**Ministry of Science and Higher Education of the Russian Federation**

**NATIONAL RESEARCH TOMSK STATE UNIVERSITY (NR TSU)**

**Higher IT School**

Course Work

Development of Student Management System

ZhiYuan Sun

Supervisor

Assistant Professor at the

Department of Software

Engineering

\_\_\_\_\_\_\_\_\_\_\_ Daniel Sokolov

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2021

Tomsk – 2021

Course work 35 pages blablablabla..

**Table of Contents**

[**Glossary** 5](#_Toc75646533)

[**Introduction** 7](#_Toc75646534)

[**Design** 8](#_Toc75646535)

[**Requirements:** 8](#_Toc75646536)

[**Roles Introduction:** 8](#_Toc75646537)

[**Functional Requirements:** 8](#_Toc75646538)

[**Non-Functional Requirements:** 8](#_Toc75646539)

[**Problem Domain Model:** 12](#_Toc75646540)

[**Used technologies and tools** 13](#_Toc75646541)

[**ASP.NET Core:** 13](#_Toc75646542)

[**The major characteristics of ASP.NET Core:** 13](#_Toc75646543)

[**Main Reasons why ASP.NET Core was chosen:** 14](#_Toc75646544)

[**ASP.NET Core MVC:** 15](#_Toc75646545)

[**Model Responsibilities:** 15](#_Toc75646546)

[**View Responsibilities:** 16](#_Toc75646547)

[**Controller Responsibilities:** 16](#_Toc75646548)

[**Entity Framework Core:** 16](#_Toc75646549)

[**Main Reasons why Entity Framework core was chosen:** 17](#_Toc75646550)

[**BootStrap:** 17](#_Toc75646551)

[**Main Reasons why BootStrap was chosen:** 18](#_Toc75646552)

[**jQuery:** 18](#_Toc75646553)

[**Main Reason why jQuery was chosen:** 18](#_Toc75646554)

[**Microsoft SQL Server:** 19](#_Toc75646555)

[**Implementation** 19](#_Toc75646556)

[**Class Diagram:** 19](#_Toc75646557)

[**Model Create:** 20](#_Toc75646558)

[**ApplicationDbContext Build:** 21](#_Toc75646559)

[**Email send using SMTP:** 22](#_Toc75646560)

[**Filtering, Sorting and Pagination:** 26](#_Toc75646561)

[**Rich text editor add:** 27](#_Toc75646562)

[**Localization:** 28](#_Toc75646563)

[**Record Creation:** 31](#_Toc75646564)

[**Login/Register with social media:** 33](#_Toc75646565)

[**Conclusion** 37](#_Toc75646566)

[**Refrences** 38](#_Toc75646567)

# **Glossary**

**SMS** – (abbreviation) Student Management System.

**Web Application** – Application software that runs on a web server.

**VS** – (abbreviation) Visual Studio. An integrated development environment (IDE) from Microsoft.

**Frontend** – The practice of converting data to a graphical interface, through the use of HTML, CSS, and JavaScript, so that users can view and interact with that data.

**Backend** – In the computer world, the "backend" refers to any part of a website or software program that users do not see. In programming terminology, the backend is the "data access layer".

**JavaScript** –  Often abbreviated as JS, is a programming language that conforms to the ECMAScript specification.

**BootStrap** – A free and open-source CSS framework directed at responsive, mobile-first front-end web development.

**jQuery** – A JavaScript library designed to simplify HTML DOM tree traversal and manipulation, as well as event handling, CSS animation, and Ajax.

**EF Core** – (abbreviation) Entity Framework Core is a lightweight, extensible, open source and cross-platform version of the popular Entity Framework data access technology.

**DB** – (abbreviation) Database.

**DBMS** – (abbreviation) Database Management System.

**Microsoft SQL Server** – A relational database management system developed by Microsoft.

**UML** – (abbreviation) Unified Modelling Language.

**NuGet** – A package manager designed to enable developers to share reusable code.

**Plugin** – A software add-on that is installed on a program, enhancing its capabilities.

**CDN** – (abbreviation) Content Delivery Network.

**AJAX** – (abbreviation) Asynchronous JavaScript and XML.

**UI** – (abbreviation) User interface design. This is the point of human-computer interaction and communication in a device. This can include display screens, keyboards, a mouse and the appearance of a desktop. It is also the way through which a user interacts with an application or a website.

