Higher Nationals – Assignment Front Sheet

Student Name/ID	Vo Dinh Nhat/ GBD210218		
Unit Title	Unit 30: Application Development		
Assignment Number	Assignment 2	**Assessor**	Pham Thanh Son
Submission Date		**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	

Grading grid

P4	P5	P6	M3	M4	M5	D2	D3
✗	✓	✓					

Assessor Feedback:

*Please note that constructive and useful feedback should allow students to understand:

a) Strengths of performance
b) Limitations of performance
c) Any improvements needed in future assessments

Feedback should be against the learning outcomes and assessment criteria to help students understand how these inform the process of judging the overall grade.

Feedback should give full guidance to the students on how they have met the learning outcomes and assessment criteria.

Grade:	**Assessor Signature:**	**Date:**

Resubmission Feedback:

*Please note resubmission feedback is focussed only on the resubmitted work

Grade:	**Assessor Signature:**	**Date:**

Internal Verifier's Comments:
Signature & Date:

* Please note that grade decisions are provisional. They are only confirmed once internal and external moderation has taken place and grades decisions have been agreed at the assessment.

Group 6:

Lê Nguyễn Thiên Ân

Võ Đình Nhật

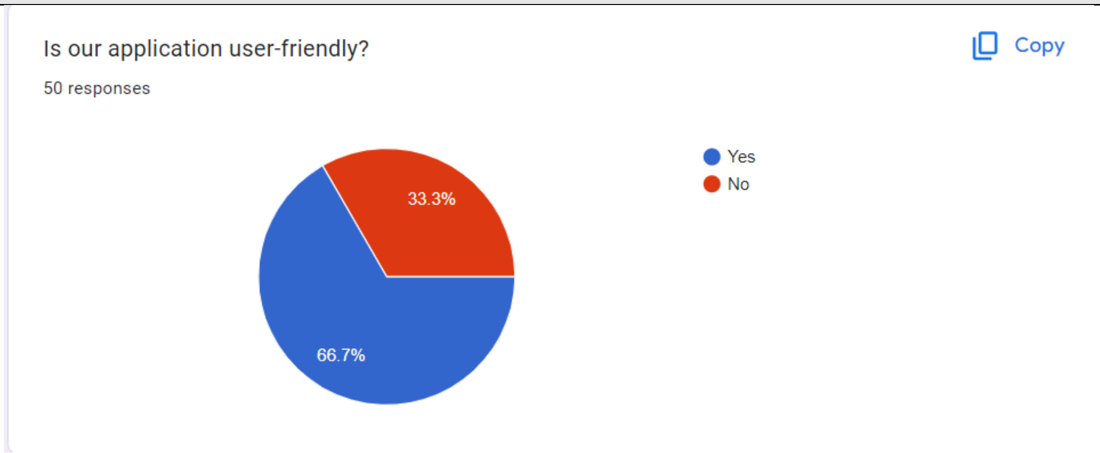
I. Introduction

My team used the specs as a reference to complete the system design for the previous report. Furthermore, from the beginning of the project, we recognized a number of risks and provided a solution for each. In this report, we will go over system development, code implementation, user evaluation after system experience, and possible future enhancements.

1. Peer Review and Feedback Analysis

TODO:

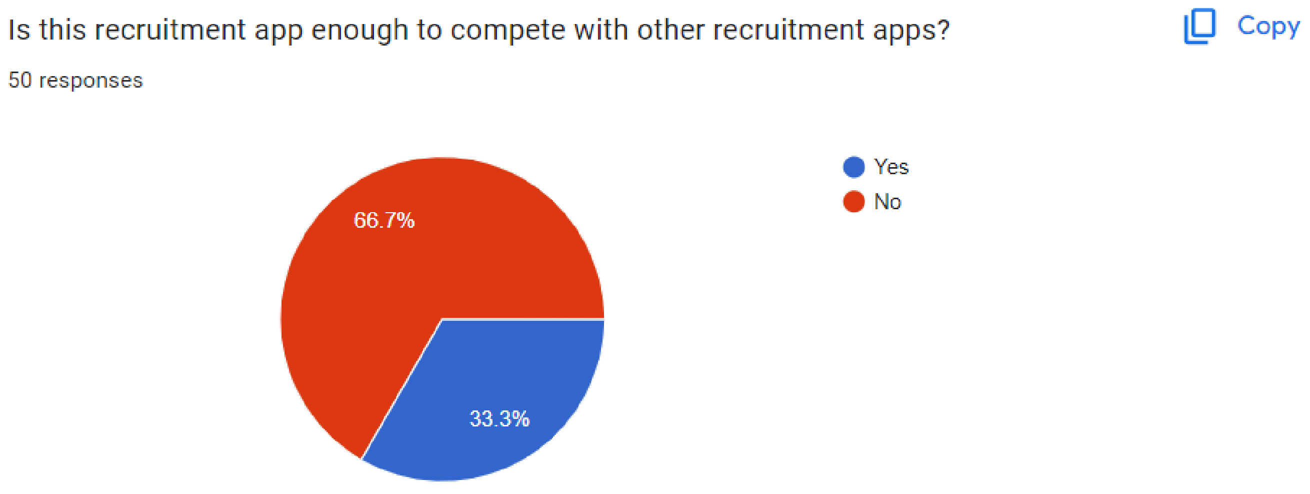
- *Formal questionnaire to reviews the business application, problem definition statement, proposed solution and development strategy*
- *Collect review feedbacks*
- *Interpret peer-review feedbacks*
- *Evaluate any new insights, ideas or potential improvements*



Is our application user-friendly?

