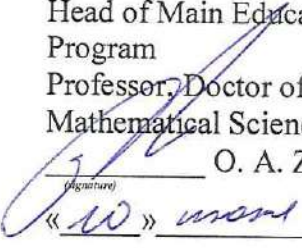


Ministry of Science and Higher Education of the Russian Federation
NATIONAL RESEARCH
TOMSK STATE UNIVERSITY (NR TSU)
Research and Education Center "Higher IT School" (HITs)

APPROVED BY
Head of Main Educational
Program
Professor, Doctor of Physical and
Mathematical Sciences


(signature) O. A. Zmeev,
« 10 » 06 2022

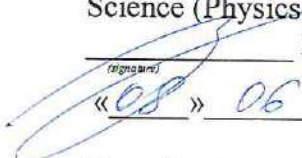
BACHELOR'S THESIS

THE DEVELOPMENT OF WEB APPLICATION "TSU STUDENT MANAGEMENT
SYSTEM" USING ASP. NET CORE

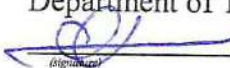
Main educational program
02.03.02 – Fundamental Computer Science and Information Technology
Specialization "Software Engineering"

Sun Zhiyuan

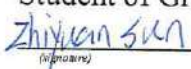
Bachelor's Thesis Supervisor
Associate Professor, Cand. of
Science (Physics & Mathematics)


(signature) Zh. N. Zenkova
« 08 » 06 2022

Consultant
The head of Digital Service
Department of TSU


(signature) D. A. Sokolov
« 08 » 06 2022

Written by
Student of Group No. 971812


(signature) Zh. Sun
« 07 » 06 2022

The Ministry of Science and Higher Education of the Russian Federation
NATIONAL RESEARCH TOMSK STATE UNIVERSITY (NR TSU)

REC "Higher IT School"

APPROVED
MAP Head
Dr. Sc. (Phys.-Math.), professor
O.A. Zmeev

 20 22

THE TASK

Regarding the bachelor's final qualification work implementation by a student
Zhiyuan Sun

(student's full name)

In the field of study Fundamental Computer Science and Information Technology, specialization
"Software Engineering"

1. The bachelor's final qualification work theme (in the English and in the Russian language)
THE DEVELOPMENT OF WEB APPLICATION "TSU STUDENT MANAGEMENT

SYSTEM" USING ASP. NET CORE / РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ "ТГУ

СИСТЕМА УПРАВЛЕНИЯ СТУДЕНТАМИ" С ИСПОЛЬЗОВАНИЕМ ASP. NET CORE

2. The deadlines for the completion of task:

a) to academic office – 10.06. 2022.

б) to State Examination Board – 14.06. 2022

3. Description:

The purpose of this work is to develop a web application which allows teachers to create

goals and objectives of the FQW, expected results

reports, make notes recording student's practice and help them communicate with each other.

The expected results: creating a full stack web application using ASP .NET Core MVC.

The organization that is being researched:

Tomsk State University, Digital Service Department

Final qualification work supervisor

Associate Professor, Cand. of Science

(Physics & Mathematics)

(position, place of work)

Final qualification work consultant

The head of Digital Service Department
of TSU

(position, place of work)

The task is accepted for execution

18.02 2022
(date)

(signature)

/ Zh. N. Zenkova
(initials, surname)

(signature)

/ D.A. Sokolov
(initials, surname)

(signature)

/ Zh. Sun
(initials, surname)

ABSTRACT

Bachelor's thesis 53 pages, 59 pictures, 15 code samples, 1 appendix.

The object of study is the development of the TSU student management system.

The purpose of the study is to develop a web application using ASP. NET Core MVC framework.

Research methods are theoretical research and practical implementation.

Work results: application requirements identified, the design is carried out, and the application is implemented.

Key words: STUDENT MANAGEMENT SYSTEM DEVELOPMENT, ASP. NET CORE, ENTITY FRAMEWORK CORE, BOOTSTRAP, MICROSOFT SQL SERVER, WEB APPLICATION.

CONTENT

Glossary.....	4
Introduction	6
1 Design.....	7
1.1 Requirements	7
1.2 Use case diagram.....	8
1.3 Activity diagrams	9
1.4 Problem domain model.....	11
1.5 Used technologies and tools	12
1.5.1 ASP. NET CORE	12
1.5.2 ASP. NET CORE MVC.....	13
1.5.3 Entity framework core	15
1.5.4 Bootstrap.....	16
1.5.5 jQuery	17
1.5.6 Microsoft SQL Server.....	18
2 Implementations.....	19
2.1 Class diagram.....	19
2.2 Preparation	19
2.3 Model creation	20
2.4 ApplicationDbContext creation	20
2.5 Record creation	22
2.6 Localization	26
2.7 Login/register with social media.....	29

2.8 Real time public chat	33
Conclusion.....	40
References	41
Appendix A	42

GLOSSARY

TSU – (abbreviation) Tomsk State University.

SMS – (abbreviation) Student Management System.

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

Full Stack – The development of both frontend and backend portions of web application.

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.

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

DOM – (abbreviation) Document object model. When a page is created and loaded into the browser, the DOM comes into being, which converts the web page document into a document object, whose main function is to process the content of the web page. In this document object, all elements present a hierarchical structure.

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.

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”.

CRUD – (abbreviation) Create, read, update, delete.

UML – (abbreviation) Unified Modelling Language.

IDE – (abbreviation) An integrated development environment (IDE) is a software for building applications that incorporates commonly used developer tools into a single graphical user interface (GUI).

DB – (abbreviation) Database.

DBMS – (abbreviation) Database Management System.

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.

HTTP – (abbreviation) Hyper Text Transfer Protocol. A simple request-response protocol.

ES – (abbreviation) ECMAScript.

API – (abbreviation) Application Programming Interface.

Cookie – A piece of data that website puts on the user's local device when you visit. It usually contains information about the website itself.

INTRODUCTION

Nowadays, the number of students in each department increases rapidly due to the social development and the admission expansion of universities every year.

But in the student management part, a lot of universities are still using outdated tools like Excel or similar applications. Some even work with papers, files and binders. No doubt, the traditional way of managing student's work and progress is a challenge for university's employees. Because of facing mass data, people will be more error prone. Then for solving those problems and liberating people from complex and cumbersome work. A web application specific for managing students is in demand. Imagining, all data is stored in the database, you don't need to worry about where to store those easily lost papers and security of those data is ensured. And a lot of formatted templates are supported, you can easily create projects you need and record student's progress.

Student management system is a web application which can suit all your requirements above. It allows teachers to create subjects and make records for student's progress. More than that, real time chat technology is built-in for simplifying the chatting between teachers and students. And due to the echarts's diagrams, all data in the site will be readable and manifest.

The project is considered to be a full-stack project as in the backend .NET Core framework was chosen, and in the frontend bootstrap and jQuery was chosen.

1 Design

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

1.1 Requirements

After receiving requirements for this application from the university, functional and non-functional requirements were formed. Below are all functionalities that a complete application should be provided to the user.

Roles Introduction:

The website has three kinds of roles: Teacher, Student and Admin. Teachers are the main users of the website. Have permission to use or view most of the resources but can't change the deep data of the website. Students are the users which have basic authorization. Admin is the special role which can manage and control the whole site. Has permission to do all changes at the website. Also, one person can have more than one role at the same time.

Functional Requirements:

Users:

Communicate with each other in the chat room.

Students:

Upload personal attachments directly in the project.

Teachers:

1. CRUD personal subjects, projects, attendances, notifications and records.
2. Mark student's attachments.

Administrators:

Manage all users, subjects, projects, records, attachments and chat rooms.

Non-Functional Requirements:

1. The site supports at least two languages.
2. Users can login or register with social medias.
3. ASP. NET Core should be the development framework.

1.2 Use case diagram

For showing users' possible interactions with a system, use case diagram was made (Figure 1).



Figure 1 – Use case diagram

Manage in the diagram means the CRUD actions. For the students and teachers, they can only manage the personal data or those data which are related to them. Administrators have the highest authority which allows them to access all data on the site.

1.3 Activity diagrams

To present the most necessary scenarios of how system's logic work, two activity diagrams were prepared (Figure 2, Figure 3):

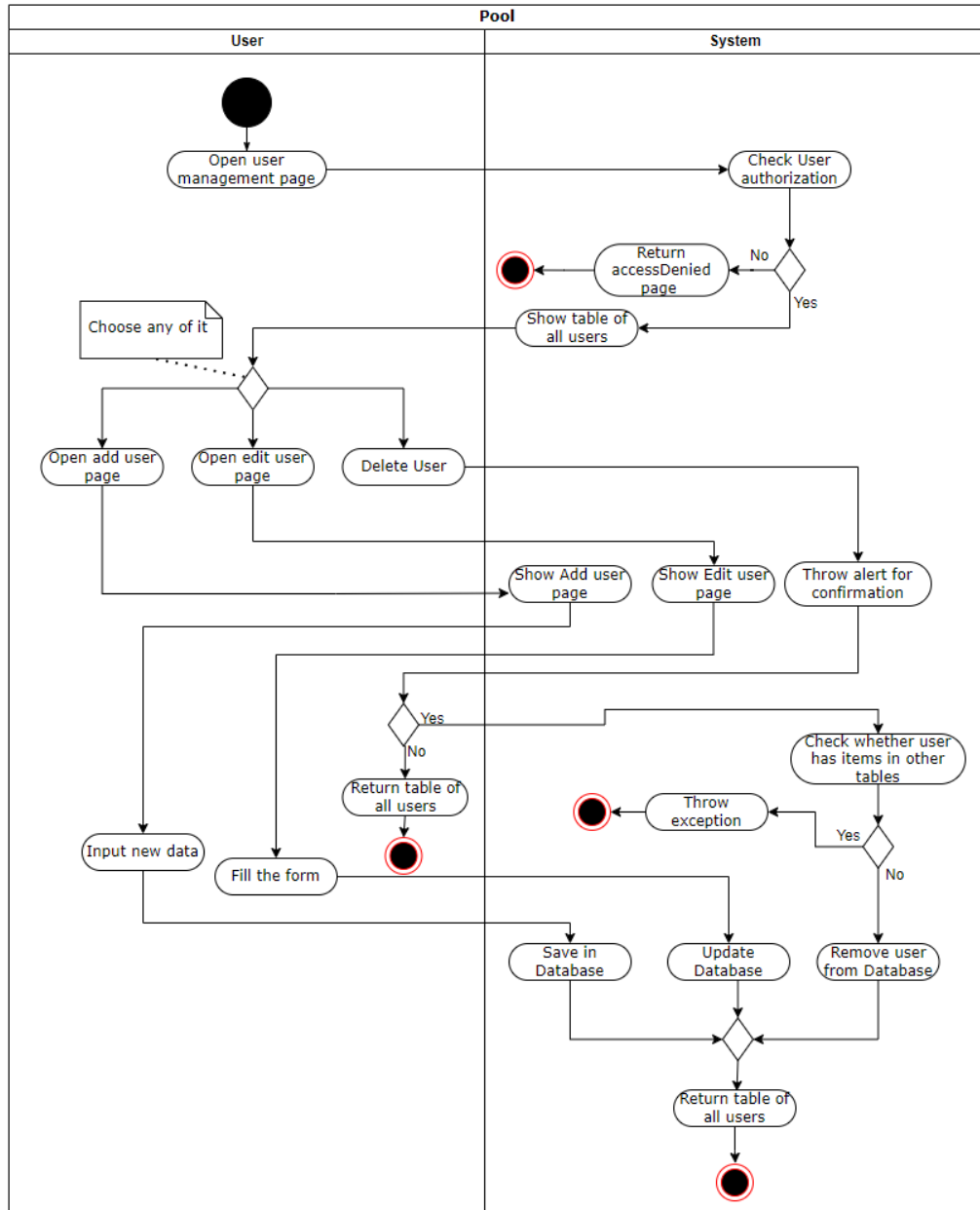


Figure 2 – Admins manage users

Above is an example about how admins manage users. In the logic of how the system works, it is supposed to have an authorization part which can check the current user's roles. Due to the check result, it will block or unlock resources of the system to this user. And also, a specific page should be designed (access denied page). User will be returned to this page if he doesn't have enough permission.

For avoiding accidental deletion and also for protecting user's data. Delete action in this site is distinct compared to the traditional delete action. After the delete button is clicked, the system will check whether still have resources exist in the site which belong to this user. If the result is true, then this delete process will be paused. Admin must remove all resources which belong to this user first and then he can delete this user.

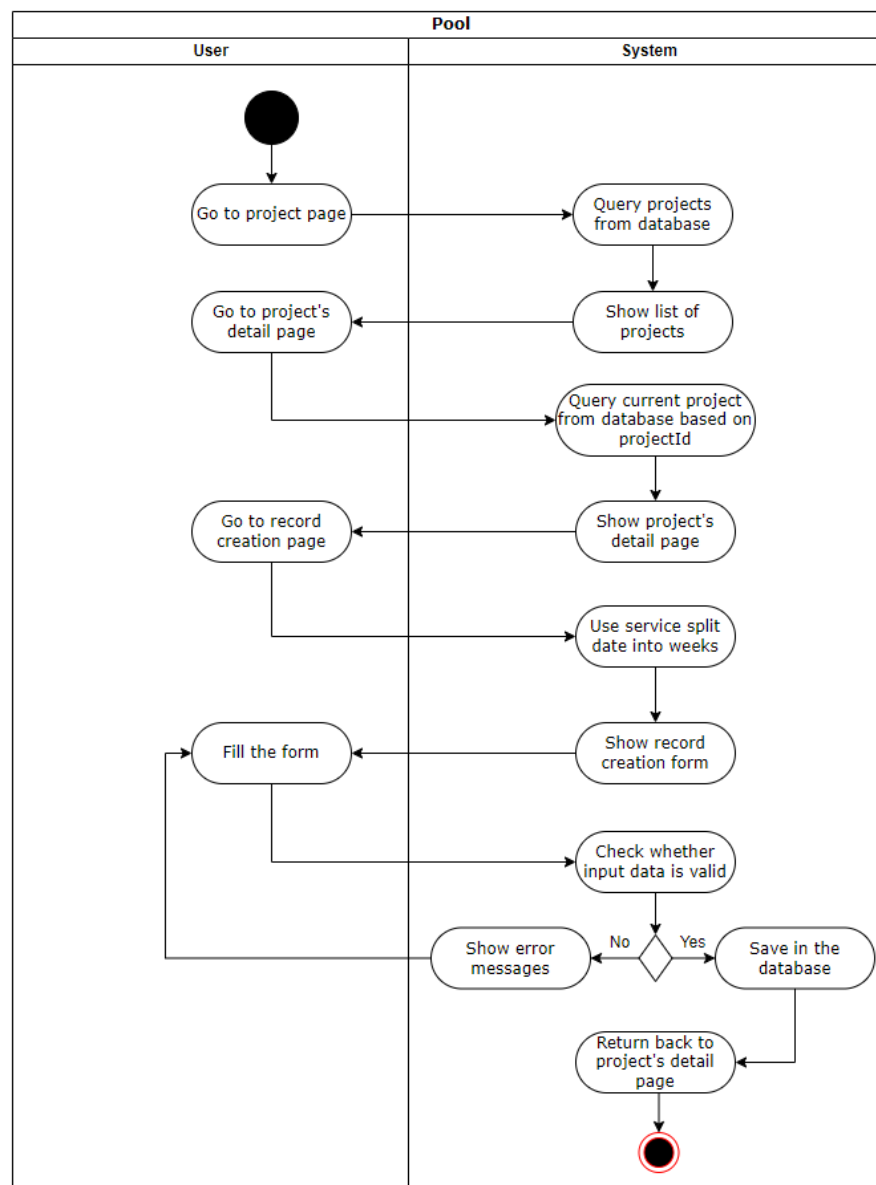


Figure 3 – Record creation

Records are built based on the project, that's why the user needs to visit the project's detail page first. And for helping teachers to create records for each week, a special service which can

split project's start time and deadline into weeks is needed. So, teachers can see both the range of dates and week numbers at the same time.