**MailTrap** – Mailtrap is a service for the safe testing of emails sent from the development and staging environments. Mailtrap catches your emails in a virtual inbox so that you can test and optimize your email campaigns before sending them to real users.

**SMTP** – (abbreviation) – Simple Mail Transfer Protocol. An internet standard communication protocol for electronic mail transmission.

**Identity token** – A security token that contains Claims about the Authentication of an End-User by an Authorization Server when using a OAuth Client, and potentially other requested Claims. The Identity Token is represented as a JSON Web Token.

**CKEditor** – A WYSIWYG rich text editor which enables writing content directly inside of web pages or online applications.

# **Introduction**

Nowadays, because of the size of university increases. Every department has larger

amounts of students compared to the previous years.

  So, the traditional way to use Excel or similar applications to record students’ progress

in the practice becomes difficult and poor readability. Because use a form can’t show us

the students’ progress dynamic. Also, along with the tasks’ amounts increase, the

recorder even may forget the input data or input wrong data. For those reasons we need a web application to help people specially handle those works. Imagine you just need to create a task that you want to record students’ progress. And after you fill the forms, the system will save those data and show you dynamic percentage of each student’s progress.

  And Student Management System is a web application which can help you solve those

problems. It allows teachers to create reports and make notes recording student’s

practice. With the help of this web application, the recording work will be easier and

quicker.

# **Design**

  Before starting creating an application in the code, application design was conducted.

## **Requirements:**

  After receiving requirements for this application from the university, functional and

non-functional requirements were formed. This is a functionality that a complete

application should be provided to the user.

### **Roles Introduction:**

The whole website has two kinds of roles: Teacher and Admin. Teachers are the main users of the website. Has permission to use or view most of the resources but can’t change the deep data of the website. Admins are the people who mange and control the site. Has permission to do all changes at the website. Also one person can has two roles at the same time.

### **Functional Requirements:**

1. User can login, register and logout.
2. User can manage personal projects, records and profile.
3. User can add notes for projects and also and delete own notes.
4. Admin can manage whole projects, users, notes and records.

### **Non-Functional Requirements:**

1. Only authorized users can access the system with email and password.
2. Emails should be sent with a latency of no greater than 1 day.
3. Lifetime of the token should be more than 1 day.
4. The site supports at least two languages.
5. User can login/register with social medias.

**Use Case Diagram:**

For showing users’ possible interactions with a system, use case diagram was made(Picture 1):

图示

描述已自动生成

*Picture 1 - Use Case Diagram*

**Activity Diagram:**

To present the most necessary scenarios of how system’s logic work, two

activity diagrams were prepared(picture 2, picture 3):

图示

描述已自动生成

*Picture 2 – Activity Diagram about Admin manages users*

Here is one example of activity diagram about how admin manages users. In the logic

of how system works, it should has authorization part so based on it different roles

can have different permissions to unlock the functionalities in the system. If user try to

visit those parts which their role can’t support, they will be sent to access denied page. Also at the part in the database delete. Unlike the traditional way in the database if users delete one item all other items connect with it will all be deleted. In system’s database build, if users delete one item and this item has connection with other tables. Users must disconnect them first. In the system, all other parts about admin’s managements will follow the same logic.

图示

描述已自动生成

*Picture 3 – Activity Diagram about Teachers create record*

Because of the relationship between project and record is composition. So the record creation should be based on the project’s detail page.

## **Problem Domain Model:**

To present the business domain corresponding UML class diagram was made(Picture 4).

图形用户界面, 图示, 应用程序

描述已自动生成

*Picture 4 – Problem domain model*

In the system, one user can have multiple roles, teacher and admin. Also users can create project but also they can access other people’s projects. So relationship between them should be many to many. Record and note can only be created under the project. So relationship between relationship between project and record or project and note should be one to many.

## **Used technologies and tools**

Choosing a web application framework is challenging, there are a lot of frameworks that are very powerful nowadays, each framework has its own pros and cons. Asp.Net Core was chosen to develop the back-end of this web application for many reasons that is going to be demonstrated below. Bootstrap and jQuery were chosen to develop the front-end of this project. Microsoft SQL Server was chosen as my database.