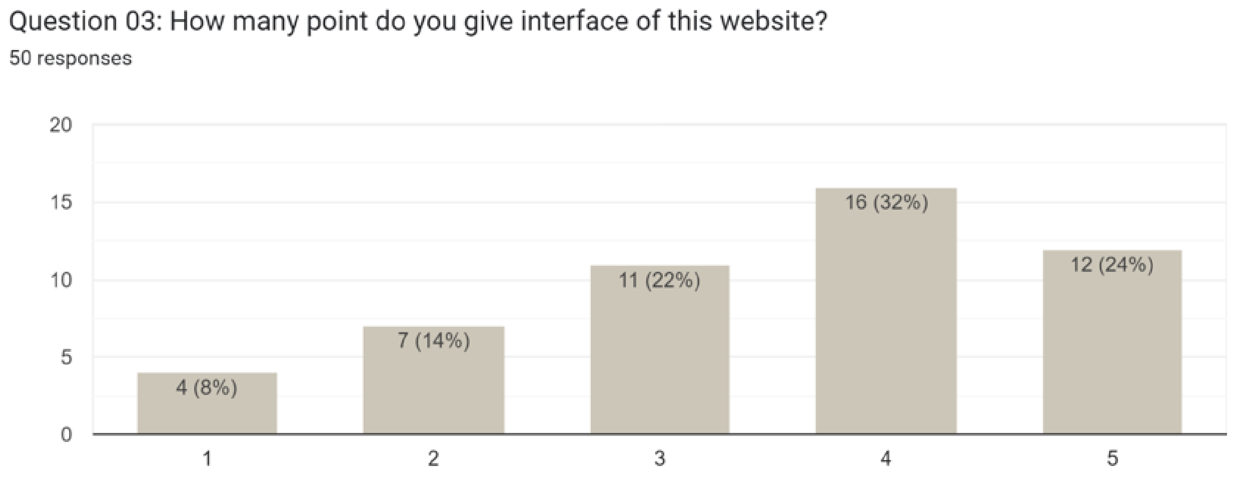
50 responses

Most people feel my app is user-friendly. accounting for 66.7%.


Is this recruitment app enough to compete with other recruitment apps?

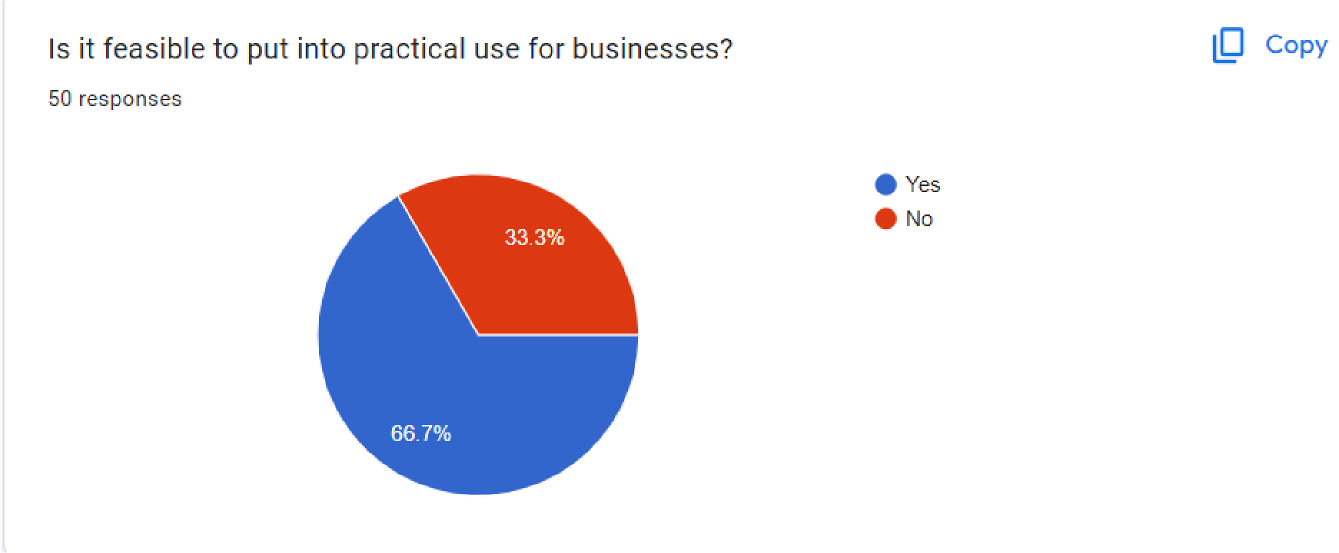
50 responses

It's sad that most people think that my application is not enough to compete with other applications. I think it's because my application still has many shortcomings.


Question 03: How many point do you give interface of this website?