1.4 Problem domain model

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

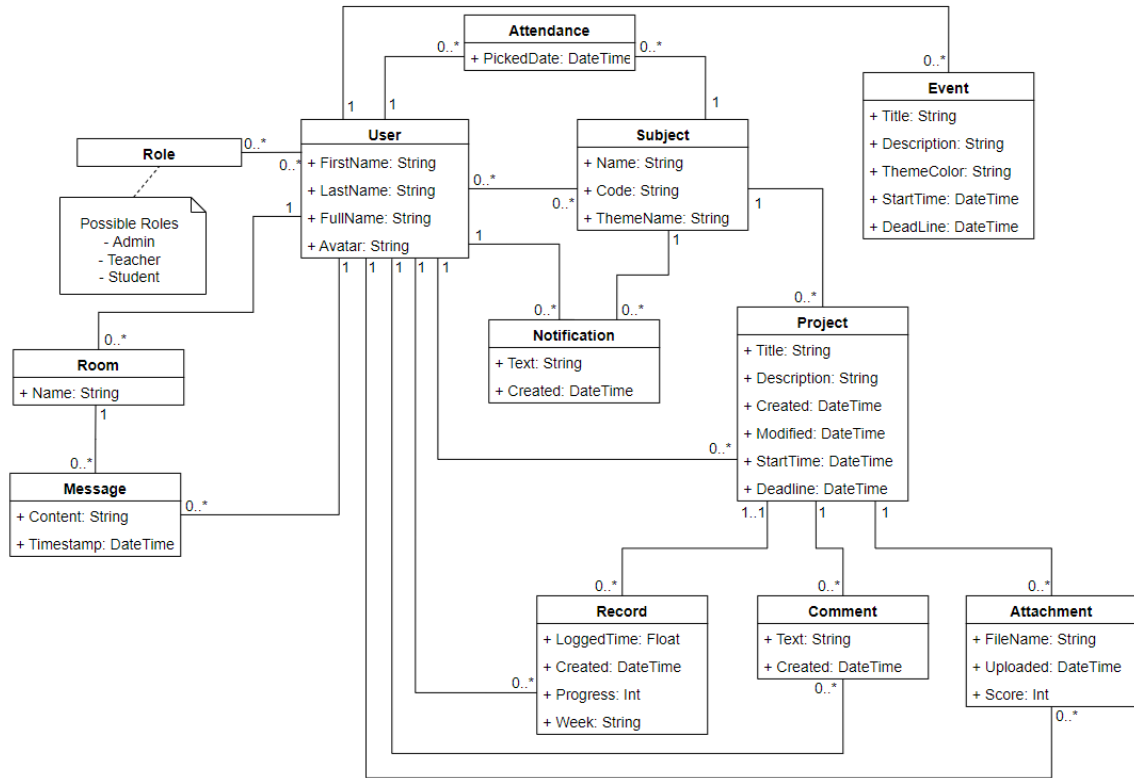


Figure 4 – Problem Domain Model

As can be seen from the diagram. In the system, almost all models are built based on the subject. In the logic of how the system works, users are required to join some subjects and subjects contain different roles' users. That's why the relationship between them is many to many. For those users that join the subjects, teachers can specifically create attendance, records for them. And sure, those users can create their own comments and upload personal attachments in the projects like mentioned before in the functional requirements. Also, unlike some traditional chat rooms. Users need to join the room first and then they can send messages. In SMS, a simple chat room should be created. Which means users can directly send messages in different rooms instead of needing to join them first. So, due to this reason, in the diagram, the relationship between user and chat room is one to one. It means the creator and chat room.

1.5 Used technologies and tools

Follow the development of programming, 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 the database.

1.5.1 ASP. NET CORE

ASP. NET Core is an application framework which is written in C# and C++. It is developed by Microsoft and currently managed under the .NET Foundation (a non-profit open source organization). Using ASP. NET Core, developers can build Web applications and services, Internet of Things (IoT) applications, and mobile backends.

The major characteristics of ASP. NET Core:

- Cross-platform support

Applications built with ASP. NET Core can be run and its code can be reused instead of thinking about platform target. It currently supports three operating systems which are Windows, Linux and MacOS. The number of supported operating systems and scenarios will increase overtime.

- Flexible deployment

Users can deploy a .NET Core application side by side with their application seamlessly. It supplies two types of deployments for .NET Core applications which are framework-dependent deployment and self-contained deployment. 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 through NuGet. In other

words, .NET Framework is a large assembly that contains most of the core functionalities and .NET Core is made available as smaller feature-centric packages. It enables users to take advantage of the package users want to use rather than including the entire .NET Core framework. It brings performance boost, tighter security and service reduction.

- Open-source

.NET Core is open source using MIT and Apache 2 licenses and is available in GitHub. Open source enables speed and facilitates the exchange of human knowledge and skills. It attracts a large group of users and continue to enable a great community. With it, frameworks errors can be found quickly and improved.

1.5.2 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 (Figure 5).

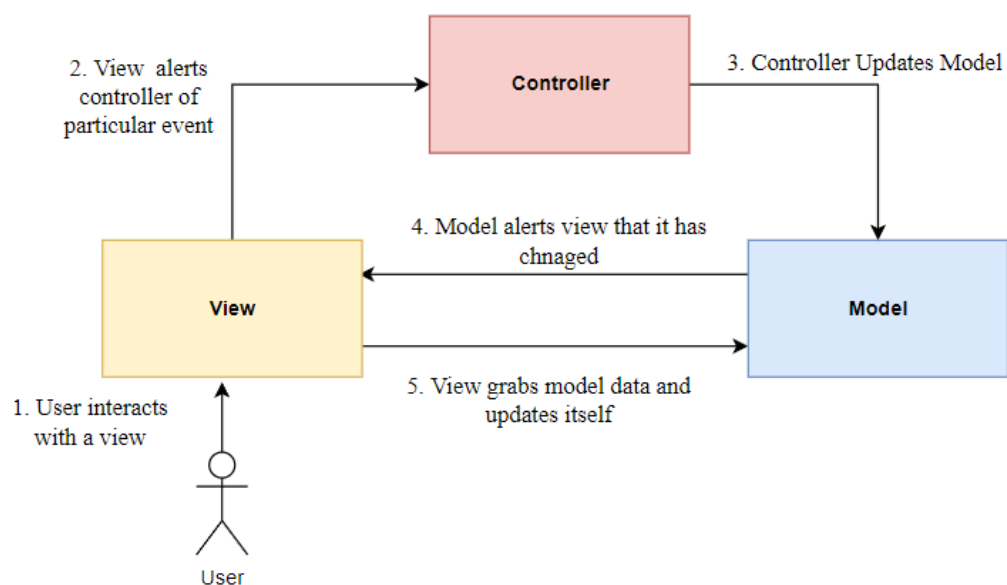


Figure 5 – MVC Architectural Diagram

ASP. NET Core MVC always works together with 3-tiers architecture. The purpose of layering is to achieve the idea of "high cohesion and low coupling", which is conducive to the maintenance, updating or porting of the system later on. From a personal point of view, MVC just subdivides the UI layer into three more tiers (Figure 6).

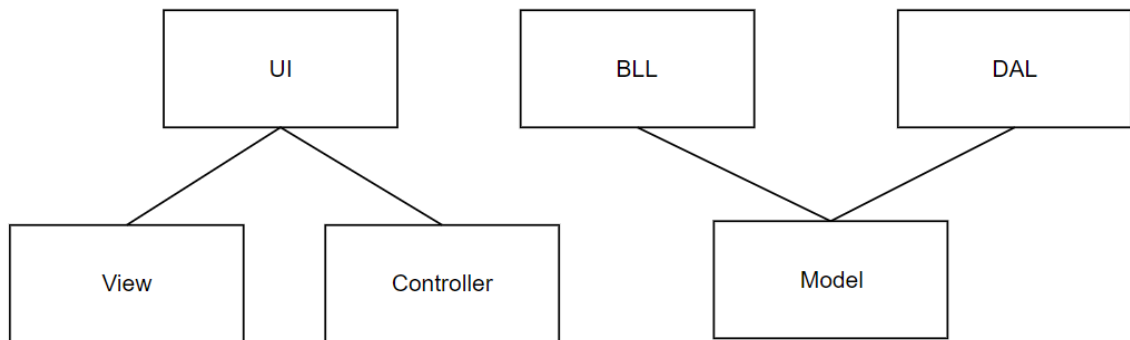


Figure 6 – MVC and 3-tiers architecture

Model Responsibilities:

For the Model, the main thing is to save and export information. Data, behavior and methods are the main elements of the Model. The Model is where the logic is most complex because the 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. Everything that is not relevant to the display interface should not appear inside the view. In other words, complex judgement statements and complex arithmetic procedures should generally not be present in the view. There can be simple looping statements, formatting statements. View Component , ViewModel or view template can also be used to simplify the view.

Controller Responsibilities:

Controller is the component that responds to user requests, decides what view to use and what data needs to be prepared for display. It acts as a mediator between view and model. In an MVC application, the view only displays information; the controller handles and responds to user input and interaction. The Controller should be limited to fetching the user request data and should not have any manipulation or pre-processing of the data

1.5.3 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 (Figure 7).

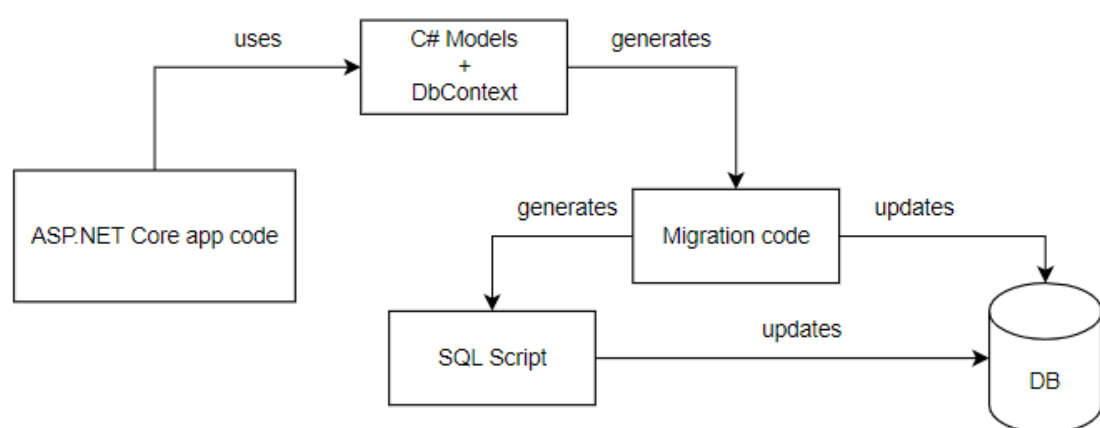


Figure 7 – EF Core working logic

Main Reasons why Entity Framework core was used:

- The visual interface reduces the difficulty and time lost in creating entities.

- The database sync is easy for environments using migration. It is a really needed.
- Feature which makes it possible to upgrade or downgrade any change/commit.
- No more worrying about how the application connects to the database.
- Instead of requiring users have good knowledge in SQL scripts, users can easily control database by LINQ queries even they are green hand with it.

1.5.4 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 used:

- Fewer Cross browser bugs.
- A consistent framework that supports major of all browsers and CSS compatibility fixes.
- Good documentation and community support. The official documentation explains the use of each component in detail.
- A large selection of free and professional templates. Developers can find and download a lot of free and professional templates that suit their website which very helpful to confirm the general style.
- Grid system. It is a pain for frontend developers to implement page adaption for different sizes of screens. But the bootstrap grid system can automatically scale up to 12 columns as the device or viewport size increases (Figure 8). Grid system has four pre-denied classes for different viewports (Figure 9), the principle is to use media queries to obtain the dimensions of the user's device. And the pages are automatically adapted to the different devices.

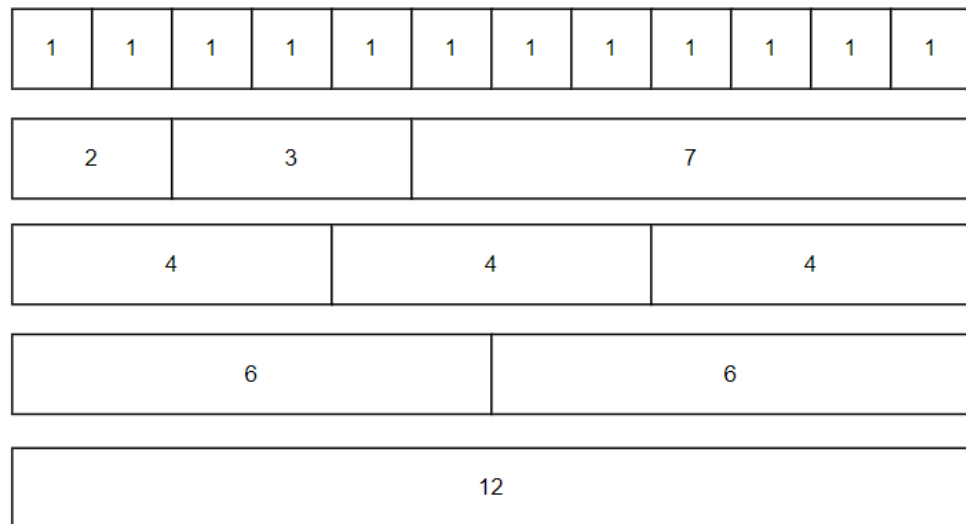


Figure 8 – Bootstrap grid system

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
# of columns	12				
Gutter width	30px (15px on each side of a column)				
Nestable	Yes				
Column ordering	Yes				

Figure 8 – Bootstrap grid options

1.5.5 jQuery

jQuery is a free and open-source JavaScript library. The motto of it is “write less, do more”. It simplifies the Document Object Model (DOM) operations compare to the native JavaScript code by encapsulation the common functionalities like elements acquisition, events binding, styles modification and so forth.

Not just these, jQuery also make the operations like animation, ajax much simpler and good cross browser support.

Main Reason why jQuery was used:

- Simple and clean code. Benefits from the excellent DOM operations encapsulation, jQuery can easily complete various operations that were originally very complex, it reduces the lengthy JavaScript code that was needed before (Figure 10). Developers can be handy to write DOM operations related programs, and also allow JavaScript novices to write excellent programs.

JavaScript	jQuery
<code>document.getElementById("elementId").value()</code>	<code>\$("#elementId").val()</code>

Figure 10 – Code comparison between JavaScript and jQuery

- Perfect Ajax. jQuery encapsulates all Ajax operations into a function \$. ajax, which allows developers to concentrate on processing business logic when dealing with Ajax without having to worry about complex browser compatibility and creation and use of XMLHttpRequest objects.
- Rich plug-in support. jQuery supplies a lot of powerful plugins the system needs like rich text editor, full calendar and so forth.

1.5.6 Microsoft SQL Server

Microsoft SQL Server is a relational database management system launched by Microsoft. It has the advantages of ease of use, good scalability and high integration of related software. Microsoft SQL Server is a comprehensive database platform that uses integrated business intelligence (BI) tools to provide enterprise-level data management. The Microsoft SQL Server database engine provides more secure and reliable storage capabilities for relational and structured data, allowing users to build and manage high-availability and high-performance data applications for the business.

Main Reason why Microsoft SQL Server was used:

- Easy Installation and free. Same as the IDE Microsoft Visual Studio, MSSQL is also a product from Microsoft. Those two tools have perfect compatibility, users can quickly install MSSQL through NuGet in Visual Studio and establish connection between them.

2 Implementations

2.1 Class diagram

Based on the selection of tools, the final class diagram is obtained (Figure 11).

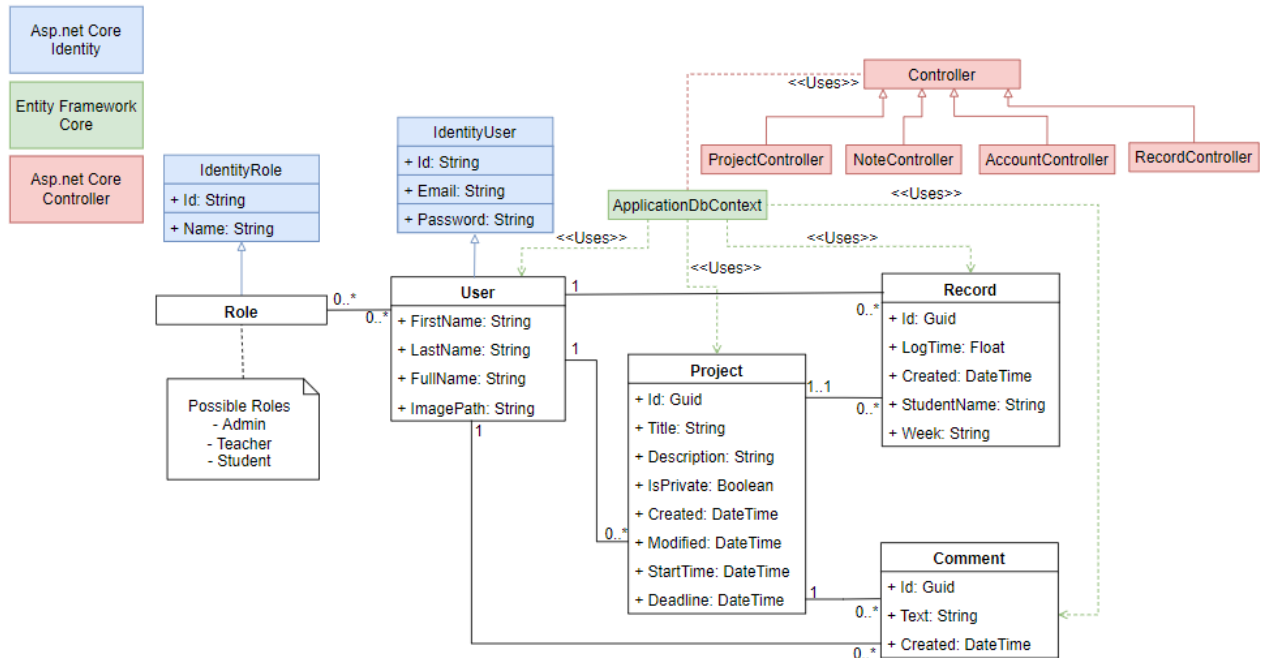


Figure 11 – Class Diagram

Due to the page size and thinking of readability. Only important models were picked from the problem domain model to show the logic of how the holistic system works. The rest of the models follow the same logic as models shown here.