### **ASP.NET Core:**

ASP.NET Core is a cross-platform, high-performance, open-source framework for building modern, cloud-enabled, Internet-connected apps.

#### **The major characteristics of ASP.NET Core:**

* Cross-platform support  
  It can work on multiple platforms, such as Windows, Mac, and Linux.
* Flexible deployment  
  You can deploy a .NET Core application side by side with your application seamlessly. It is a general-purpose development platform that consists of several components. These include the managed compilers, the runtime, and the base class libraries. It also includes many application models, such as the ASP.NET Core
* Modular  
  .NET Core is composed of a set of modular components. This enables you to take advantage of the package you want to use rather than including the entire .NET Core framework. This boosts performance as you end up creating applications that contain just what you need.
* Open-source  
  .NET Core is open source using MIT and Apache 2 licenses and is available in GitHub.

#### **Main Reasons why ASP.NET Core was chosen:**

* A unified story for building web UI and web APIs.
* Architected for testability.
* Razor Pages makes coding page-focused scenarios easier and more productive.
* Blazor lets you use C# in the browser alongside JavaScript. Share server-side and client-side app logic all written with .NET.
* Ability to develop and run on Windows, macOS, and Linux.
* Open-source and community-focused.
* A cloud-ready, environment-based configuration system.
* Built-in dependency injection.
* A lightweight, high-performance, and modular HTTP request pipeline.
* Ability to host on the following:
  + Kestrel
  + IIS
  + HTTP.sys
  + Nginx
  + Apache
  + Docker
* Side-by-side versioning.
* Tooling that simplifies modern web development.

#### **ASP.NET Core MVC:**

ASP.NET Core MVC is a rich framework for building web apps and APIs using the Model-View-Controller design pattern. The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. This pattern helps to achieve separation of concerns. Using this pattern, user requests are routed to a Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries. The Controller chooses the View to display to the user, and provides it with any Model data it requires(Picture 5).

图形用户界面, 应用程序, 日程表

描述已自动生成

*Picture 5 – MVC Architectural Diagram*

##### **Model Responsibilities:**

The Model in an MVC application represents the state of the application and any business logic or operations that should be performed by it. Business logic should be encapsulated in the model, along with any implementation logic for persisting the state of the application. Strongly-typed views typically use ViewModel types designed to contain the data to display on that view. The controller creates and populates these ViewModel instances from the model.

##### **View Responsibilities:**

Views are responsible for presenting content through the user interface. They use the Razor view engine to embed .NET code in HTML markup. There should be minimal logic within views, and any logic in them should relate to presenting content. If you find the need to perform a great deal of logic in view files in order to display data from a complex model, consider using a View Component, ViewModel, or view template to simplify the view.

##### **Controller Responsibilities:**

Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render. In an MVC application, the view only displays information; the controller handles and responds to user input and interaction. In the MVC pattern, the controller is the initial entry point, and is responsible for selecting which model types to work with and which view to render (hence its name - it controls how the app responds to a given request).

#### **Entity Framework Core:**

Entity Framework Core (EF Core) is an object-oriented, lightweight, and extensible technology from Microsoft for data access. EF Core is an object-relational mapping (ORM) tool. That is, EF Core allows you to work with databases, but it represents a higher level of abstraction: EF Core allows you to abstract from the database itself and its tables and work with data regardless of the type of storage. If at the physical level we operate with tables, indexes, primary and foreign keys, but at the conceptual level that Entity Framework offers us, we are already working with objects.

Entity Framework Core supports many different database systems. Thus, we can work with any DBMS through EF Core, if the required provider is available for it. For showing the logic how EF core works, a simplified diagram was drawn(Picture 6).

图示

描述已自动生成

*Picture 6 – EF Core working logic*

##### **Main Reasons why Entity Framework core was chosen:**

* The visual design interface can help to create entities easily in a short time.
* The database sync is easy for environments using migration. It is a really needed.
* Аeature which makes it possible to upgrade or downgrade any change/commit.
* You get control on the code so you can validate fields from classes as well.
* Can easily use and control database even you don’t have good knowledge with it.

### **BootStrap:**

Bootstrap is a free and open-source framework for creating websites and web applications. It's the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.