50 responses

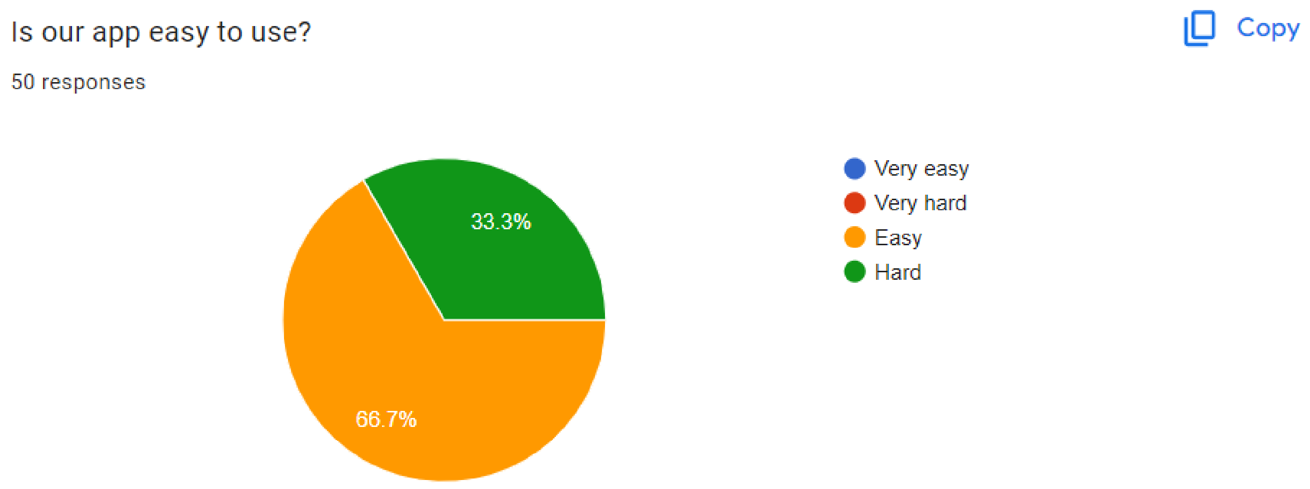
The website decoration is what we feel is the most incomplete of all the parts because we don't focus too much on decoration. That is shown through the data above, most people rate it at level 4 which is about 16% stable.



Is it feasible to put into practical use for businesses?

50 responses

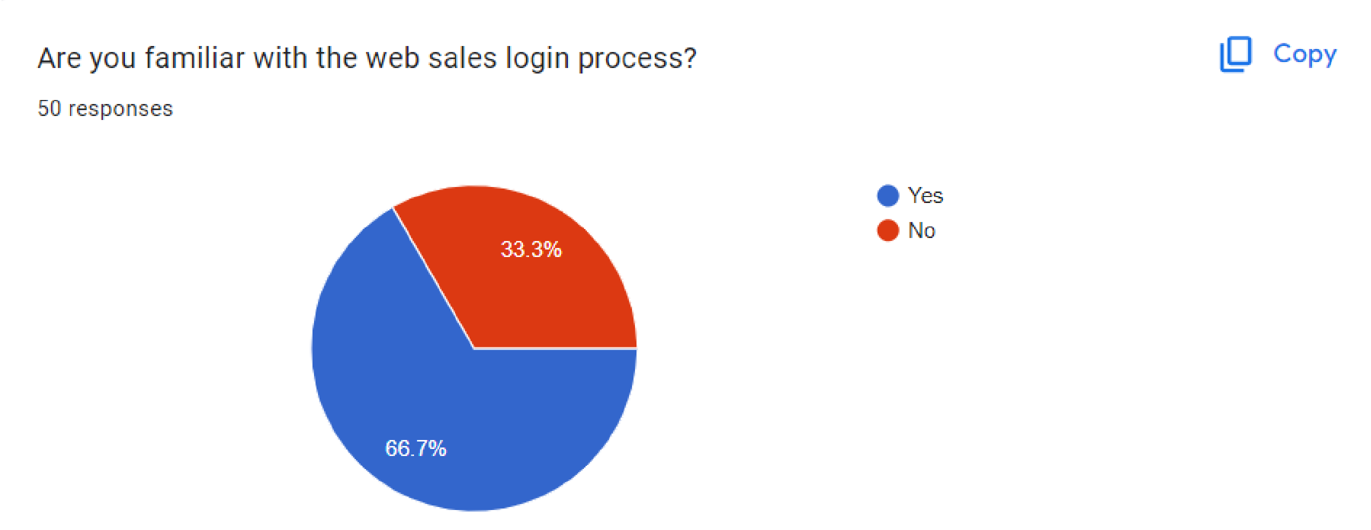
Most people think that my application is enough for practical applications for businesses. Accounting for 66.7%.



Is our app easy to use?