User model and role model can easily inherit necessary properties they need from .NET Core Identity instead of adding by themselves. All those models will be added in ApplicationDbContext instance which from EF Core as entities. Controller uses ApplicationDbContext instance to implement CRUD activities of each entity.

2.2 Preparation

To start the development, VS should be installed on a local device. Also, after creating a new project on the VS. Some basic necessary packages should be downloaded from NuGet.

The project needs *Microsoft.EntityFrameworkCore.SqlServer*, *Microsoft.EntityFrameworkCore.Tools*, *Microsoft.Identity.EntityFrameworkCore*.

2.3 Model creation

Following the problem domain model diagram above, domain models can be easily created.

Domain model is a collection of objects that represents the data in the database, it usually has a one to one relationship with the tables in the database. And below was an example about subject model creation (Code sample 1).

```
public class Subject
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public string Name { get; set; }
    public User Creator { get; set; }
    public DateTime Created { get; set; }
    public string Code { get; set; }
    public string ThemeName { get; set; }
    public ICollection<Project> Projects { get; set; }
    public ICollection<Attendance> Attendances { get; set; }
    public ICollection<Notification> Notifications { get; set; }
    public ICollection<UserSubject> UserSubjects { get; set; }
}
```

Code sample 1 – Create User Model

As said before, domain models represent the data in the database. So, they only define the properties of the data to be stored in the database. Reasons why use properties rather than fields are having properties could make some things easier especially when working with reflection and fields are only good for private variables. Also, extra configurations are needed to use fields in EF Core. For the relationship between each model, 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. In the above code, Creator is a reference navigation property because subject is created by user. And for Projects, Attendances, Notifications, they are collection navigation properties because they are based on the Subject entity.

2.4 ApplicationDbContext creation

After finishing models' creation, it is time to create ApplicationDbContext class (Code sample 2).

```
public class ApplicationDbContext : IdentityDbContext<User>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}
```

```

public DbSet<Subject> Subjects { get; set; }
public DbSet<Project> Projects { get; set; }
public DbSet<Notification> Notifications { get; set; }
public DbSet<Attendance> Attendances { get; set; }
public DbSet<Note> Notes { get; set; }
public DbSet<Record> Records { get; set; }
public DbSet<Message> Messages { get; set; }
public DbSet<Room> Rooms { get; set; }
public DbSet<Attachment> Attachments { get; set; }
public DbSet<UserSubject> UserSubjects { get; set; }
public DbSet<Event> Events { get; set; }
}

```

Code sample 2 – Create ApplicationDbContext Class

A class that derives from the *IdentityDbContext* class (a regular *DbContext* with two *DbSets*. One for users and the other for roles.) was created and be named as *ApplicationDbContext*. For the *ApplicationDbContext* class to be able to do any useful work, it needs an instance of the *DbContextOptions* class which can carry configuration information such as the connection string, database provider to use etc. For each entity in the model, class includes a *DbSet<TEntity>* property. The LINQ queries against the *DbSet<TEntity>* will be translated into queries against the underlying database (Figure 12).

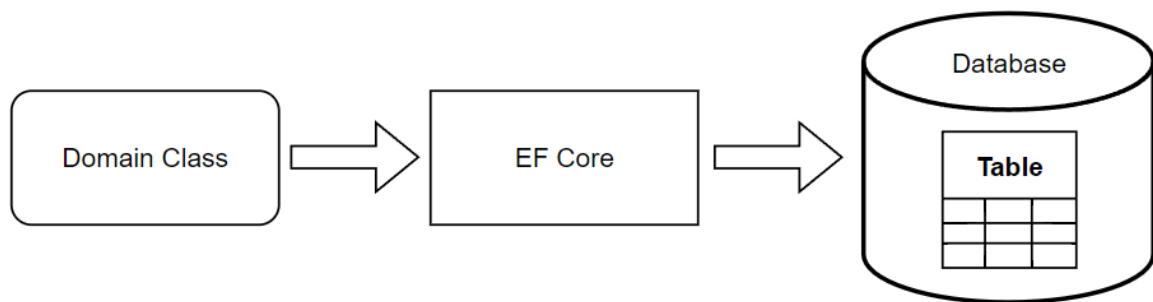


Figure 12 – Database creation from the domain classes

As mentioned in the activity diagram about admin manages users (Figure 2). The delete action in this site is different compared to traditional sites. But in EF Core, by default the foreign keys in the parent table have cascade delete behavior which means when a record is removed from the parent table, then the corresponding records in the child table are deleted automatically. To avoid it, in the *OnModelCreating* method of *ApplicationDbContext* class, the delete behavior of foreign keys need to be modified (Code sample 3).

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{

```

```

base.OnModelCreating(modelBuilder);
foreach (var foreignKey in modelBuilder.Model.GetEntityTypes()
    .SelectMany(e => e.GetForeignKeys()))
{
    foreignKey.DeleteBehavior = DeleteBehavior.Restrict;
}
}

```

Code sample 3 – Modify foreign keys delete behavior

2.5 Record creation

Before talking about how to create records, a diagram about how CUD actions work between EF Core and database was made (Figure 13).

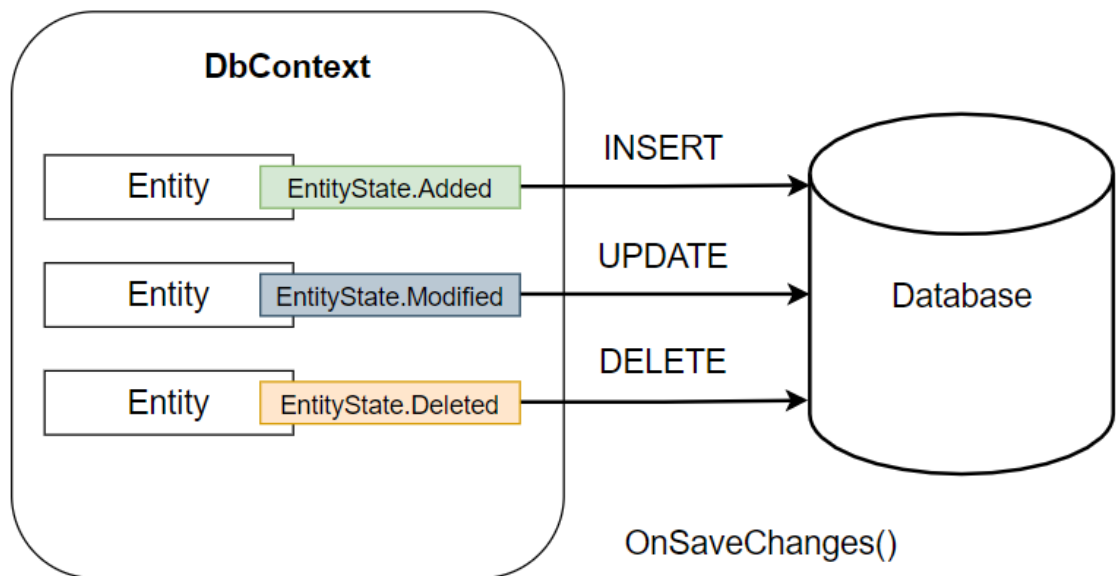


Figure 13 – Working logic between EF Core and Database

In the project's properties have start time and deadline. For matching the requirement of record creation diagram (Figure 3), a special class which can split days between start time and deadline should be created (Figure 14).

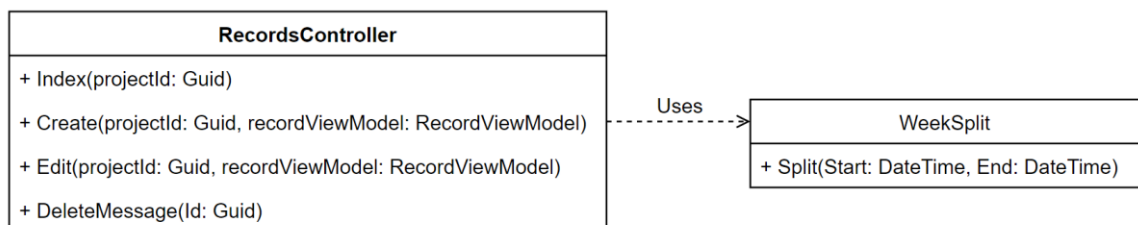


Figure 14 – Week split class

The view model is the object which holds the data that needs to be shown, modified or inserted to the user in the view page. It can be thought as a DTO (Data transfer object) which is used as a container to encapsulate data and pass it from one layer of the application to another. View model is related to the presentation layer of the application which matches the usage of DTO to return data back to the presentation layers, it makes the application more secure as it can avoid exposing the potentially dangerous properties. With data annotations, programmers can decorate and validate the properties easily (Code sample 4).

```
public class RecordCreateViewModel
{
    [Required]
    public string StudentId { get; set; }
    [Required]
    public float LogTime { get; set; }
    [Required]
    public string Week { get; set; }
    [Required]
    public string Note { get; set; }
    [Required]
    public int Progress { get; set; }
}
```

Code sample 4 – RecordCreateViewModel

In the RecordsController, the record creation needs two versions of the create method. One that renders the record creation form by HttpGet (Code sample 5) and the other handles the request when this creation form is submitted by HttpPost.

```
[HttpGet]
public async Task<IActionResult> Create(Guid? projectId)
{
    if(projectId == null)
    {
        return View("~/Views/Shared/NotFound.cshtml");
    }
    var project = await this.context.Projects
        .Include(x=>x.Subject).ThenInclude(x=>x.UserSubjects)
        .ThenInclude(x=>x.User)
        .SingleOrDefaultAsync(x => x.Id == projectId);

    var getWeek = Week.Split(project.StartTime, project.DeadLine);
    ViewBag.Week = new SelectList(getWeek);
    return View(new RecordCreateViewModel());
}
```

Code sample 5 – Create method in HttpGet

ASP. NET Core MVC supplies an interior process which is model binding. It is the process controller and razor pages work with data that comes from HTTP request. Model binding can get

information directly from forms or URL routing (Figure 15). Project's id in the above code is the data sent from the project's detail page by using anchor tag helper *asp-route*.

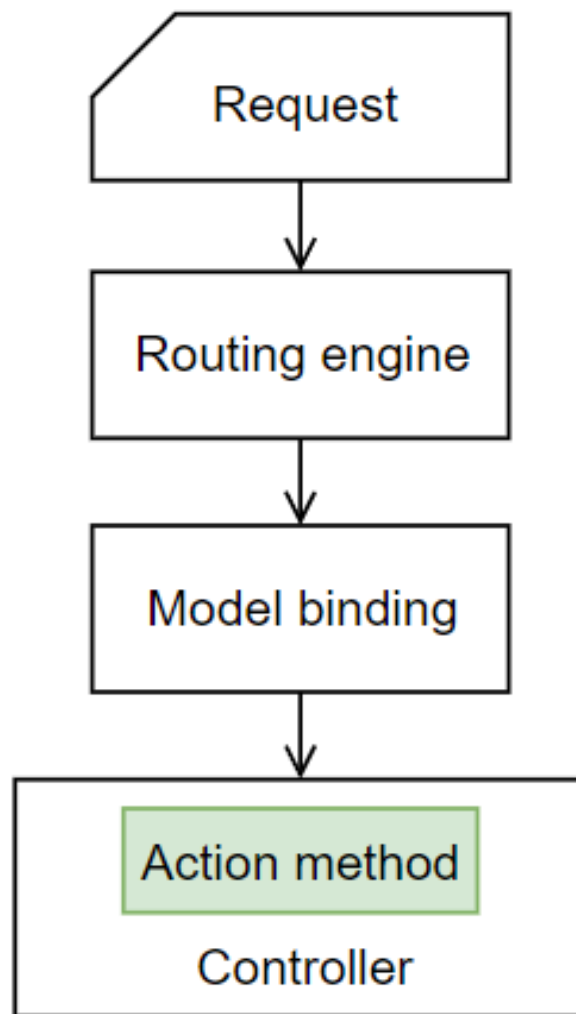


Figure 15 – Model binding

The record is based on the project, so the method needs to check whether project id is sent together when users visit the record creation page. EF Core queries project table for getting the project which matches this project id. If either of them equals null, the user will be sent to the NotFound page. Due to the record and user are two separate entities, the *include* method from EF Core was used to specify the related objects to include in the query result.

For sending temporary data from controller to the view, ASP. NET Core MVC supplies a dynamic property called *ViewBag* (Figure 16).

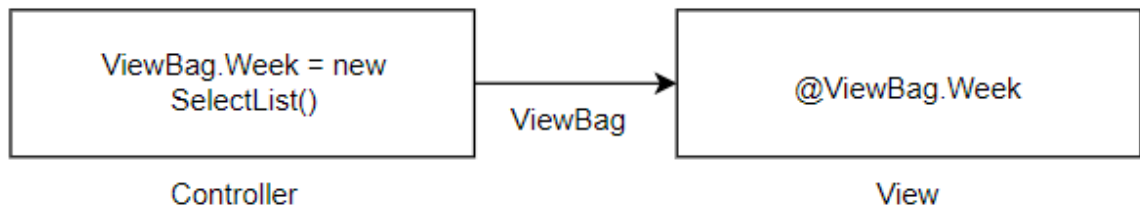


Figure 16 – Data sending between controller and view by ViewBag

After the building of front page, the record creation form is done (Figure 17).

Create Record

Student

Spent Time (Hours)

Week

Progress(%)
 1

Note

Figure 17 – Record creation form

In the `HttpPost` create action, `ModelState.IsValid` method (Code sample 6) which represents errors that come from model binding and model validation was used to judge whether user's input data is matching the *DataAnnotations* which developers configured in the *ViewModel* (Figure 18).

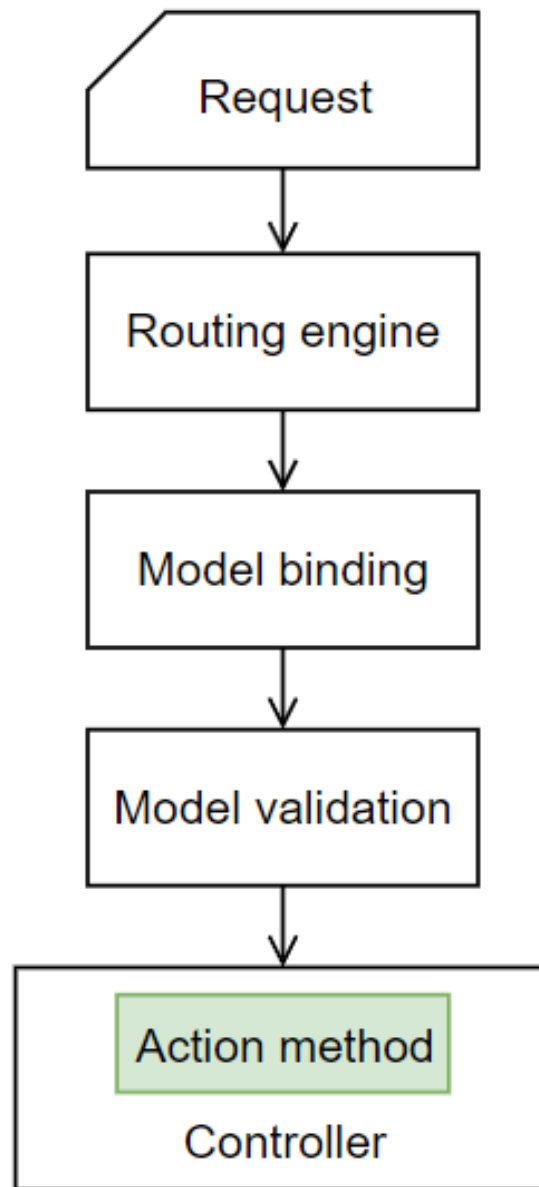


Figure 18 – *ModelState.IsValid* method

```
if (this.ModelState.IsValid)
```

Code sample 6 – Model validation in create action

If all user's input data is valid, EF Core will insert a new record in the record diagram as the diagram shows (Figure 13).

2.6 Localization

As said in non-functional requirements, the site supports two languages which are simplified Chinese and English. Before starting to create the resource files, some settings which related to localization need to be configured in *Startup.cs* file (Code sample 7).

```

services.AddLocalization(opt => { opt.ResourcesPath = "Resources"; });
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 7 – Configure localization files

For the path of the resource files, in the application root directory a folder which was named as Resources should be created and service uses *opt.ResourcesPath* to find it. *CultureInfo* class is for storing culture-specific information, so here it was used to define the allowed cultures.

