityResult createUserResult = await UserManager.CreateAsync(user, model.Password);

if (!createUserResult.Succeeded)

{

return GetErrorResult(createUserResult);

}

var addRoleResult = await UserManager.AddToRoleAsync(user.Id, UserRoles.User);

return !addRoleResult.Succeeded ? GetErrorResult(addRoleResult) : Ok();

}

/// <summary>

/// Register external

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

[OverrideAuthentication]

[HostAuthentication(DefaultAuthenticationTypes.ExternalBearer)]

[Route("RegisterExternal")]

public async Task<IHttpActionResult> RegisterExternal(RegisterExternalBindingModel model)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var info = await Authentication.GetExternalLoginInfoAsync();

if (info == null)

{

return InternalServerError();

}

var user = new User() { UserName = model.Email, Email = model.Email };

IdentityResult result = await UserManager.CreateAsync(user);

if (!result.Succeeded)

{

return GetErrorResult(result);

}

result = await UserManager.AddLoginAsync(user.Id, info.Login);

if (!result.Succeeded)

{

return GetErrorResult(result);

}

return Ok();

}

// [Route("ChangeUserInfo")]

// public IHttpActionResult ChangeInfo([FromUri] )

// {

// return Ok();

// }

protected override void Dispose(bool disposing)

{

if (disposing && \_userManager != null)

{

\_userManager.Dispose();

\_userManager = null;

}

base.Dispose(disposing);

}

#region Helpers

private IAuthenticationManager Authentication

{

get { return Request.GetOwinContext().Authentication; }

}

private IHttpActionResult GetErrorResult(IdentityResult result)

{

if (result == null)

{

return InternalServerError();

}

if (!result.Succeeded)

{

if (result.Errors != null)

{

foreach (string error in result.Errors)

{

ModelState.AddModelError("errors", error);

}

}

if (ModelState.IsValid)

{

// No ModelState errors are available to send, so just return an empty BadRequest.

return BadRequest();

}

return BadRequest(ModelState);

}

return null;

}

private class ExternalLoginData

{

public string LoginProvider { get; set; }

public string ProviderKey { get; set; }

public string UserName { get; set; }

public IList<Claim> GetClaims()

{

IList<Claim> claims = new List<Claim>();

claims.Add(new Claim(ClaimTypes.NameIdentifier, ProviderKey, null, LoginProvider));

if (UserName != null)

{

claims.Add(new Claim(ClaimTypes.Name, UserName, null, LoginProvider));

}

return claims;

}

public static ExternalLoginData FromIdentity(ClaimsIdentity identity)

{

if (identity == null)

{

return null;

}

Claim providerKeyClaim = identity.FindFirst(ClaimTypes.NameIdentifier);

if (providerKeyClaim == null || String.IsNullOrEmpty(providerKeyClaim.Issuer)

|| String.IsNullOrEmpty(providerKeyClaim.Value))

{

return null;

}

if (providerKeyClaim.Issuer == ClaimsIdentity.DefaultIssuer)

{

return null;

}

return new ExternalLoginData

{

LoginProvider = providerKeyClaim.Issuer,

ProviderKey = providerKeyClaim.Value,

UserName = identity.FindFirstValue(ClaimTypes.Name)

};

}

# ПРИЛОЖЕНИЕ Б

Опись листов графической части дипломного проекта

Лист 1 – Цели и задачи проекта.

Лист 2 – Главная контекстная диаграмма (модель AS-IS).

Лист 3 – Декомпозиция контекстной диаграммы AS-IS.

Лист 4 – Главная контекстная диаграмма (модель TO-BE).

Лист 5 – Декомпозиция контекстной диаграммы TO-BE.

Лист 6 – Общая диаграмма вариантов использования.

Лист 7 – Физическая модель данных.

Лист 8 – Структура аппаратного развертывания приложения.

Лист 9 – UML диаграмма классов доступа к данным.

Лист 10 – Технико-экономические показатели проекта.

Лист 11 – Выводы по проекту.