50 responses

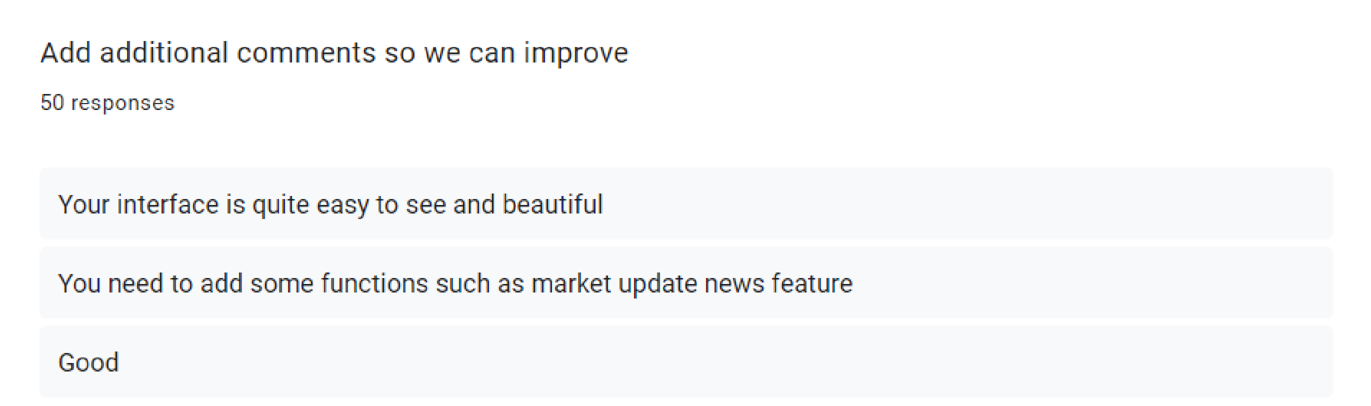
Most people find my app easy to use. The rest find it difficult to use, accounting for 33.3%.



Are you familiar with the web sales login process?

50 responses

The login process of our application is quite easy for everyone to use, the majority of people also think so, accounting for 66.7%.



Add additional comments so we can improve

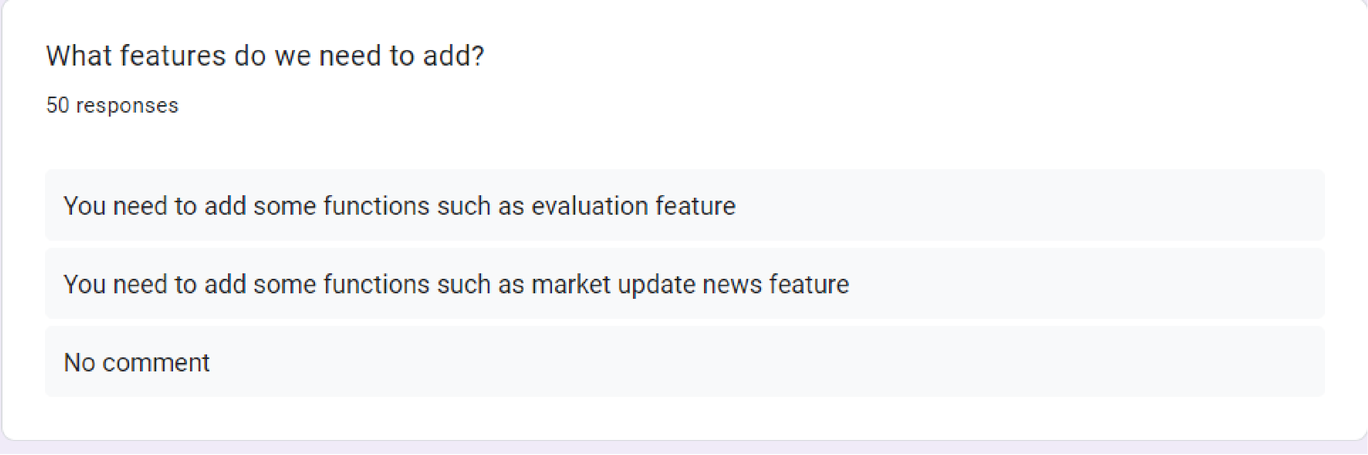
50 responses

Your interface is quite easy to see and beautiful

You need to add some functions such as market update news feature

Good

Some reviews include both positive and negative towards my website, which motivates us to improve the application in the future.



What features do we need to add?

50 responses

You need to add some functions such as evaluation feature

You need to add some functions such as market update news feature

No comment

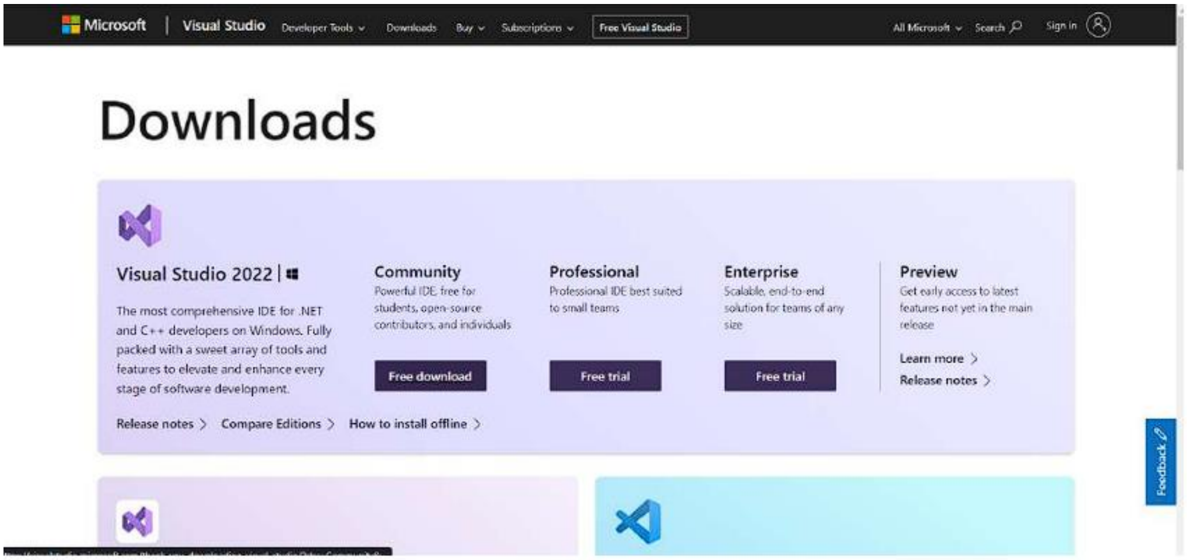
2. Develop the application

- Project FPTJobMatch will be develop with following phases:

- Phase 1: Local development 1.