#### **Main Reasons why BootStrap was chosen:**

* Fewer Cross browser bugs
* A consistent framework that supports major of all browsers and CSS compatibility fixes
* Lightweight and customizable
* Responsive structures and styles
* Good documentation and community support
* Loads of free and professional templates, WordPress themes and plugins
* Great grid system.

### **jQuery:**

**jQuery**is an open source JavaScript library that simplifies the interactions between an HTML/CSS document, or more precisely the Document Object Model (DOM), and JavaScript.  
 Elaborating the terms, jQuery simplifies HTML document traversing and manipulation, browser event handling, DOM animations, Ajax interactions, and cross-browser JavaScript development.

#### **Main Reason why jQuery was chosen:**

* Wide range of plug-ins. jQuery allows developers to create plug-ins on top of the JavaScript library.
* Large development community.
* It has a good and comprehensive documentation.
* It is a lot more easy to use compared to standard JavaScript and other JavaScript libraries.
* Being Light weight and a powerful chaining capabilities makes jQuery more strong.

### **Microsoft SQL Server:**

Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run either on the same computer or on another computer across a network (including the Internet).

# **Implementation**

## **Class Diagram:**

图示, 示意图

描述已自动生成

*Picture 7 – Class Diagram*

**Preparation:**

To start the development, VS should be installed on a local device. Also after creating a new project on the VS. The necessary packages should be downloaded from NuGet. Need EntityFrameworkCore.SqlServer, EntityFrameworkCore.Tools and AspNetCore.EntityFrameworkCore.Identity. And then add them in the startup.cs.

## **Model Create:**

Follow the class diagram above,model can be easily created. And below an example was shown about how user model created(Code sample 1).

public class User : IdentityUser

{

public string FirstName { get; set; }

public string LastName { get; set; }

public string ImagePath { get; set; }

public string FullName

{

get { return FirstName + " " + LastName; }

}

public ICollection<Project> Projects { get; set; }

public ICollection<Note> Notes { get; set; }

public ICollection<Record> Records { get; set; }

}

*Code sample 1 – Create User model*

As can see, due to the package AspNetCore.EntityFrameworkCore.Identity installed. User Model can inherit necessary properties it needs from IdentityUser like Id, Password etc.… instead of creating by ourselves. Also the important thing needs to notice is the relationships between user and other classes. As said before, the system uses EF Core to manage database. The most common pattern for relationships is to have navigation properties defined on both ends of the relationship and a foreign key property defined in the dependent entity class. So in the user model, Icollection with Record was added.

## **ApplicationDbContext Build:**

After all necessary entities finished build. It is time to add them in the ApplicationDbContext(Code sample 2).

public class ApplicationDbContext : IdentityDbContext<User>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext>

options):base(options)

{

}

public DbSet<Student> Students { get; set; }

public DbSet<Project> Projects { get; set; }

public DbSet<Note> Notes { get; set; }

public DbSet<Record> Records { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<Project>().HasOne(x => x.Creator).WithMany().OnDelete(DeleteBehavior.Restrict);

modelBuilder.Entity<Project>().HasOne(x => x.Creator).WithMany().OnDelete(DeleteBehavior.Restrict);

modelBuilder.Entity<Project>().HasOne(x => x.Creator).WithMany().OnDelete(DeleteBehavior.Restrict);

foreach (var foreignKey in modelBuilder.Model.GetEntityTypes()

.SelectMany(e => e.GetForeignKeys()))

{

foreignKey.DeleteBehavior = DeleteBehavior.Restrict;

}

}

}

*Code sample 2 – ApplicationDbContext build*

Cascading referential integrity constraint allows to define the actions Microsoft SQL Server should take when a user attempts to delete or update a key to which an existing foreign keys points. So remember the activity diagram(picture 2). In there has logic about if the user you want to delete has linked with other tables. You must break the key between them first and then you can delete this user. So for this reason in the applicationDbContext adds the code which highlighted for helping system enforce on delete no action.

## **Email send using SMTP:**

In the non-functional requirements. There mentioned about email send. So the system will sends email contains token to user’s email address for helping them confirm their account or when they forgot password help them change password.

Since know it is not a real website yet. So mailtrap.io was used as a simulation.

So after getting the SMTP settings from the website, it should be added to the appsettings.json file(Code sample 3).