After resource files be created (Figure 19, Figure 20), it will be injected to the view page and replace the original text (Code sample 8).

	Name	Value
▶	notice	You don't have permission to view this resource
	title	Access Denied
*		

Figure 19 – Resource file in English

	Name	Value
▶	notice	你没有权限访问当前页面
	title	拒绝访问
*		

Figure 20 – Resource file in simplified Chinese

```

@inject Microsoft.AspNetCore.Mvc.Localization.IViewLocalizer localizer
<div class="text-center">
    <h1 class="text-danger">@localizer["title"]</h1>
    <h6 class="text-danger">@localizer["notice"]</h6>
    
</div>

```

Code sample 8 – Inject localization file in view page

To be more user friendly, the system is supposed to save users' language selection results inside the cookie, so when users visit the different page, they don't need to switch language again.

To achieve it, a method called *CultureManagement* was created in the HomeController (Code sample 9).

```
[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 9 – CultureManagement action in HomeController

In general, cookie is a data carrier of only 4 kb in size that saves in the user's local browser. When users first time visit a site, the server sends a *HttpResponse* which includes Set-Cookie in the header to clients' side. The browser saves this cookie in the local and every subsequent *HttpRequest* will automatically carry this cookie (Figure 21).

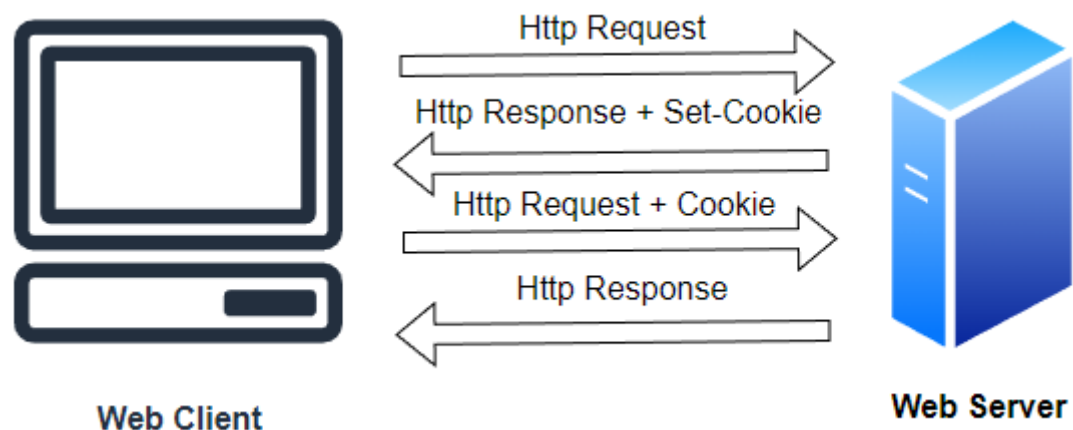


Figure 21 – The process about set cookie

The *CultureManagement* action expects to receive two parameters – culture and returnUrl. Culture means the language which users selected and returnUrl can help the site back to the page where users send this action.

Then in the razor page, after the code about switch button was written (Code sample 10), implementation about localization is done.

```
@{
    var returnUrl = string.IsNullOrEmpty(Context.Request.Path) ? "~//" :
    $"{Context.Request.Path.Value}{Context.Request.QueryString}";
}
<form id="switchEn" asp-action="CultureManagement" asp-controller="Home" method="post"
asp-route-returnUrl="@returnUrl">
```

```

<input type="hidden" name="culture" value="en" />
<a class="collapse-item"
href="#" onclick="document.getElementById('switchEn').submit();" >English</a>
</form>

```

Code sample 10 – Html and JS code about switch button

2.7 Login/register with social media

SSM supports user login/register with their Google or Facebook account. The benefits of it are that the system no longer needs to store and maintain those highly sensitive information such as the password in the database and simplified registration steps (users no longer need to input email or other personal information that the system can get from social media providers). Instead, it is the responsibility of those external authentication providers.

For showing clearly how different components in the site work with the external login. A sequence diagram about it was made (Figure 22).

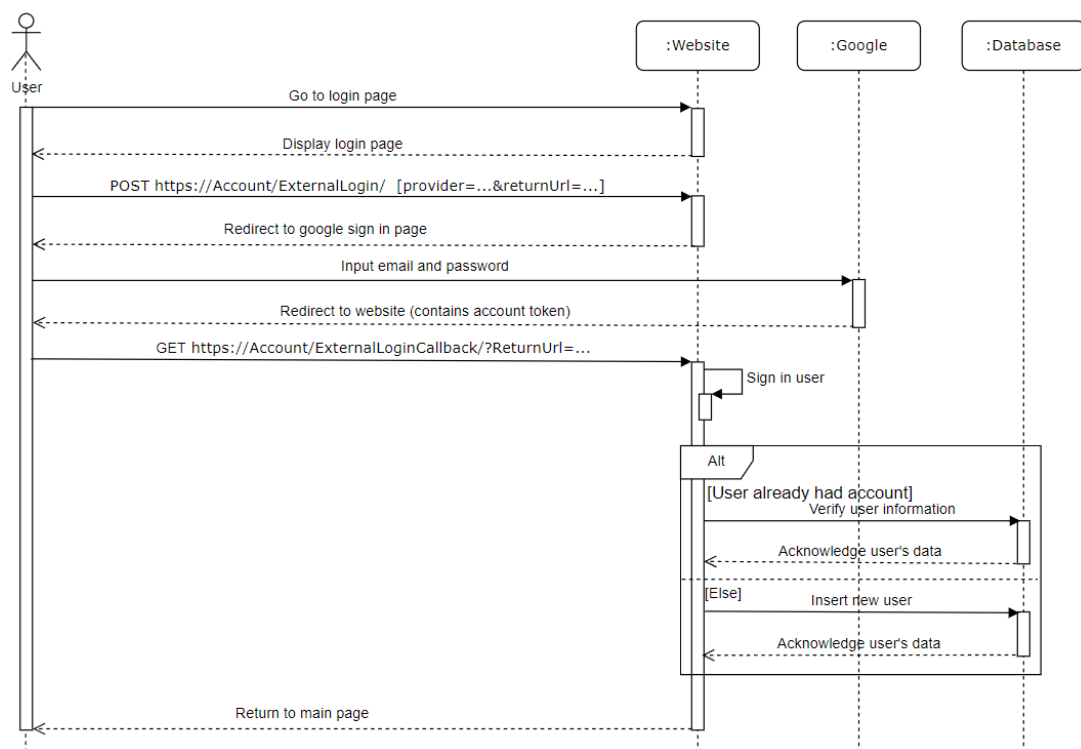


Figure 22 – Sequence diagram about external login

Before starting on the material code, two packages which are *AspNetCore.Authentication.Facebook* and *AspNetCore.Authentication.Google* should be installed from the NuGet. Then in the *Startup.cs* file, authentications about Google and Facebook need to

be configured which are the client id and client secret from their OAuth client credentials (Figure 23).

Client ID	324353359328-0kjj[REDACTED]apps.googleusercontent.com
client secret	1lQeKN[REDACTED]yz0V02f
Date created	22 Oct 2020 GMT+7 19:15:21

Figure 23 – Client ID from login provider

For displaying each available external login way, just need to traverse all available providers from the *AuthenticationScheme* which in *Microsoft.AspNetCore.Authentication* namespace in the login view to get those data which registered (Code sample 11).

```
@foreach (var provider in Model.ExternalLogins)
{
    if (provider.Name == "Facebook")
    {
        .....
    }
}
```

Code sample 11 – Traverse and display all available login ways

After users click one of those social media buttons. Undoubtedly, they will see an error and that is page not found. Because the current system still doesn't have *ExternalLogin* action which can redirect users to the external login provider login page like the picture show (Figure 24).

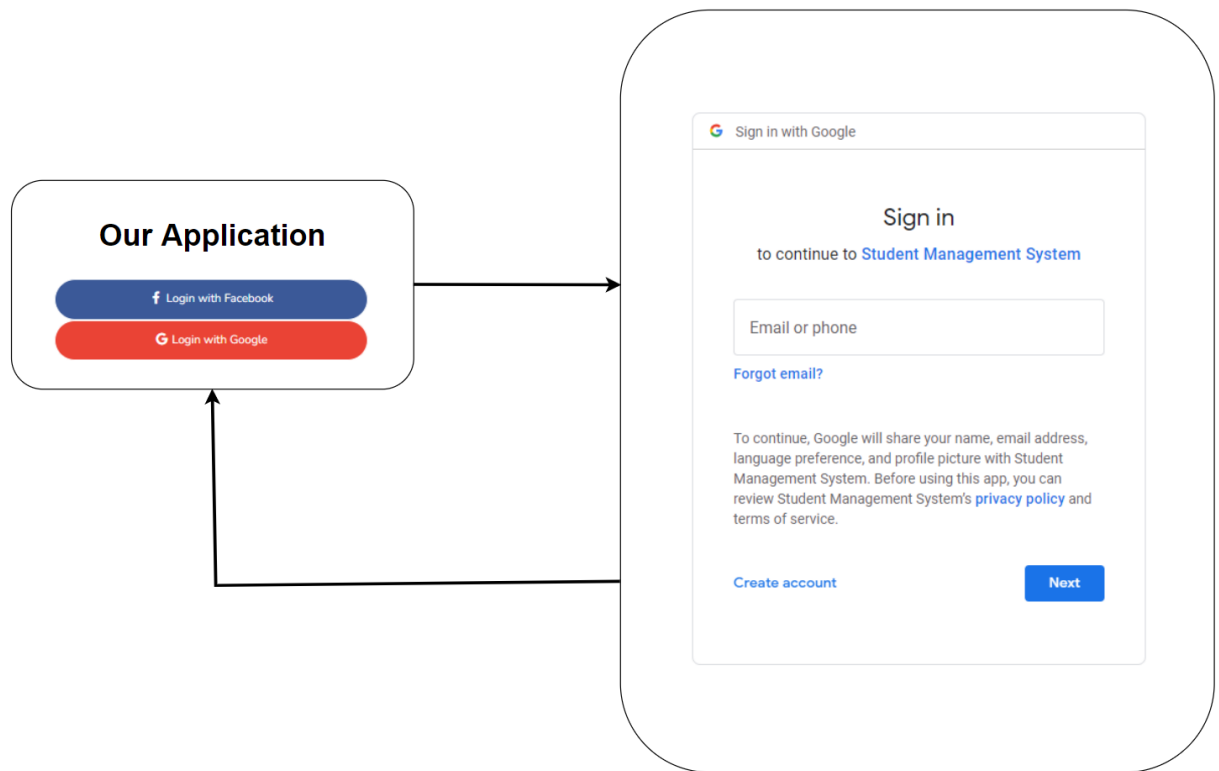


Figure 24 – Redirect users to the external login page

But that's still not the whole action, because users just input their email and password and send to those external login providers, the system doesn't have any action which can receive the data which providers return. Also, as the picture 13 shows, we still need to judge whether users exist or not in the database. So, to tackle those situations, ExternalLoginCallback action is needed. For helping understand the interior logic of this action, a process flow diagram was made (Figure 25).

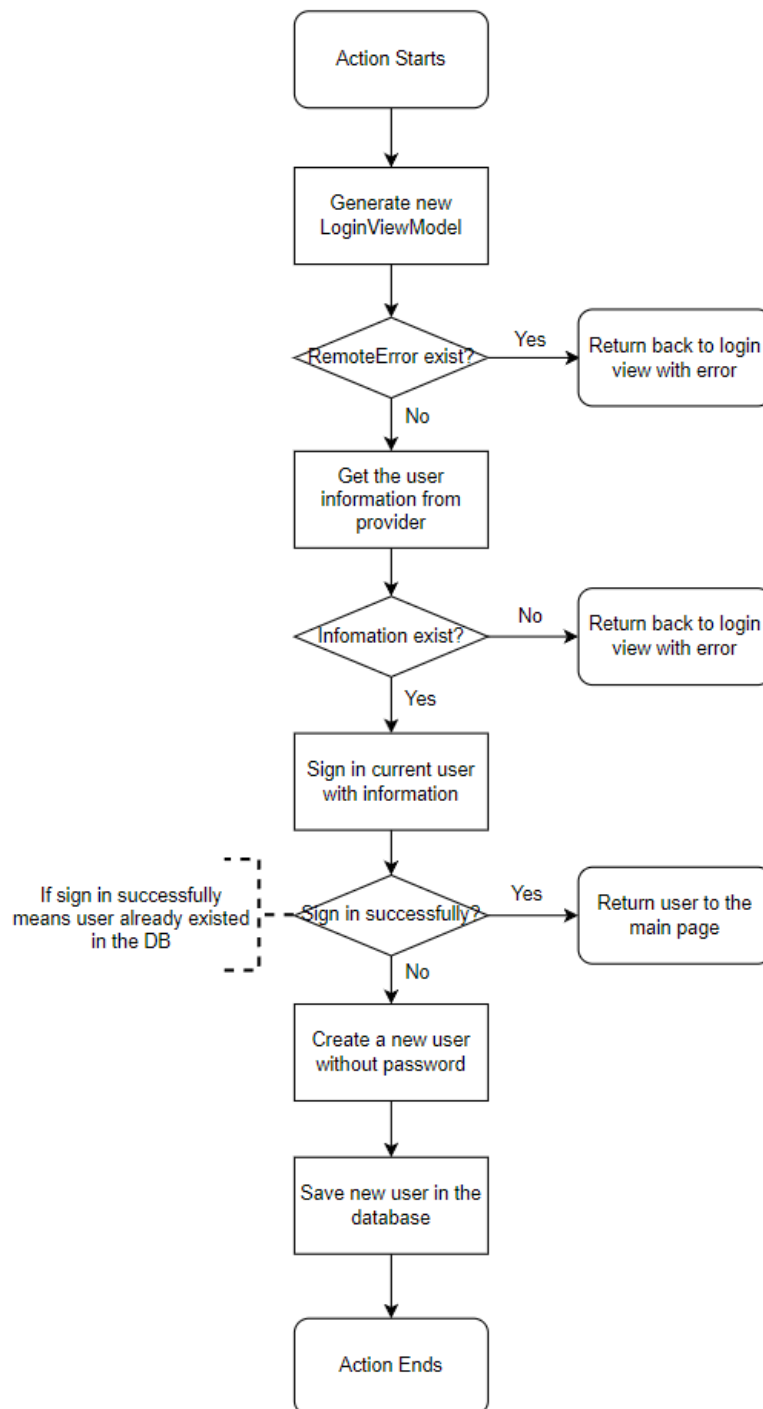


Figure 25 – ExternalLoginCallback action process flow diagram

As default, the ExternalLoginCallback action receives two parameters which are returnUrl and remoteError. First the system will judge whether returnUrl is null, but due to the logic that all users must login so they can visit their main page and each controller has *[Authorize]* attribute, so it is 100% percentage that this value is null. For the remoteError, if it's not null, probably the

system receives some error information from the external provider which means action is failed so the user will be redirected to the login page (Code sample 12).