Installing newest version of IDE (Visual Studio) and .NET

- Installing Visual Studio on Microsoft website. We will be using community version



Downloads

Visual Studio 2022

The most comprehensive IDE for .NET and C++ developers on Windows. Fully packed with a sweet array of tools and features to elevate and enhance every stage of software development.

Community
Powerful IDE, free for students, open-source contributors, and individuals
Free download

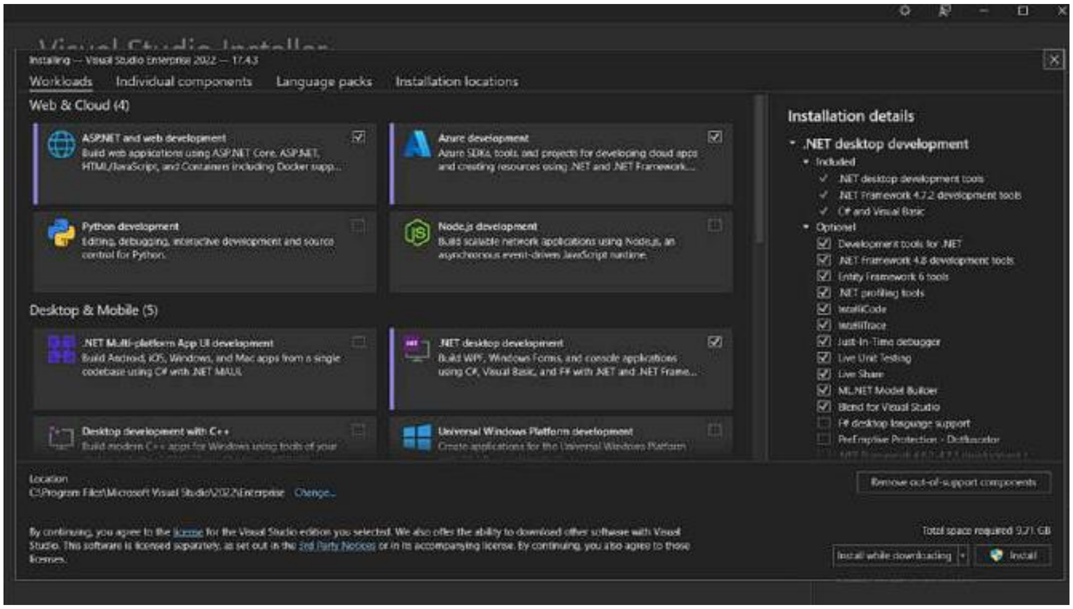
Professional
Professional IDE best suited to small teams
Free trial

Enterprise
Scalable, end-to-end solution for teams of any size
Free trial

Preview
Get early access to latest features not yet in the main release
Learn more >
Release notes >

Release notes > Compare Editions > How to install offline >

- The packages we require for the development process will be selected in the Visual Studio installation. We will only install packages that are compatible with the C# and.NET framework since this project is written in these languages.


Visual Studio Installer

Installing — Visual Studio Enterprise 2022 — 17.4.3

Workloads | Individual components | Language packs | Installation locations

Web & Cloud (4)

- ASP.NET and web development ☑
  Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...
- Azure development ☑
  Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework...
- Python development □
  Editing, debugging, interactive development and source control for Python.
- Node.js development □
  Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

- .NET Multi-platform App UI development □
  Build Android, iOS, Windows, and Mac apps from a single codebase using C# with .NET MAUI.
- .NET desktop development ☑
  Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
- Desktop development with C++ □
  Build modern C++ apps for Windows using tools of your
- Universal Windows Platform development □
  Create applications for the Universal Windows Platform

Installation details

- .NET desktop development
  - Included
    - ✓ .NET desktop development tools
    - ✓ .NET Framework 4.7.2 development tools
    - ✓ C# and Visual Basic
  - Optional
    - ☑ Development tools for .NET
    - ☑ .NET Framework 4.8 development tools
    - ☑ Entity Framework 6 tools
    - ☑ .NET profiling tools
    - ☑ IntelliCode
    - ☑ IntelliTrace
    - ☑ Just-In-Time debugger
    - ☑ Live Unit Testing
    - ☑ Live Share
    - ☑ ML.NET Model Builder
    - ☑ Blend for Visual Studio
    - □ F# desktop language support
    - □ PreEmptive Protection - Dotfuscator

Location
C:\Program Files\Microsoft Visual Studio\2022\Enterprise Change...