"SMTPConfig": {

"SenderAddress": "no-reply@StudentManagementApp.com",

"SenderDisplayName": "Student Management System Team",

"UserName": "45f5eb7xxxxx",

"Password": "f298b249xxxxx",

"host": "smtp.mailtrap.io",

"Port": "25",

"EnableSSL": true,

"UseDefaultCredentials": false,

"IsBodyHTML": true

}

*Code sample 3 – Add SMTPConfig in appsettings.json*

After the basic setup finished. Still needs two classes in the model. One called UserEmailOptions(Code sample 4), it will contains email’s form, subject, body, receivers and placeHolders. The other one called SMTPConfigModel(Code sample 5) will match the fields which be added in the appsettings.json file(Code sample 3).

public class UserEmailOptions

{

public List<string> ToEmails { get; set; }

public string Subject { get; set; }

public string Body { get; set; }

public List<KeyValuePair<string, string>> PlaceHolders { get; set; }

}

*Code sample 4 – UserEmailOptions*

public class SMTPConfigModel

{

public string SenderAddress { get; set; }

public string SenderDisplayName { get; set; }

public string UserName { get; set; }

public string Password { get; set; }

public string host { get; set; }

public int Port { get; set; }

public bool EnableSSL { get; set; }

public bool UseDefaultCredentials { get; set; }

public bool IsBodyHTML { get; set; }

}

*Code sample 5 – SMTPConfigModel*

Before creating the EmailService. Still has last step. Create email template View(Code sample 6).

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

</head>

<body>

<p>

Hello **{{**UserName**}}**, <br />

Thank you for using the student management system.<br />

You have created a new account on student management system. <br />

Click on the following link to confirm email account. <br />

<a href="**{{**Link**}}**">Vertify Email</a>

</p>

</body>

</html>

*Code sample 6 – Email template*

So finally after those preparations. It is time to move to EmailService create part. For getting the email forms where created from the template and matching the SMTP settings in the appsettings.json.file. SendEmail function was created(Code sample 7).

private async Task SendEmail(UserEmailOptions userEmailOptions)

{

MailMessage mail = new MailMessage

{

Subject = userEmailOptions.Subject,

Body = userEmailOptions.Body,

From = new MailAddress(\_smtpConfig.SenderAddress, \_smtpConfig.SenderDisplayName),

IsBodyHtml = \_smtpConfig.IsBodyHTML

};

foreach (var toEmail in userEmailOptions.ToEmails)

{

mail.To.Add(toEmail);

}

NetworkCredential networkCredential = new NetworkCredential(\_smtpConfig.UserName, \_smtpConfig.Password);

SmtpClient smtpClient = new SmtpClient

{

Host = \_smtpConfig.host,

Port = \_smtpConfig.Port,

EnableSsl = \_smtpConfig.EnableSSL,

UseDefaultCredentials = \_smtpConfig.UseDefaultCredentials,

Credentials = networkCredential

};

mail.BodyEncoding = Encoding.Default;

await smtpClient.SendMailAsync(mail);

}

*Code sample 7 – SendEmail function*Also due to template’s type is html page, so need other function for getting the email Body(Code sample 8).

private string GetEmailBody(string templateName)

{

var body = File.ReadAllText(string.Format(templatePath, templateName))

return body;

}

*Code sample 8 – SendEmail function*

In the template uses {{UserName}} and {{link}}. Because system should send dynamic datas due to differnent users. For solving this problem, UpdatePlaceHolders function was created(Code sample 9) for helping change those data to the dynamic value.

private string UpdatePlaceHolders(string text, List<KeyValuePair<string, string>> keyValuePairs)

{

if(!string.IsNullOrEmpty(text) && keyValuePairs != null)

{

foreach(var placeholder in keyValuePairs)

{

if(text.Contains(placeholder.Key))

{

text = text.Replace(placeholder.Key, placeholder.Value);

}

}

}

return text;

}

*Code sample 9 – UpdatePlaceHolders function*

And in the interface IEmailService(Code sample 10) has two properties. Now it is time to add them in the EmailService(Code sample 11).

public interface IEmailService

{

Task SendEmailForEmailConfirmation(UserEmailOptions userEmailOptions);