```
if (remoteError != null)
{
    ModelState.AddModelError(string.Empty, $"Error from external provider: {remoteError}");
    return View("Login", loginViewModel);
}
```

Code sample 12 – Remote error is not null

AddModelError method from *ModelState* allows us to display additional errors related to specific properties. It can easily add customized error messages to the *ModelState*.

The system accesses the next process for getting user's information from the external provider and then judges whether user's information exists. If it is, the system will try to sign in this user. Due to the result of the sign in action, the system can know whether this user exists in the database. If succeeded means the user already existed in the database and all necessary operations were done so the system can directly stop current action. Otherwise, the system needs to generate a new user account and insert it in the user table .

As a final result, a new user whose *PasswordHash* is null that perfectly fit the system requirements that doesn't save any highly sensitive information shows up in the *dbo.AspNetUsers* table (Figure 26).

	Id	UserName	NormalizedU...	Email	NormalizedE...	EmailConfirm...	PasswordHash
▶	3f2f5ee5-e90...	zhiyuansun70...	ZHIYUANSUN...	zhiyuansun70...	ZHIYUANSUN...	False	NULL

Figure 26 – External login new user

2.8 Real time public chat

For implementing the real time chat functionality, SignalR which provides API for the communication from the server-to-client using Remote-Procedure-Calls was chosen as the tool (Figure 27). It can scale up to handle increasing traffic, send messages to all connected clients simultaneously and handle automatic connection management.

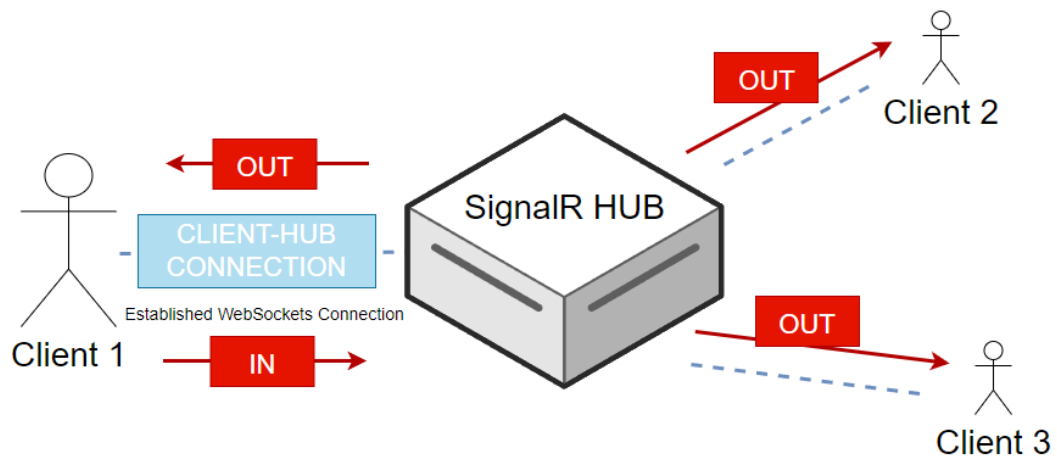


Figure 27 – SignalR working logic

Follow the same logic as code sample 1 and code sample 5, message model and message view model were created. However, UI layers might fail to async with the entities. Therefore, to map entities to model or view model, AutoMapper was selected. It acts as a mapper between two objects and transforms one object type into another (Figure 28).

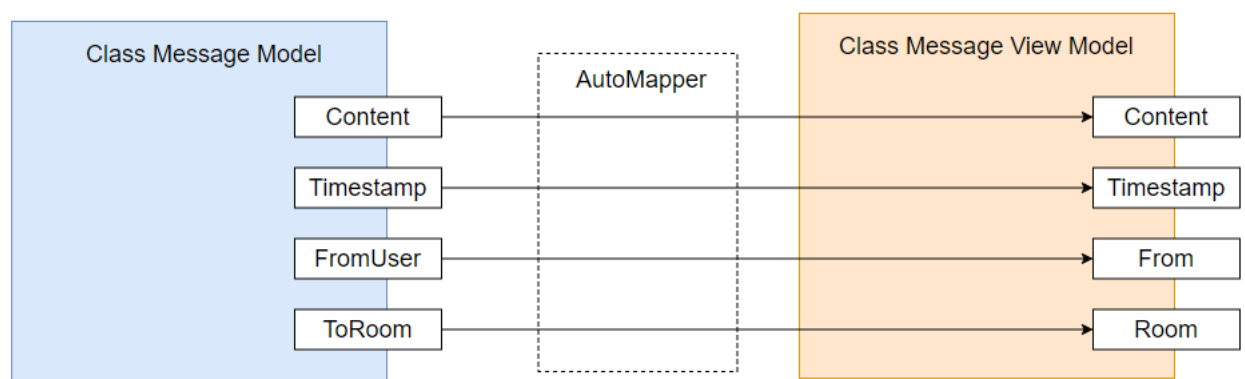


Figure 28 – AutoMapper between message model and message view model

The system used the profile from AutoMapper to configure bilateral properties. So, a class named MessageProfile which inherit profile from AutoMapper was created (Code sample 13).

```
public class MessageProfile : Profile
{
    public MessageProfile()
    {
        CreateMap<Message, MessageViewModel>()
            .ForMember(dst => dst.From, opt => opt.MapFrom(x => x.FromUser.FullName))
            .....
        CreateMap<MessageViewModel, Message>();
    }
}
```

Code sample 13 – MessageProfile class for configuring AutoMapper

For handling client-server communication, a ChatHub class which inherits hub from signalR hub that serves as a high-level pipeline was created. Inside this class, the system will tackle the data user sent and then send the tackled data to the chat room view's JavaScript file for rendering the new DOM element and demonstrate it to those users which connect to this hub (Figure 29).

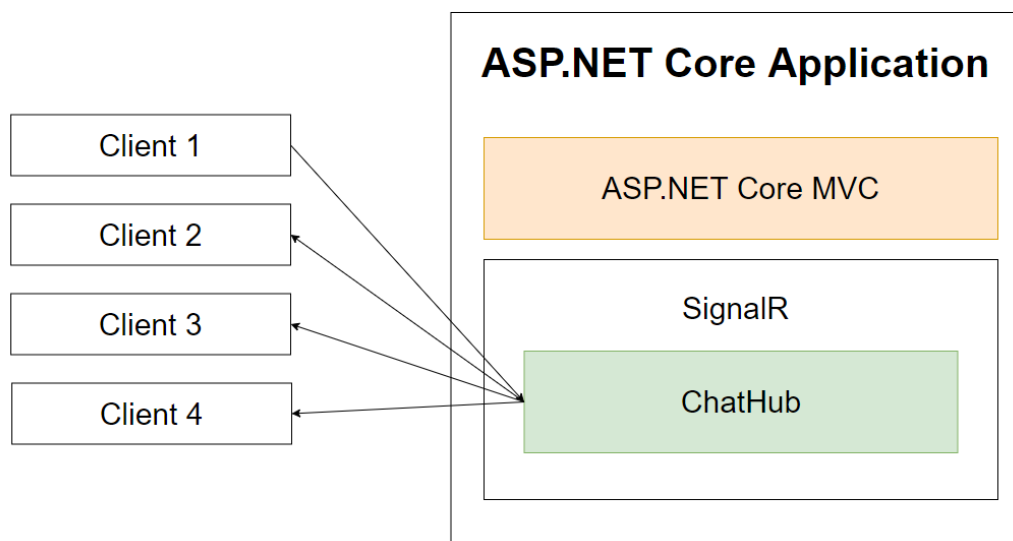


Figure 29 – Chathub class in the system

One of the most common ways to use signalR is write down methods in the *ChatHub.cs* file. But in the system's chat functionality, some actions like send private messages, join/quit the room and get user's device. Those actions will not save any data in the database. Meanwhile, actions like create rooms, send messages, edit rooms, etc. Those actions are related to inserting or removing data from the database. To make a distinction, only those actions which not influence the database will be written in the *ChatHub.cs* file. Vice versa, for those actions related to the database, they will be written in the controller. As a final result, a class diagram has been made to describe the relationship between *ChatHub.cs* and controllers (Figure 30).

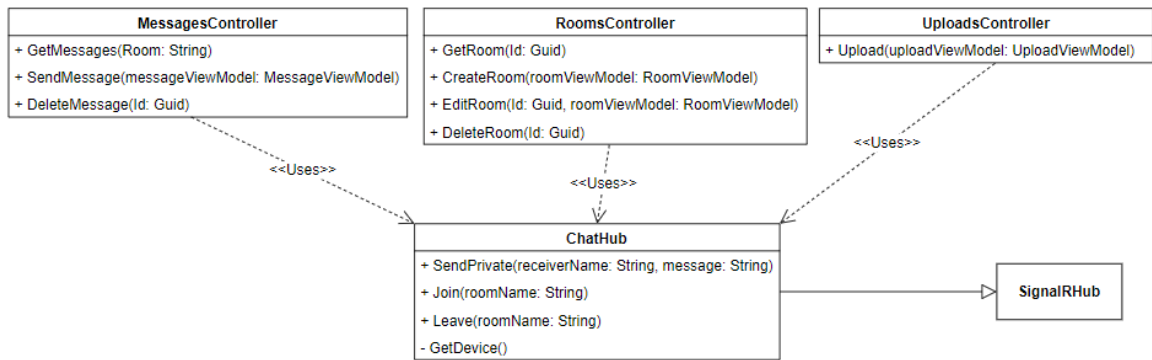


Figure 30 – Class diagram about RealTime chat

Due to the thinking of content space and most of the signalR operations in this project are the same. So below we only talk about the send message functionality which is one of the most important functionalities in this project.

The signalR is a JavaScript client library and real time chat has a lot of DOM actions. Instead of standard MVC controller, API controller was picked for this part. By using API controller, the view can directly receive the JSON data (Figure 31) and based on those data, new elements can be easily created and appended in the view page.

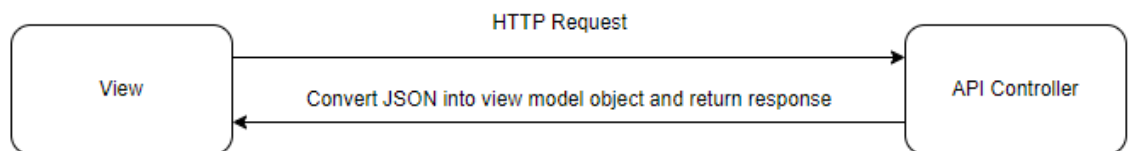


Figure 31 – HTTP request between view and controller

HTTP supplies four methods for helping users send HTTP Requests due to different situations (Figure 32)

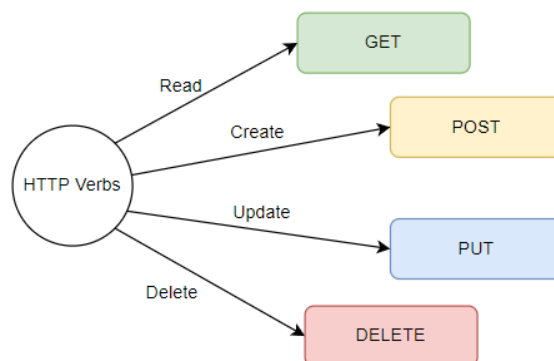


Figure 32 – HTTP verbs

In general, the activity diagram about how users send message can be figured out (Figure 33).

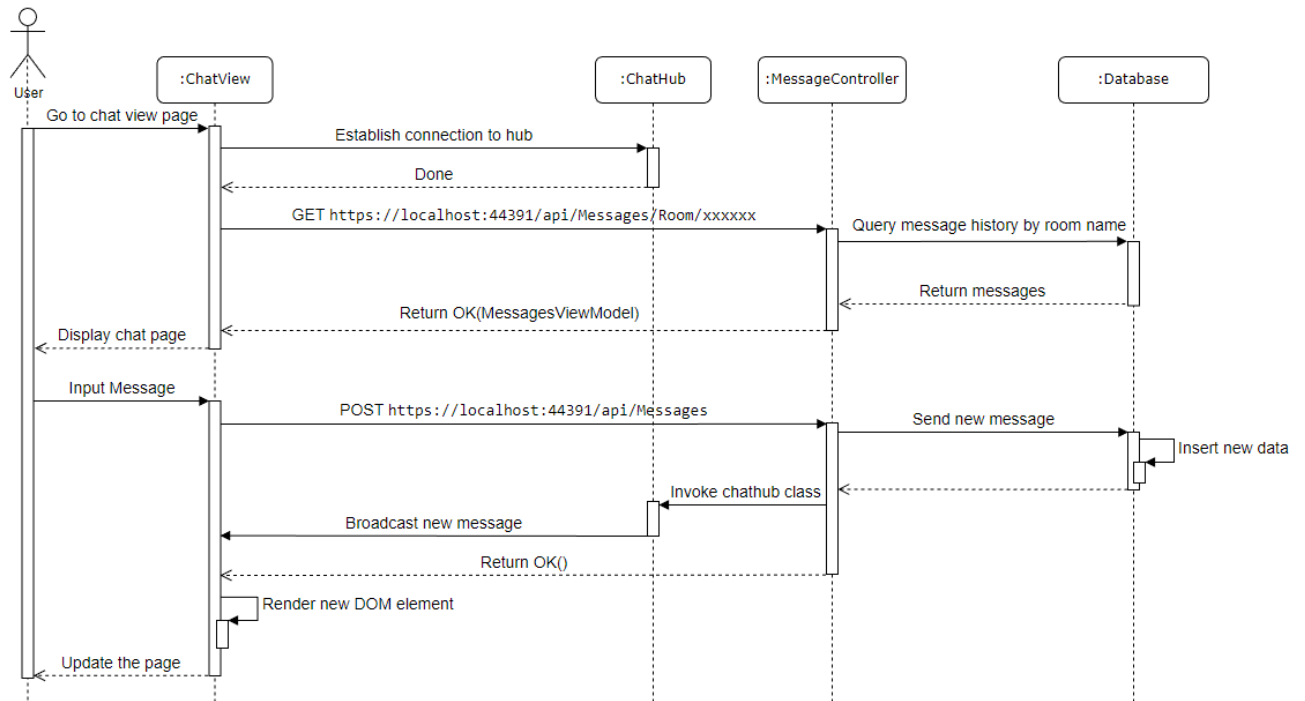


Figure 33 – Sequence diagram about users send message

When users come to chat view page, the interior JavaScript file which named as chat.js in this page will establish connection to the *ChatHub.cs* file by call the *signalR.HubConnectionBuilder* method (Code sample 14).

```
var connection = new signalR.HubConnectionBuilder().withUrl("/chatHub").build();
```

Code sample 14 – Establish connection to the chathub

For getting the history messages in the chat room. An action called *GetMessages* was created in the *MessageController*. It receives one parameter which is *roomName* and based on the *roomName* the system can confirm the room that current user visited and query all history messages in current room (Figure 34).

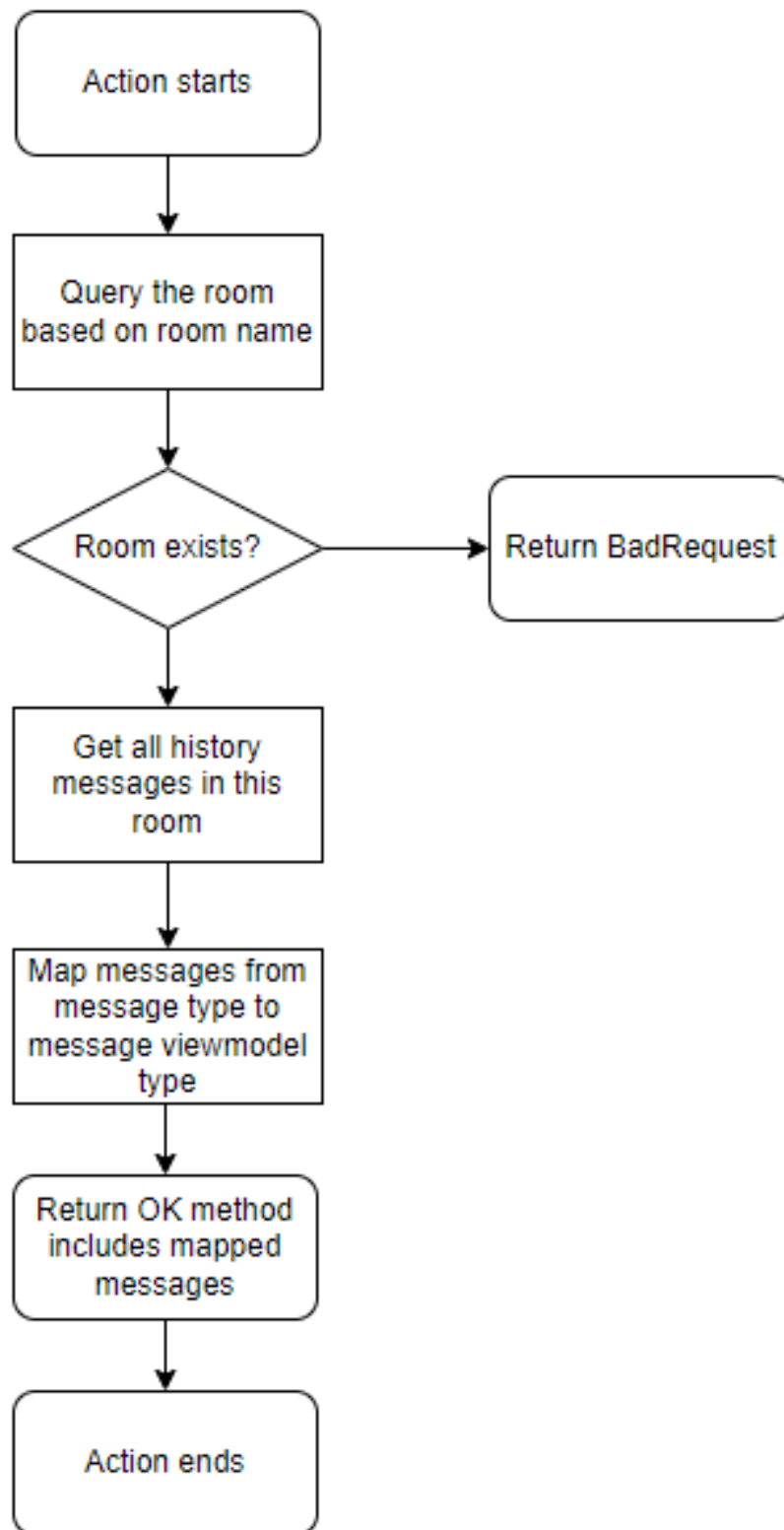


Figure 34 – Process flow diagram about GetMessages action

In the frontend part, for sending HTTP requests to get the data that API controller returned, fetch API from ES5 had been chosen. Fetch API uses the promise method which means it has three possible states which are pending, fulfilled and rejected (Figure 35).

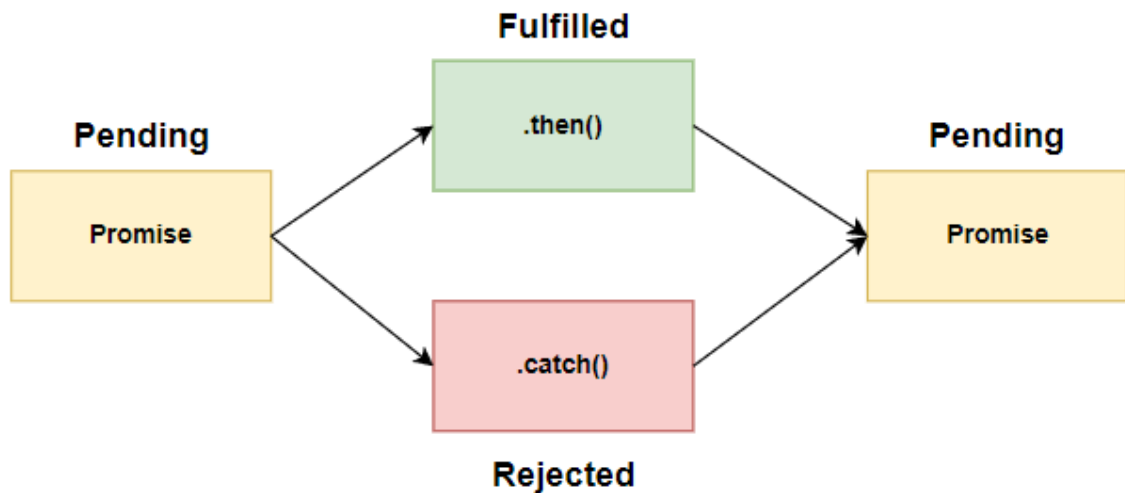


Figure 35 – Promise in JavaScript

Pending is the state until it reaches the fulfilled or rejected, users are in the black box and don't know the outcome. Fulfilled means HTTP request successful, developers can use `.then` method to get the data returned by the server. There are a lot of reasons to cause rejected state like wrong parameters passing, incorrect URL, timeout, etc. The interesting part can be seen from the above picture is no matter the request is successful or failed, it will still return a promise for helping us use promise chain call.

After injecting the *ChatHub* in the message controller's constructor, we can use the methods from *ChatHub* inside message controller (Code sample 15).