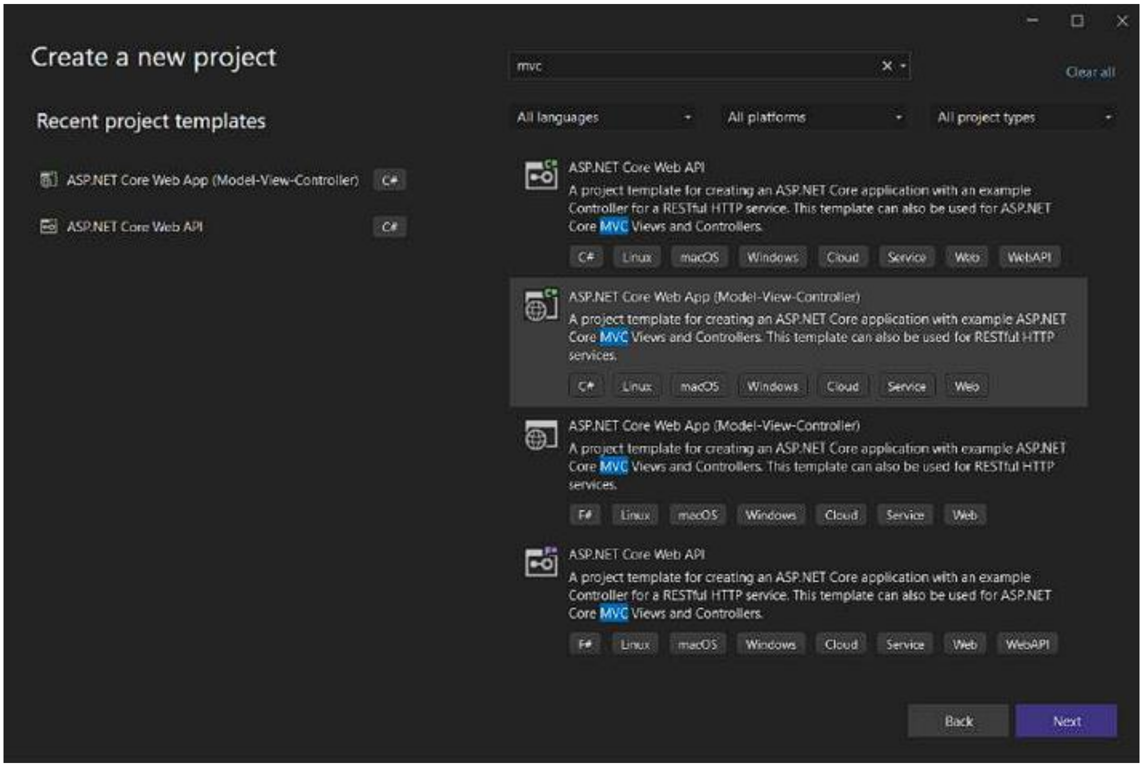
By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Remove out-of-support components

Total space required 9.71 GB

Install while downloading | Install

- Start a new MVC-based.NET project. Since the project will contain authorization and authentication features, I'll go with individual account authentication. This will register a new user and log them into the system using the built-in Identity of the.NET frameworks.



ASP.NET Core Web API

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

C# Linux macOS Windows Cloud Service Web WebAPI

ASP.NET Core Web App (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web App (Model-View-Controller)

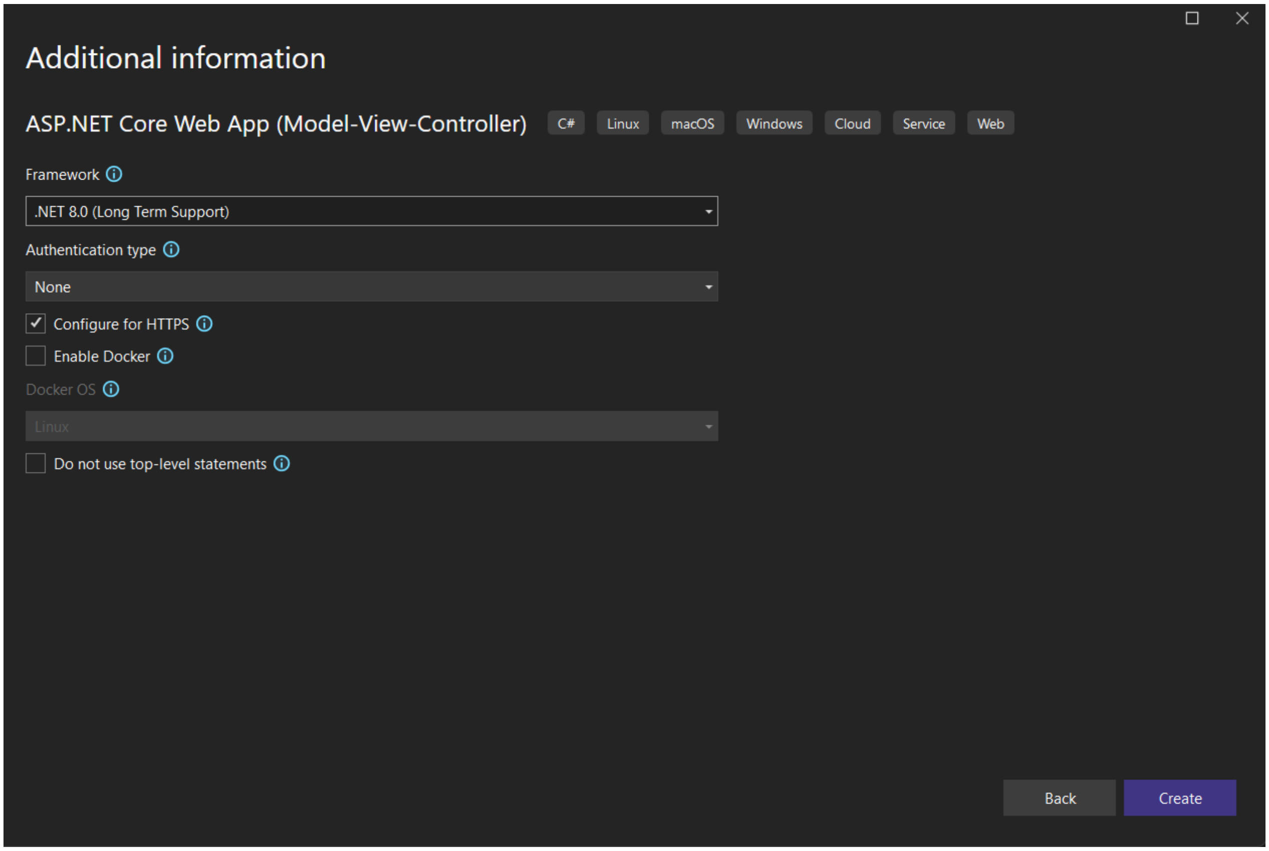
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