Task SendEmailForForgettenPassword(UserEmailOptions userEmailOptions);

}

*Code sample 10 – IEmailService*

public async Task SendEmailForEmailConfirmation(UserEmailOptions userEmailOptions)

{

userEmailOptions.Subject = UpdatePlaceHolders("Hello {{UserName}}, Please confirm your email address", userEmailOptions.PlaceHolders);

userEmailOptions.Body = UpdatePlaceHolders(GetEmailBody("ConfirmEmail"), userEmailOptions.PlaceHolders);

await SendEmail(userEmailOptions);

}

*Code sample 11 – SendEmailForEmailConfirmation*

So since now, Email service already finished build. For real send it and see the result on MailTrap.io. In the AccountController register function. System needs to generate token and confirmationLink when user registers successfully(Code sample 12).

var token = await userManager.GenerateEmailConfirmationTokenAsync(user);

var confirmationLink = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, token = token }, Request.Scheme);

await SendEmailConfirmationEmail(user, confirmationLink);

*Code sample 12 – Token and link generate*

And the last step is creating function SendEmailConfirmationEmail(Code sample 13).

private async Task SendEmailConfirmationEmail(User user, string confirmationLink)

{

UserEmailOptions userEmailOptions = new UserEmailOptions

{

ToEmails = new List<string>() { user.Email },

PlaceHolders = new List<KeyValuePair<string, string>>()

{

new KeyValuePair<string, string>("{{UserName}}", user.fullname),

new KeyValuePair<string, string>("{{Link}}", confirmationLink)

}

};

await emailService.SendEmailForEmailConfirmation(userEmailOptions);

}

*Code sample 13 – SendEmailForEmailConfirmationEmail function*

So finally the part send email with SMTP was finished and can see the email on the Website(Picture 8).

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

*Picture 8 – Confirmation Email*

## **Filtering, Sorting and Pagination:**

After user login. System should shows them a table of projects. So let user better find and manage their projects. Needs to add filtering, sorting(Picture 9) and pagination(Picture 10) in the table page.



*Picture 9 – Filtering and sorting*

*Picture 10 – Pagination*

Unlike the traditional way that create service helps split pages or in the project controller write code for filtering. In the introduction part of jQuery was mentioned, it supports lots of plugins help us simplify code work. So here dataTable was used for(Code sample 14)solving those functionalities.

@section Scripts {

<link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/v/dt/dt-1.10.23/datatables.min.css" />

<script type="text/javascript" src="https://cdn.datatables.net/v/dt/dt-1.10.23/datatables.min.js"></script>

<script>

$(document).ready(function () {

$("#Exampledatatable").DataTable();

});

</script>

}

*Code sample 14 – dataTable plugin add*

For letting this plugin works in the view page. The page should includes dataTable source files. Here uses CDN to add them to the page. After it, just needs to call the table by class or id. This plugin will helps system do all filtering, sorting and pagination work.

## **Rich text editor add:**

In the project create part, users should have a rich text editor(Picture 11) in the project’s description part. Because sometimes description can be very long or users may want to add links, photos, forms etc… in their description. With a rich text editor can help them achieve those technologies and also good type settings can help users read more easily.

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

*Picture 11 – Rich text editor*

For the rich text editor, it is sure will not write code to implement it by ourselves. Because it is too complex. So as an instead, can download a package which called CKEditor. After copying and pasting the download folder to the VS’s solution explorer(Picture 12).



*Picture 12 – Add CKEditor folder*

After it in the project’s create view page, should add this JS folder and let it replace normal text area description with rich text editor(Code sample 15).

<script src="~/js/ckeditor/ckeditor.js"></script>

<script>CKEDITOR.replace('Description');</script>

*Code sample 15 – Call CkEditor in view page*.

## **Localization:**

For helping those users who are not good at English, the localization was implemented at the system. Which means the system supports two languages: Simplified Chinese and English.

Before creating the resources file, it is required to add localization config in the startup.cs page(Code sample 16).

services.AddMvc().AddViewLocalization(LanguageViewLocationExpanderFormat.Suffix).AddDataAnnotationsLocalization();

services.Configure<RequestLocalizationOptions>(

opt =>

{

var supportedCultures = new List<CultureInfo>

{

new CultureInfo("en"),

new CultureInfo("zh")

};

opt.DefaultRequestCulture = new RequestCulture("en");

opt.SupportedCultures = supportedCultures;

opt.SupportedUICultures = supportedCultures;

});

*Code sample 16 – Config localization*

After configuring the localization, it still needs resource files where save the both English and Simplified Chinese. So due to the different view, different resources file were created(Picture 13).