```
await _hubContext.Clients.Group(room.Name).SendAsync("newMessage", createdMessage);
```

Code sample 15 – invoke chathub inside message controller

Due to the use of `sendasync` method from *ChatHub*, the new message will be broadcasted to all users which in the same room. In the users' view page, `connection.on` method helps us to access the data which `sendasync` method sent. Through the data, new DOM elements can be rendered and displayed to users (Figure 36).



Figure 36 – New message in the chat room

CONCLUSION

In the course of the work, a web application was created using the ASP. 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.

REFERENCES

1. ASP. NET Core // Microsoft Docs.– [N.p.], 2022. – URL:
<https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0>
(access date 28.04.2022)
2. ASP. NET Core MVC // Microsoft Docs.– [N.p.], 2022. – URL:
<https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-6.0> (access date 28.04.2022)
3. Entity Framework Core // Microsoft Docs.– [N.p.], 2022. – URL:
<https://docs.microsoft.com/en-us/ef/> (access date 29.04.2022)
4. Bootstrap // Bootstrap Docs.– [N.p.], 2022. – URL:
<https://getbootstrap.com/docs/5.1/getting-started/introduction/> (access date 29.04.2022)
5. jQuery // W3Schools.– [N.p.], 2022. – URL:
https://www.w3schools.com/jquery/jquery_intro.asp (access date 30.04.2022)
6. Microsoft SQL Server // Wikipedia.– [N.p.], 2022. – URL:
https://en.wikipedia.org/wiki/Microsoft_SQL_Server (access date 30.04.2022)
7. SignalR // Microsoft Docs.– [N.p.], 2022. – URL:
<https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr>
(access date 02.05.2022)
8. AutoMapper // Pro Code Guide.– [N.p.], 2022. – URL:
<https://procodeguide.com/programming/automapper-in-aspnet-core/> (access date 02.05.2022)
9. External logins // Microsoft Docs.– [N.p.], 2022. – URL:
<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/?view=aspnetcore-6.0&tabs=visual-studio> (access date 03.05.2022)
10. HTTP verbs // REST API Tutorial.– [N.p.], 2022. – URL:
<https://www.restapitutorial.com/lessons/httpmethods.html> (access date 05.05.2022)

APPENDIX A

Login Page:



Welcome Back!

Enter Email Address...

Password

☐ Remember Me

Login

[f Login with Facebook](#)


[G Login with Google](#)

[Forgot Password?](#)

[Create an Account!](#)

Figure 37 - Login page

Registration Page:



Create an Account!

First Name Last Name

Email Address

Password Repeat Password

Register Account

[G Register with Google](#)


[f Register with Facebook](#)

[Forgot Password?](#)

[Already have an account? Login!](#)

Figure 38 - Registration page

Forgot Password Page:



Forgot Your Password?

We get it, stuff happens. Just enter your email address below and we'll send you a link to reset your password!

[Reset Password](#)

[Create an Account!](#)
[Already have an account? Login!](#)

Figure 39 - Forgot password page

Main Page:

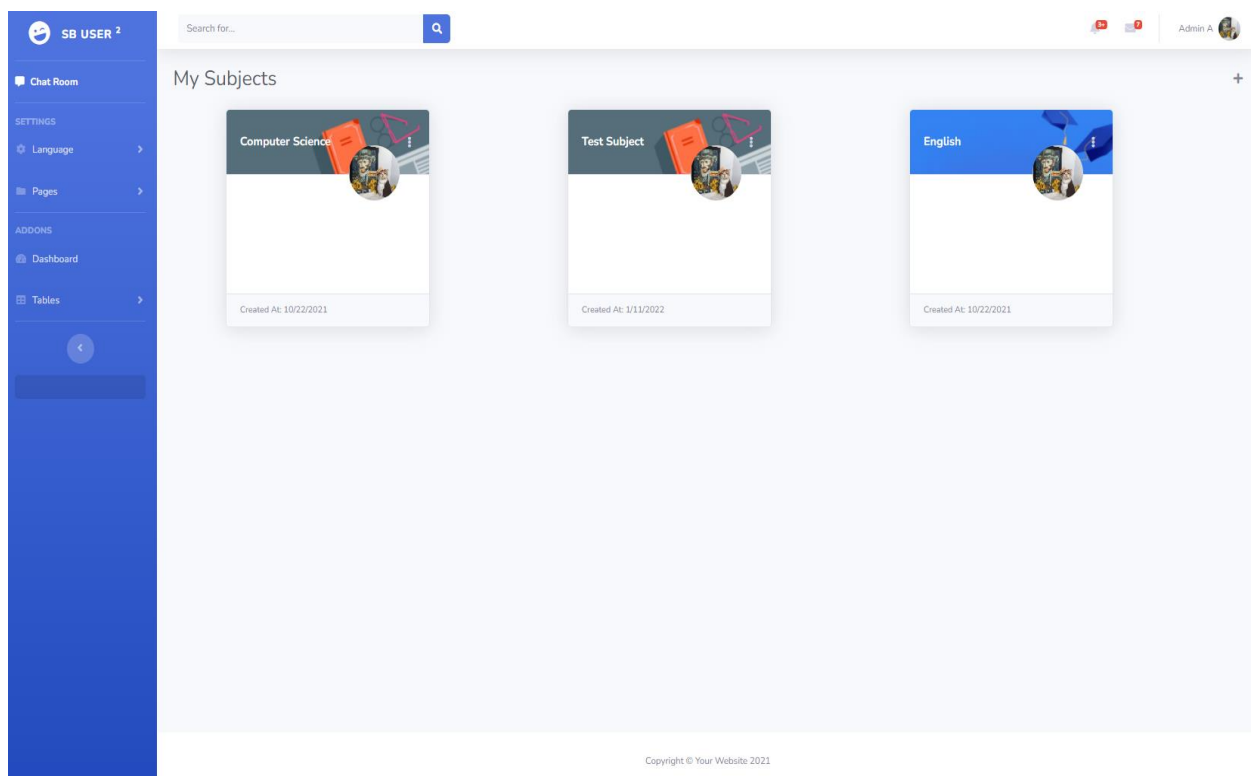


Figure 40 - Main page

Subject's Detail Page (All Projects):

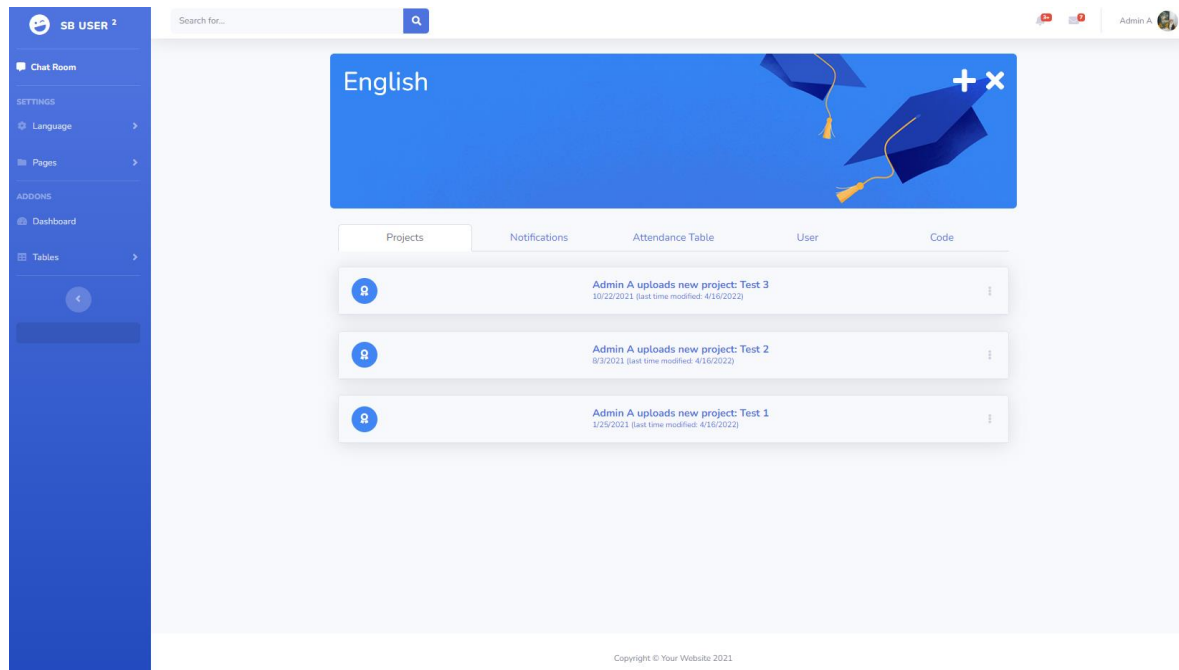


Figure 41 - Subject's detail page (All Projects)

Subject's Detail Page (Notification Creation):

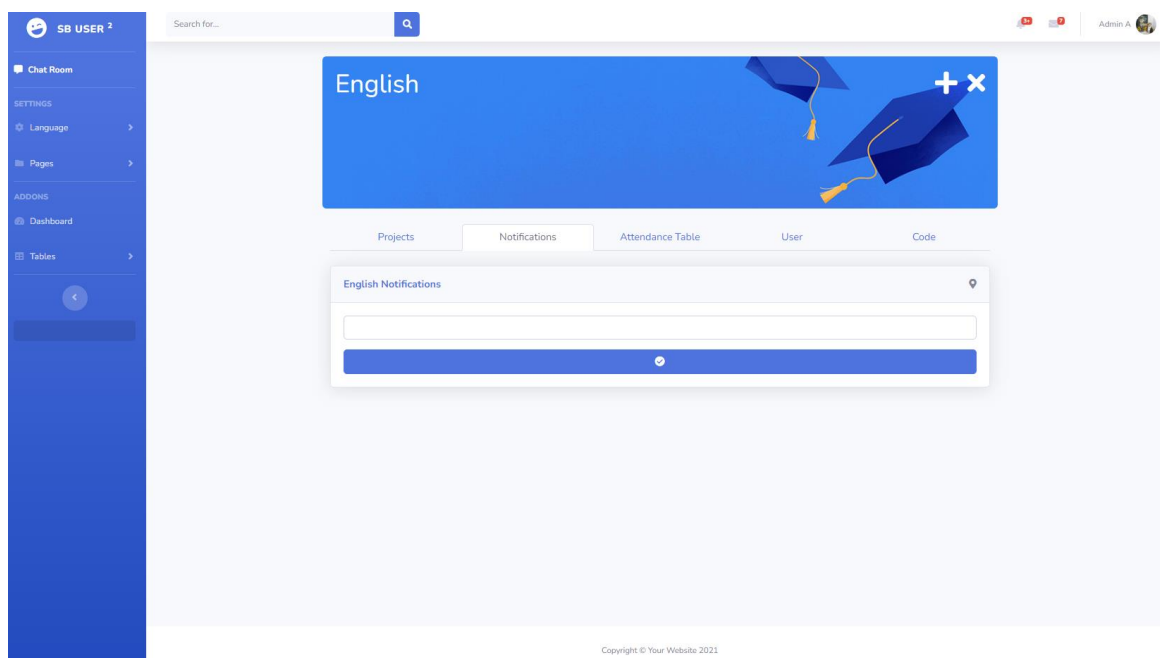


Figure 42 - Subject's detail page (Notification Creation)

Subject’s Detail Page (Attendance Table):