F# Linux macOS Windows Cloud Service Web

ASP.NET Core Web API

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

F# Linux macOS Windows Cloud Service Web WebAPI



Additional information

ASP.NET Core Web App (Model-View-Controller)

Framework: .NET 8.0 (Long Term Support)

Authentication type: None

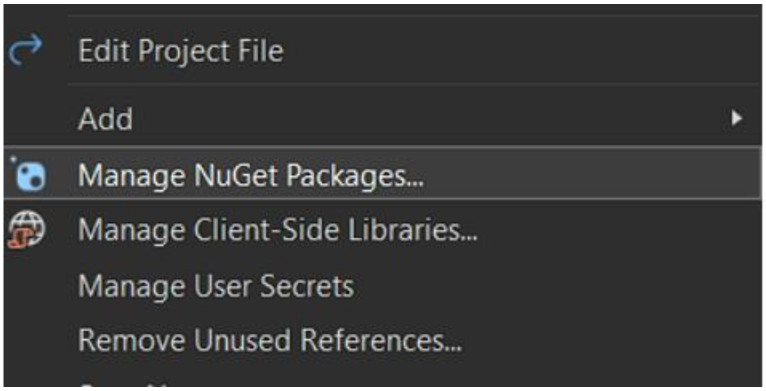
☑ Configure for HTTPS

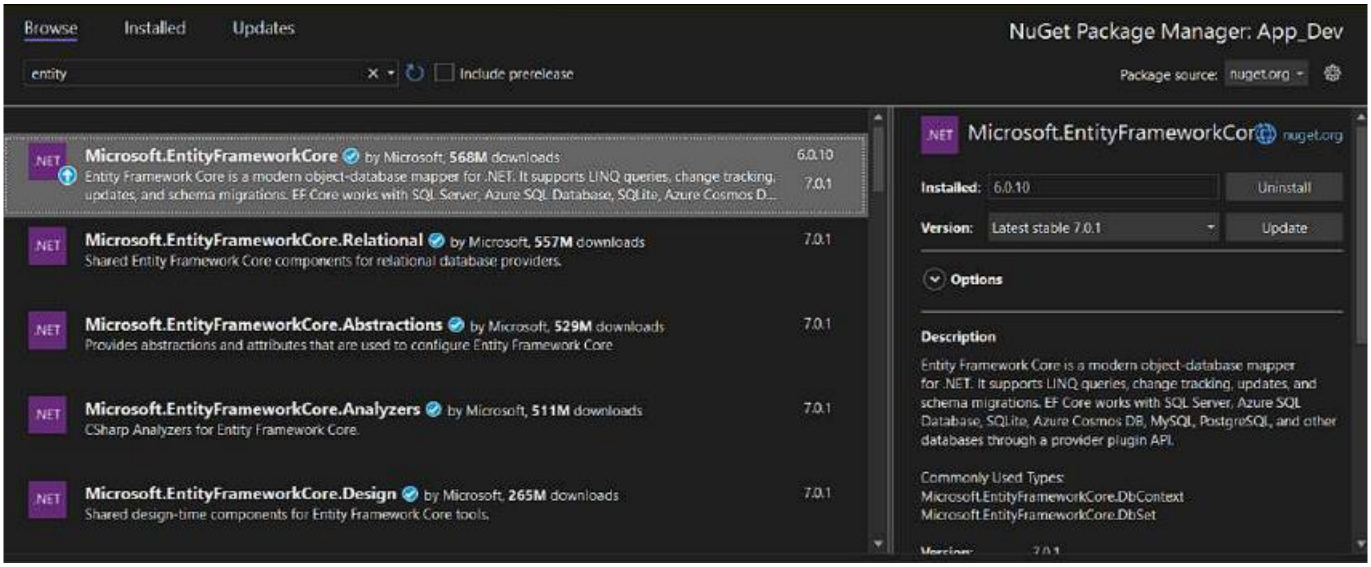
□ Enable Docker

Docker OS: Linux

□ Do not use top-level statements

- We will utilize Entity Framework to develop Object Relational Mapping (ORM) to database for.NET applications. For.NET applications, Microsoft supports the Entity Framework ORM framework, which is available as open source. Developers can work with data by using objects of domain-specific classes rather than concentrating on the underlying database tables and columns where this data is stored. By working with data at a higher degree of abstraction than they would with traditional programs, developers can use the Entity Framework to create and administer data-oriented applications.
- We must use NuGet Packages to install the Entity Framework package for the project after it has been created. I'll search for and install Entity Framework after opening the NuGet Packages.
- The NuGet package offers excellent assistance with installing, managing, and controlling packages within your project.