图形用户界面, 文本, 应用程序

描述已自动生成

*Picture 13 – Resource files*

When the resource files finish created, they should be injected in the view page(Code sample 17). And instead of normal text which input before, it should be replaced with the localization one(Code sample 18).

@inject Microsoft.AspNetCore.Mvc.Localization.IViewLocalizer localizer

*Code sample 17 – Inject localization file*

<h2>@localizer["title"]</h2>

*Code sample 18 – Text replaced*

For being more user friendly, when users switch language once, system supposes to remember that users required language and implement it to whole site instead of when users go to differnent page they need to switch again. So the action called CultureManagement was added in the HomeController for saving the users’ cookies(Code sample 19).

[HttpPost]

public IActionResult CultureManagement(string culture, string returnUrl)

{

Response.Cookies.Append(CookieRequestCultureProvider.DefaultCookieName, CookieRequestCultureProvider.MakeCookieValue(new RequestCulture(culture)),

new CookieOptions { Expires = DateTimeOffset.Now.AddDays(30) });

return LocalRedirect(returnUrl);

}

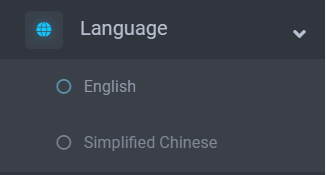
*Code sample 19 – CultureManagement*

For sending the culture and returnUrl data, in the \_layout page, inside the tag<form> input value should be wroten(Code sample 20).

<input **type**="hidden"**name**="culture"**asp-for**="@culture.RequestCulture.UICulture.Name" **value**="en" />

*Code sample 20 – Send culture data*

After some styles add, on the system sidebar page can see language switch button(Picture 14).



*Picture 14 – Language switch button*

## **Record Creation:**

The activity diagram before shows the logic how record creation works. In project’s properties. It has start time and deadline, the main difficult part in record creation is to split the date between strat time and deadline. For solving this, the class SplitWeek was created in the service(Code sample 21).

public class Week

{

public static IEnumerable<String> Split(DateTime start, DateTime end)

{

DateTime chunkEnd;

while ((chunkEnd = start.AddDays(6)) < end)

{

yield return start.ToShortDateString() + " - " + chunkEnd.ToShortDateString();

start = chunkEnd.AddDays(1);

}

yield return start.ToShortDateString() + " - " + start.AddDays(6).ToShortDateString();

}

}

*Code sample 21 – SplitWeek class*

As a web page, in the record controller. The create action consists of two parts: HttpGet and HttpPost. As shown in the problem domain model, the relationship between record and project is composition. So the code in HttpGet for confirming project was wroten(Code sample 22).

var project = await this.context.Projects

.SingleOrDefaultAsync(m => m.Id == projectId);

if (project == null)

{

return this.NotFound();

}

*Code sample 22 – Search project*

With the use of SplitWeek service, selectlist of students(Code sample 23) and the view page wrote. The record creation page was made(Picture 15).

var GetWeek = Week.Split(project.StartTime, project.DeadLine);

ViewBag.StudentName = new SelectList(context.Students, “StudentName”, “StudentId”);

*Code sample 23 – use of SplitWeek service and selectlist of students*

图形用户界面, 应用程序

描述已自动生成

*Picture 15 – Record creation page*

After user inputing the necessary data. In the HttpPost code for generating the new record and saving in database was wroten(Code sample 24).

var now = DateTime.UtcNow;

var record = new Record

{

ProjectId = project.Id,

CreatorId = user.Id,

Created = now,

LogTime = model.LogTime,

StudentEmail = model.StudentName,

Week = model.Week

};

this.context.Add(record);

await this.context.SaveChangesAsync();

*Code sample 24 – Create new record and save in database*

## **Login/Register with social media:**

As in non-functional requirements mentioned, user can login with social medias. So the system will supports users register with google or facebook. To visit the whole process request between user, site, google/facebook and database. A sequence diagram was drawn(Picture 16).