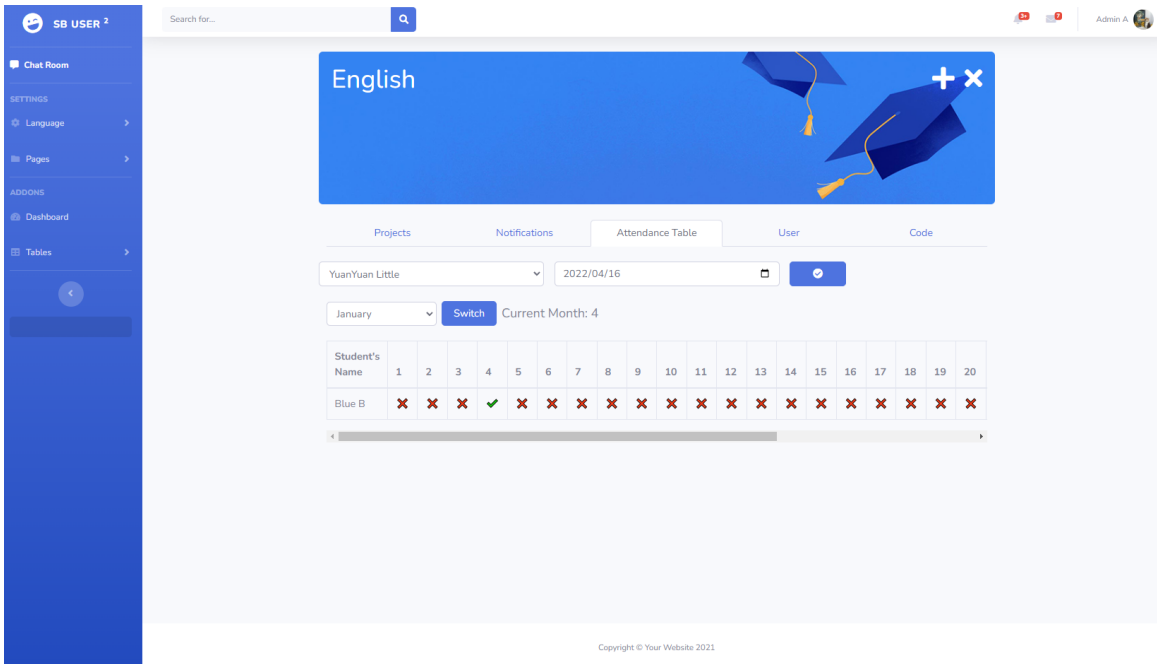


Figure 43 - Subject’s detail page (Attendance Table)

Subject’s Detail Page (Users):

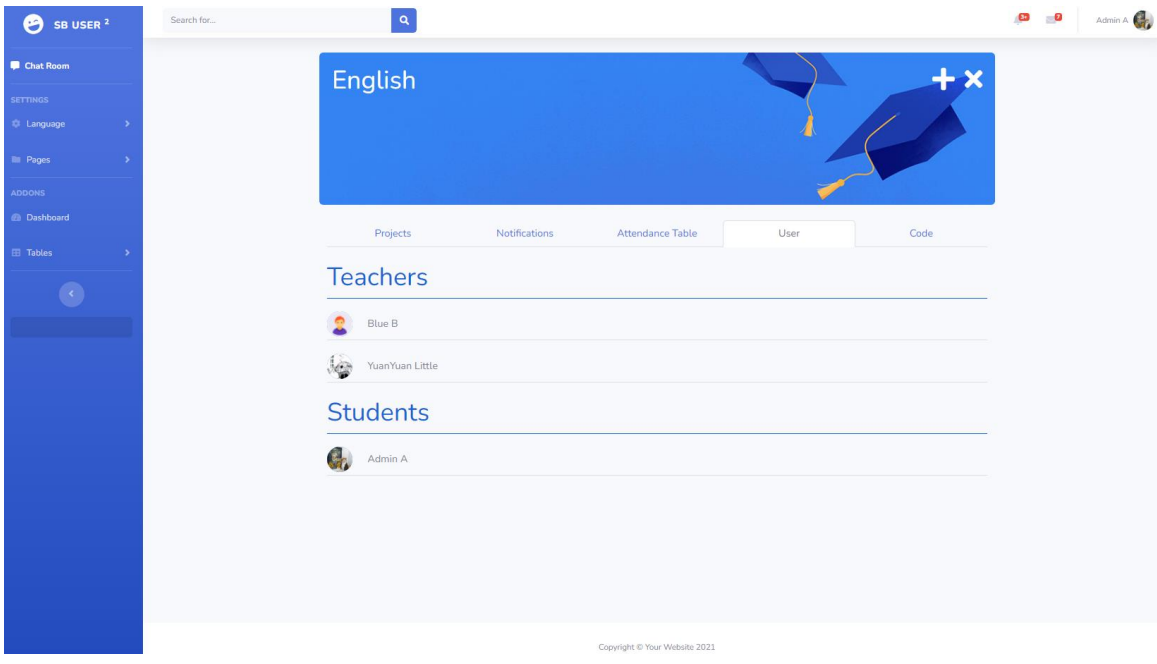


Figure 44 - Subject’s detail page (Users)

Subject's Detail Page (Subject's Code Settings):

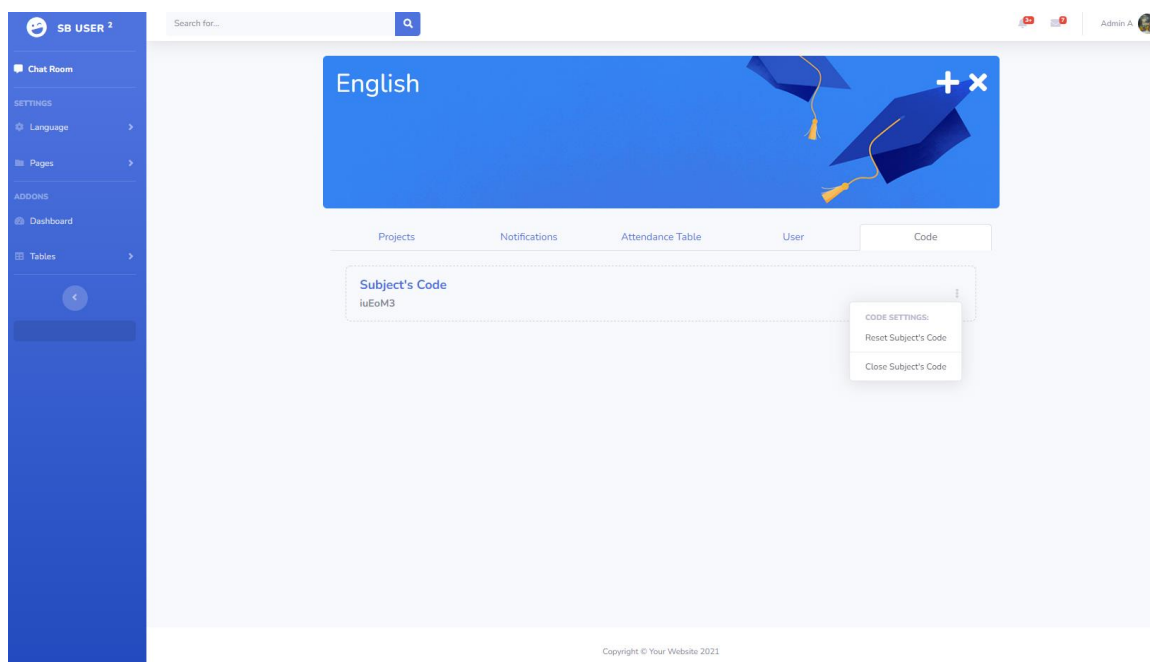


Figure 45 - Subject's detail page (Subject's Code Settings)

Subject Creation Page:

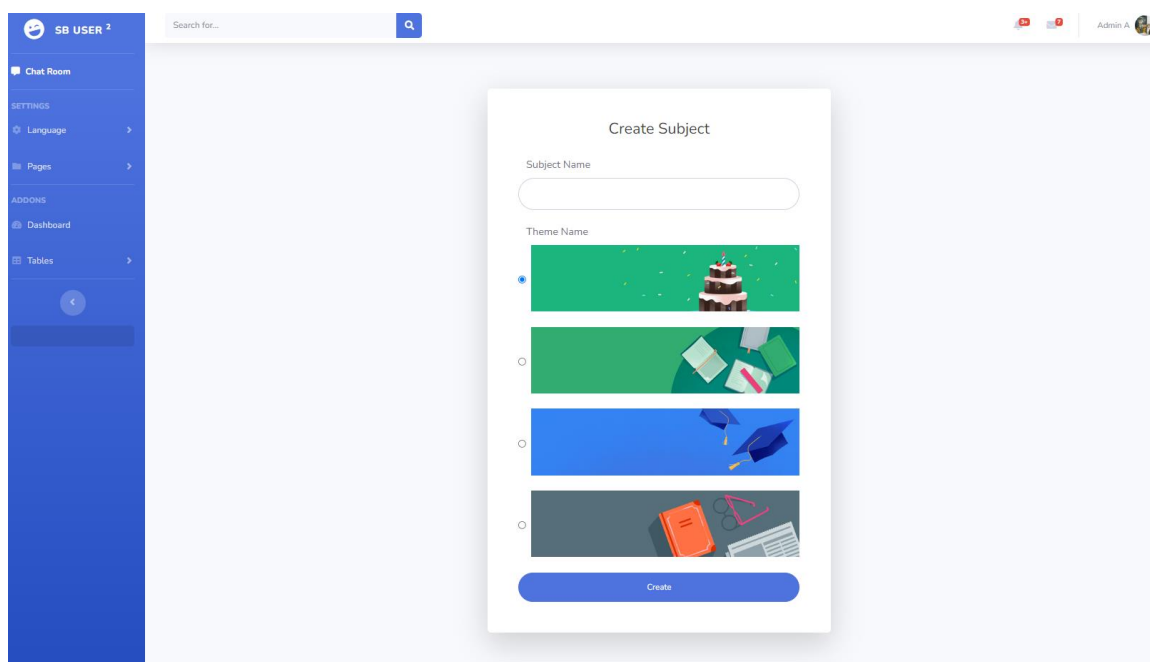
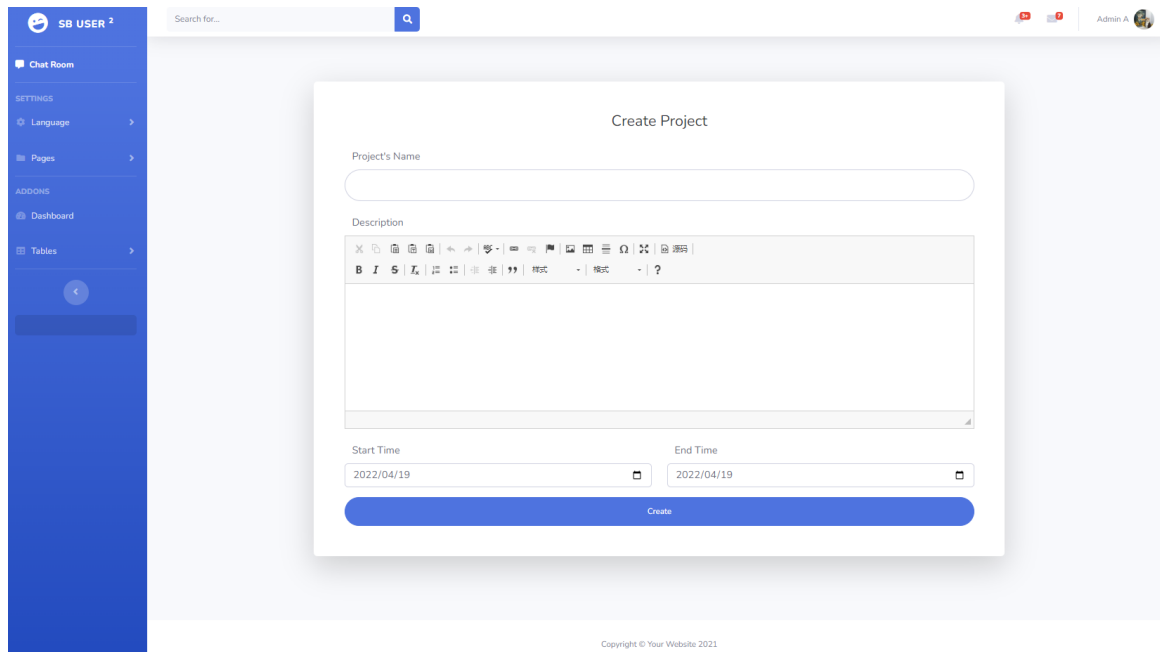


Figure 46 - Subject's creation page

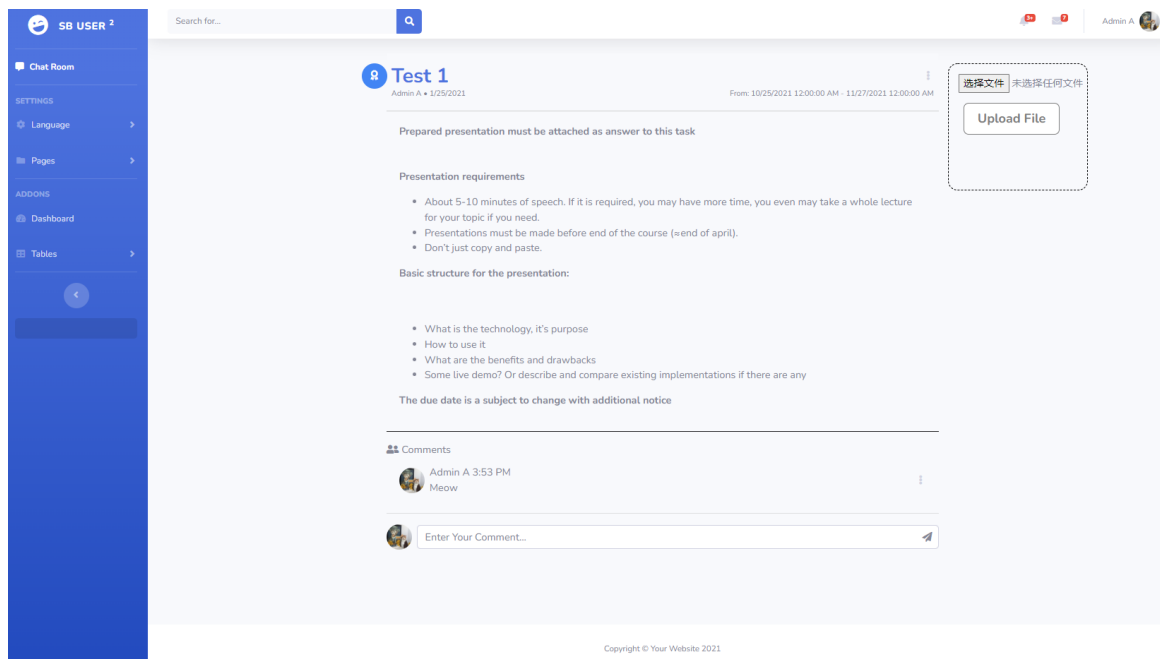
Project Creation Page:



The screenshot shows a web application interface for creating a project. On the left is a blue sidebar with a user profile 'SB USER 2' and navigation links for 'Chat Room', 'SETTINGS' (Language, Pages), and 'ADDONS' (Dashboard, Tables). The main content area has a search bar and a user profile 'Admin A'. A modal titled 'Create Project' is centered, containing a 'Project's Name' text input, a 'Description' text area with a rich text editor toolbar, and 'Start Time' and 'End Time' date pickers both set to '2022/04/19'. A blue 'Create' button is at the bottom of the modal. The footer contains the text 'Copyright © Your Website 2021'.

Figure 47 - Project creation page

Project's Detail Page:



The screenshot shows the detail page for a project titled 'Test 1' by 'Admin A' on '1/25/2021'. The page includes a sidebar and a top navigation bar. The main content area displays the project title, a date range 'From: 10/25/2021 12:00:00 AM - 11/27/2021 12:00:00 AM', and a file upload section with a button '选择文件' and a text '未选择任何文件'. Below this is an 'Upload File' button. The page contains two sections of text: 'Prepared presentation must be attached as answer to this task' and 'Presentation requirements' with a bulleted list. A 'Basic structure for the presentation:' section follows with another bulleted list. A note states 'The due date is a subject to change with additional notice'. At the bottom, there is a 'Comments' section with a comment from 'Admin A' at '3:53 PM' saying 'Meow', and a text input field for 'Enter Your Comment...'. The footer contains the text 'Copyright © Your Website 2021'.

Figure 48 - Project detail page

Record Creation Page:







The screenshot shows a web application interface with a blue sidebar on the left containing navigation links: Chat Room, SETTINGS (Language, Pages), and ADDONS (Dashboard, Tables). The main content area is light gray and features a 'Create Record' modal form. The form includes a 'Student Name' dropdown menu with 'YuanYuan Little' selected, a 'Spent Time (Hours)' input field with '0', a 'Week' dropdown menu with '10/25/2021 - 10/31/2021', a 'Progress(%)' slider set to 22, and a 'Note' text area with the placeholder 'input your note'. A blue 'Create' button is at the bottom of the form. The top of the page has a search bar, a user profile 'SB USER 2', and a user name 'Admin A'.

Copyright © Your Website 2021

Figure 49 - Record creation page

Project's Records Page:

The screenshot shows a web application interface with a blue sidebar on the left containing navigation links: Chat Room, SETTINGS (Language, Pages), and ADDONS (Dashboard, Tables). The main content area is light gray and features a 'Test 1's Records' table. The table is divided into two sections: '10/25/2021 - 10/31/2021' and '11/22/2021 - 11/28/2021'. Each section contains a table with columns for 'Student Name', 'Spent Time (Hours)', 'Progress (%)', 'Note', and 'Action'. The 'Action' column contains a green checkmark icon and a red trash can icon.

Test 1's Records				
10/25/2021 - 10/31/2021				
YuanYuan Little	1 hour(s)	10%	Work Hard	 
YuanYuan Little	2 hour(s)	45%	Checked	 
11/22/2021 - 11/28/2021				
YuanYuan Little	12 hour(s)	88%	Well Done	 

Copyright © Your Website 2021

Figure 50 - Project's records page

Project's Attachments Page:

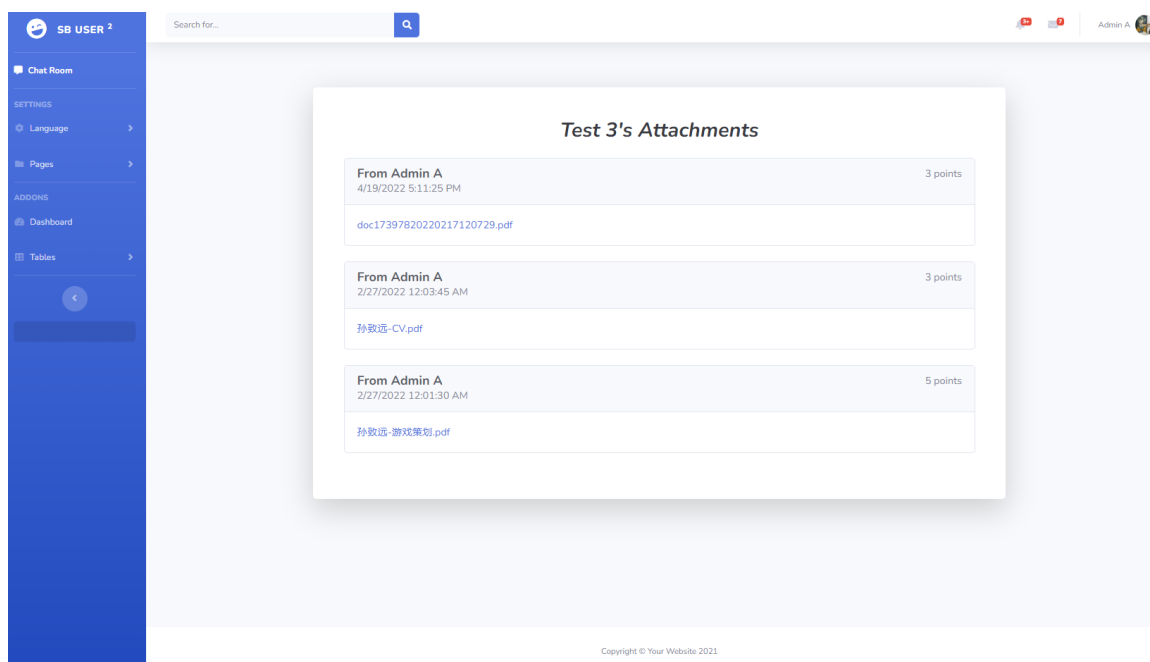


Figure 51 - Project's Attachments page

Chat Room:

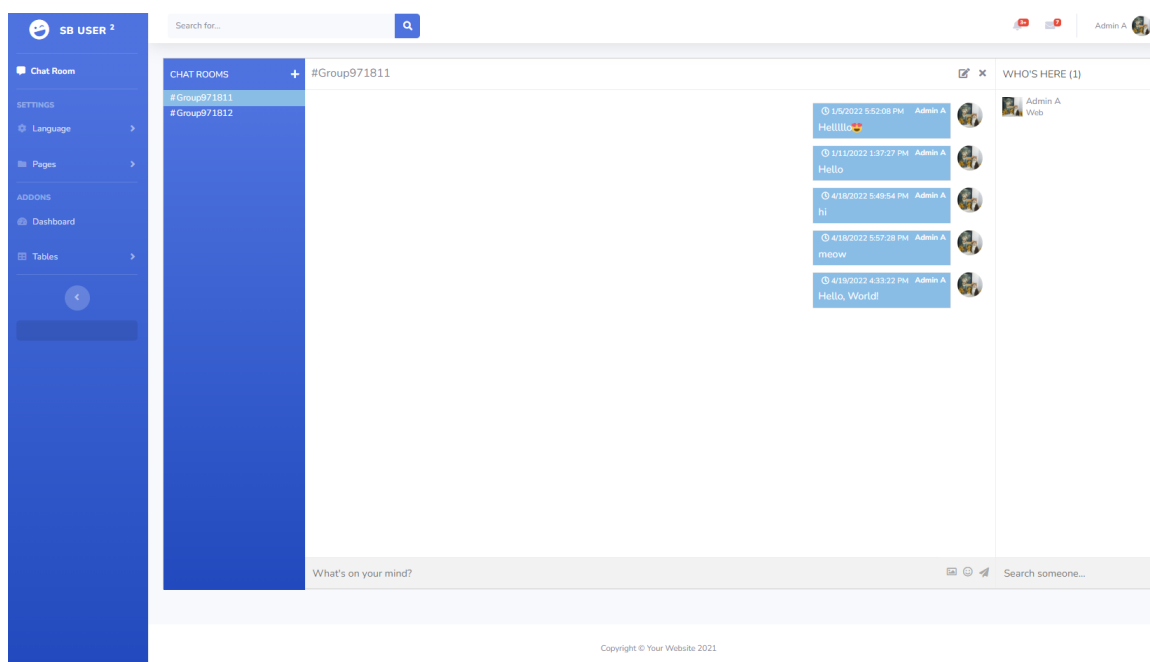


Figure 52 - Chat room

User's Events (To do list):

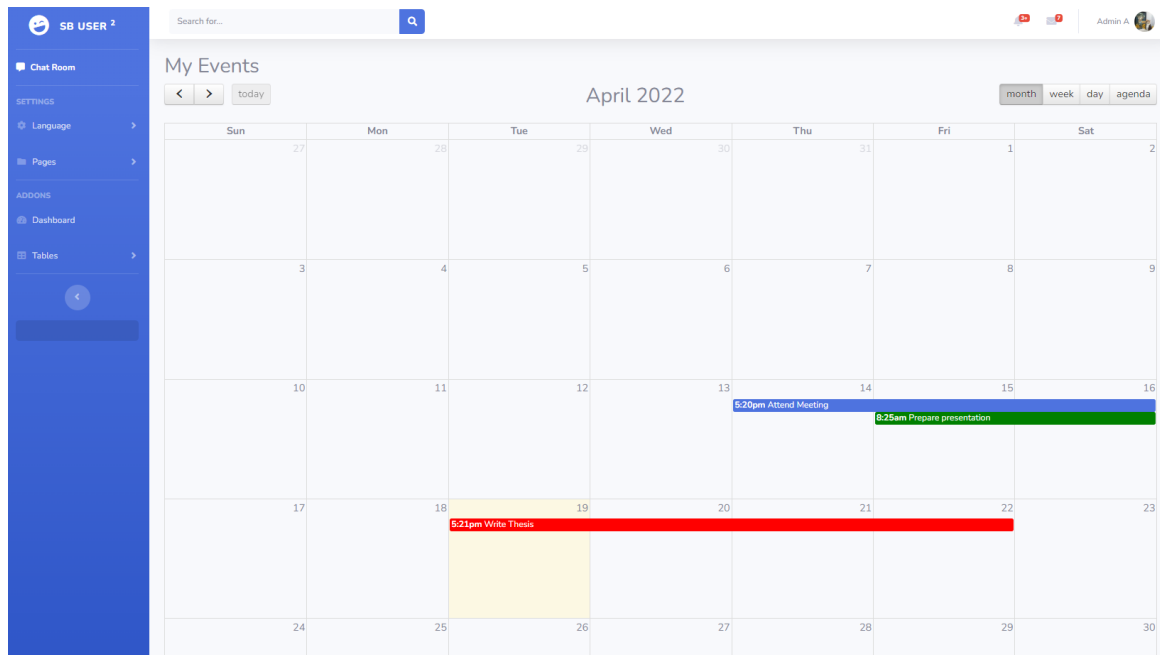


Figure 53 - User's events

User's Profile Page:

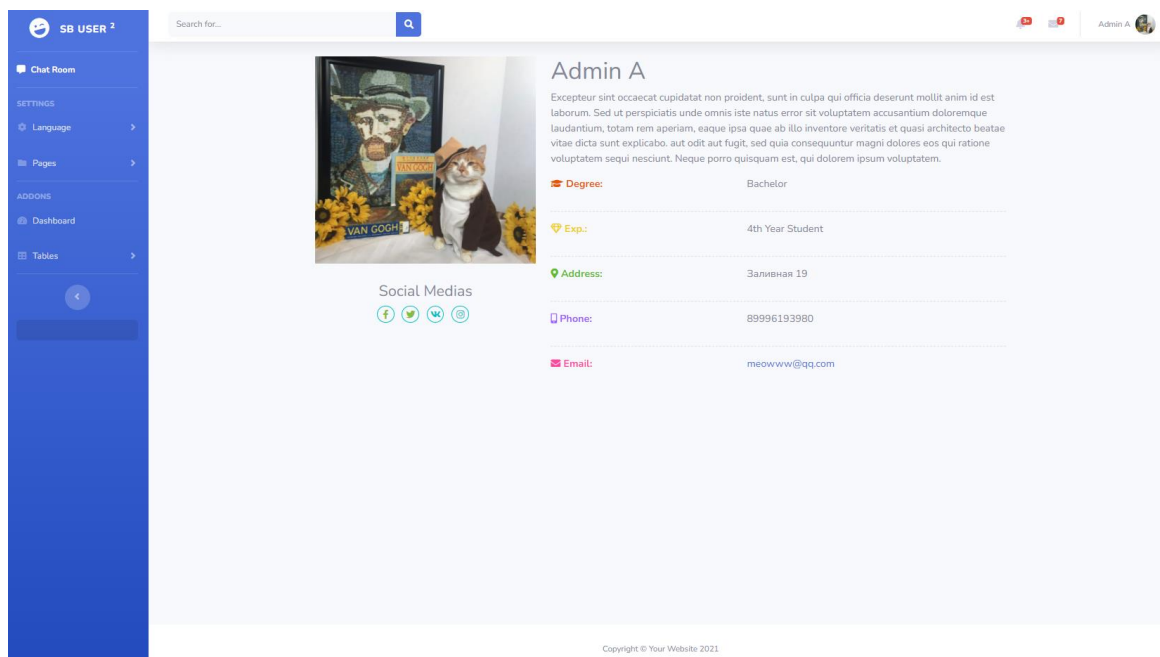


Figure 54 - User's profile page

Admin Dashboard:

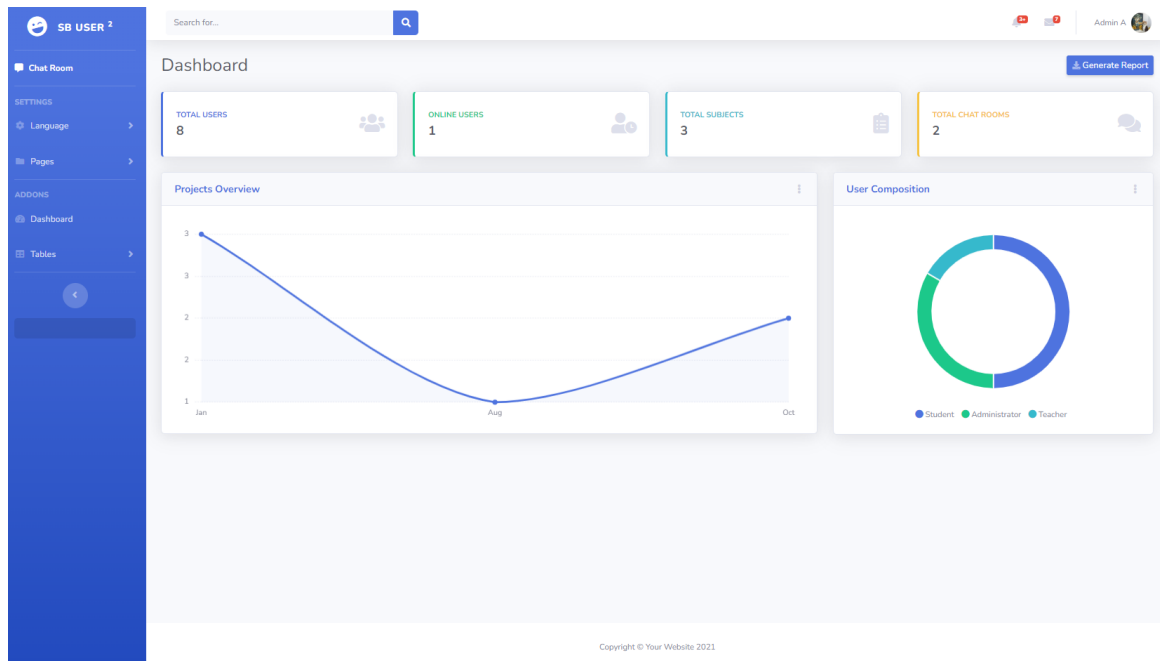


Figure 55 - Admin dashboard

Email Template:

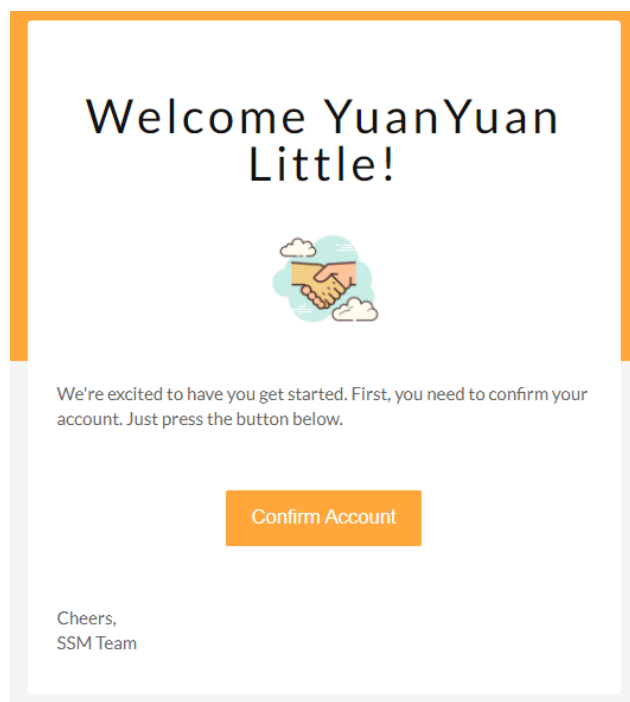


Figure 56 - Email template

Roles Table:

Search for...

SB USER 2

Chat Room

SETTINGS

Language

PAGES

ADDONS

Dashboard

Tables

All Roles

Add Role

Role's DataTable

rolename	Actions
Student	
Administrator	
Teacher	

Copyright © Your Website 2021

Figure 57 - Roles table

Edit Role Page:

Search for...

SB USER 2

Chat Room

SETTINGS

Language

PAGES

ADDONS

Dashboard

Tables

Edit Role

Role Name

Teacher

Update Role

Users in this role

meowww@qq.com

Manage Role's Users

Copyright © Your Website 2021

Figure 58 - Edit role page

Manage Users' Roles Page:

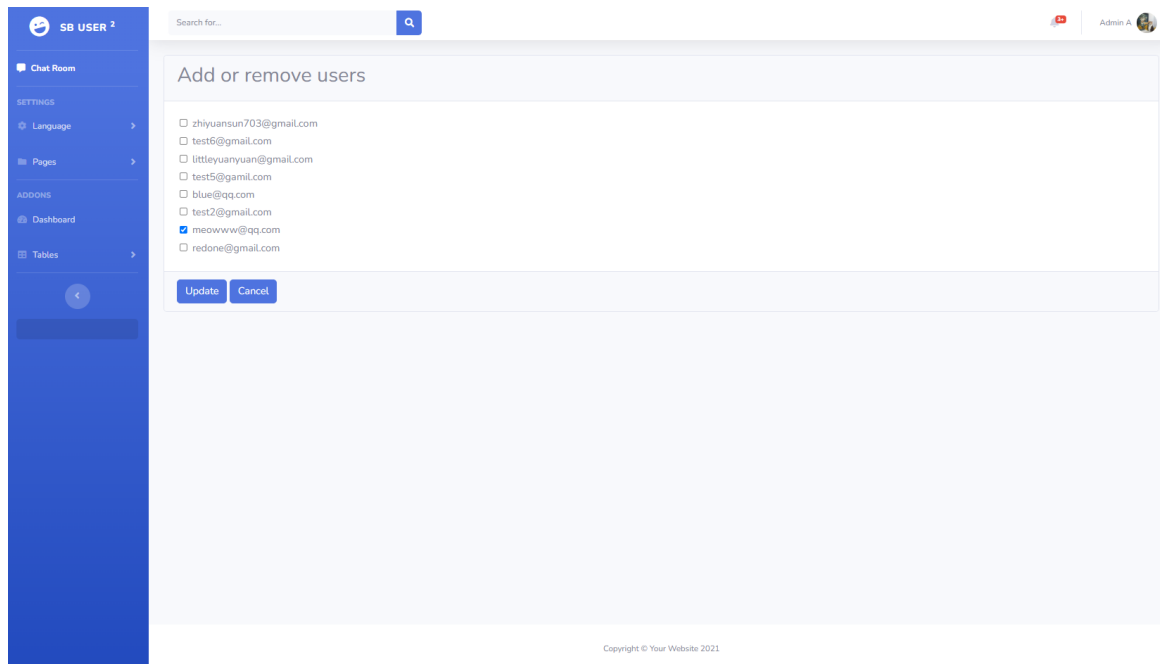


Figure 59 - Manage users' roles page

Отчет о проверке на заимствования №1



Автор: SUN ZHIYUAN

Проверяющий: SUN ZHIYUAN

Отчет предоставлен сервисом «Антиплагиат» - <http://users.antiplagiat.ru>

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 16
Начало загрузки: 30.05.2022 15:13:18
Длительность загрузки: 00:00:01
Имя исходного файла: Bachelor's thesis - ZhiYuanSun.pdf
Название документа: Bachelor's thesis - ZhiYuanSun
Размер текста: 46 кБ
Символов в тексте: 46943
Слов в тексте: 6655
Число предложений: 411

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 30.05.2022 15:13:20
Длительность проверки: 00:00:01
Комментарии: не указано
Модули поиска: Интернет Free

ЗАИМСТВОВАНИЯ
2,61%

САМОЦИТИРОВАНИЯ
0%

ЦИТИРОВАНИЯ
0%

ОРИГИНАЛЬНОСТЬ
97,39%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты, общеупотребительные выражения, фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Источник	Актуален на	Модуль поиска	Комментарии
[01]	0,98%	https://www.theseus.fi/bitstream/handle/10024/166420/Turun_Kimmo.pdf?isAllowed=y&sequence=2 https://theseus.fi/	31 Дек 2019	Интернет Free	
[02]	0,54%	TU Delft supervisor Dr. P.G. Kruit http://repository.tudelft.nl	03 Янв 2021	Интернет Free	
[03]	0,4%	Создание интернет магазина домашних кондитерских изделий с использованием современных инструментов веб-разработки http://library.eltech.ru	19 Сен 2019	Интернет Free	

Еще источников: 6

Еще заимствований: 0,67%

С отчетом ознакомлена
научный руководитель,
к.фр.-л.н., доцент

М.Н. Зенкова
02.06.2022г.