图示

描述已自动生成

*Picture 16 – Sequence diagram about external login*

For using the external login from the google or facebook. Packages AspNetCore.Authentication.Facebook and AspNetCore.Authentication.Google should be installed from the NuGet. After getting the Id and secret from google and facebook. In the startup page should configure the authentication(Code sample 25).

services.AddAuthentication()

.AddGoogle(options =>

{

options.ClientId = "324353xxxxx-.apps.googleusercontent.com";

options.ClientSecret = "1lQeKN9Ns3Zxxxxxxxx";

})

.AddFacebook(options =>

{

options.AppId = "902776xxxxxxxx";

options.AppSecret = "131c4129875fe5xxxxxxxx";

});

*Code sample 25 – Config the authentication*

After it, for sending the users’ requests and redirect to the google login UI(Picture 17), the externalLogin action was created in the account controller(Code sample 26).

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

*Picture 17 – Google login ui*

[AllowAnonymous]

[HttpPost]

public IActionResult ExternalLogin(string provider, string returnUrl)

{

var redirectUrl = Url.Action("ExternalLoginCallback", "Account",

new { ReturnUrl = returnUrl });

var properties = signInManager

.ConfigureExternalAuthenticationProperties(provider, redirectUrl);

return new ChallengeResult(provider, properties);

}

*Code sample 26 – ExternalLogin action*

In the code provider used to match the authentication server and returnUrl for helping return the site correctly after getting the user’s credentials. With the view page post part added. Buttons for external logins have done(Picture 18).

<form id="myForm" method="post" **asp-action**="ExternalLogin" **asp-route-returnUrl**="@Model.ReturnUrl">

<div class="social-login">

@foreach (var provider in Model.ExternalLogins)

{

if (provider.Name == "Facebook")

{

<input type="hidden" value="@provider.Name" name="provider" />

<a href="#" onclick="document.getElementById('myForm').submit();" class="facebook">

<span class="icon-facebook mr-3"></span>

</a>

}

</form>

*Code sample 27 – ExternalLogin View*

*图片包含 图形用户界面

描述已自动生成*

*Picture 18- External login buttons*

After getting the user’s credentials, google supposed to redirect this request back to the site. To present it, ExternalLoginCallBack action was created. The action consists of two parts. Get the login information about the user from the external login provider(Code sample 28) and check whether user has local account in the site(Code sample 29).

var info = await signInManager.GetExternalLoginInfoAsync();

var email = info.Principal.FindFirstValue(ClaimTypes.Email);

*Code sample 28 – Get login information*

var signInResult = await signInManager.ExternalLoginSignInAsync(info.LoginProvider,

info.ProviderKey, isPersistent: false, bypassTwoFactor: true);

if (signInResult.Succeeded)

{

return LocalRedirect(returnUrl);

}

else

{

if (email != null)

{

user = await userManager.FindByEmailAsync(email);

if (user == null)

{

user = new User

{

UserName = info.Principal.FindFirstValue(ClaimTypes.Email),

Email = info.Principal.FindFirstValue(ClaimTypes.Email)

};

return this.View();

}

await userManager.AddLoginAsync(user, info);

await signInManager.SignInAsync(user, isPersistent: false);

return LocalRedirect(returnUrl);

}

return View("Error");

}

*Code sample 29 – Check whether user has local account*

Above follow the sequence diagram. Site uses the logic: if user already had a local account then sign in the user with this external login provider. Otherwise, the site gets the email claim value and create an account without password. Then insert a row for the user in AspNetUserLogins table.

# **Conclusion**

In the course of the work, a web application was created using the .NET Core and EF Core.

All set goals have been achieved during the work, including:

* Creating requirements for application.
* Carrying out the designs of the application.
* Implementing all distinguished use-case in code.

Additionally, working with .NET Core, EF Core and more skills in front-end development was learned during the development.

# **Refrences**

1. Visual Studio //- Wikipedia[S,1], 2021 - URL:

<https://en.wikipedia.org/wiki/Microsoft_Visual_Studio> (access date 01.06.2021)

2. ASP.NET Core//- Docs.Microsoft[S,1], 2021 - URL:

<https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0> (access date 01.06.2021)