entity ✕ □ Include prerelease

Package source: nuget.org

Microsoft.EntityFrameworkCore by Microsoft, **568M** downloads — 6.0.10 / 7.0.1
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos D...

Microsoft.EntityFrameworkCore.Relational by Microsoft, **557M** downloads — 7.0.1
Shared Entity Framework Core components for relational database providers.

Microsoft.EntityFrameworkCore.Abstractions by Microsoft, **529M** downloads — 7.0.1
Provides abstractions and attributes that are used to configure Entity Framework Core

Microsoft.EntityFrameworkCore.Analyzers by Microsoft, **511M** downloads — 7.0.1
CSharp Analyzers for Entity Framework Core.

Microsoft.EntityFrameworkCore.Design by Microsoft, **265M** downloads — 7.0.1
Shared design-time components for Entity Framework Core tools.

Microsoft.EntityFrameworkCor nuget.org

Installed: 6.0.10 Uninstall

Version: Latest stable 7.0.1 Update

Options

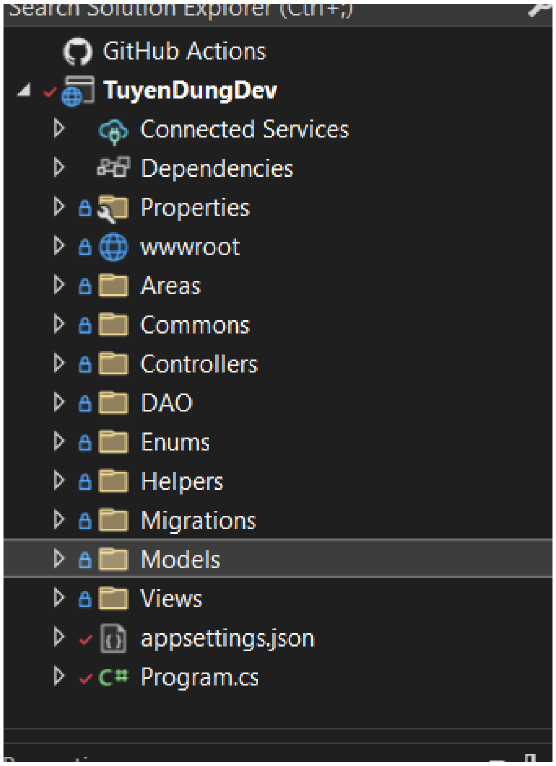
Description

Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.

Commonly Used Types:
Microsoft.EntityFrameworkCore.DbContext
Microsoft.EntityFrameworkCore.DbSet

Version: 7.0.1

a. Folder structure of the application



Search Solution Explorer (Ctrl+;)

```
GitHub Actions
◢ TuyenDungDev
    ▷ Connected Services
    ▷ Dependencies
    ▷ Properties
    ▷ wwwroot
    ▷ Areas
    ▷ Commons
    ▷ Controllers
    ▷ DAO
    ▷ Enums
    ▷ Helpers
    ▷ Migrations
    ▷ Models
    ▷ Views
    ▷ appsettings.json
    ▷ C# Program.cs
```

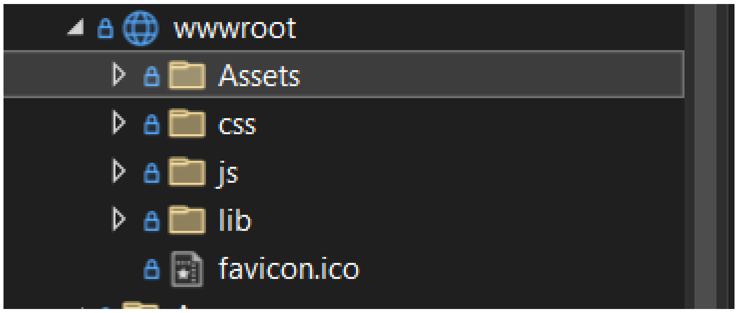
Figure 1:Folder structure of the application

A well-liked web development framework for creating scalable and reliable online applications is called ASP.NET Core. It adheres to the Model-View-Controller (MVC) design, which offers a distinct division of responsibilities amongst the many application components. We have implemented the platform using ASP.NET Core in our online book sales e-commerce application. Because of the MVC approach, the application's layout structure is quite clear. Areas, Data Access Object, Models, Controllers, and Views are some of the important folders that make up the program.

The Area folder contains two roles Administrator and "NhaTuyenDung" which are responsible for handling different roles in the platform. The Data Access Objects (DAO) directory is where the database is connected and where all data access logic is stored. The Models folder contains all objects stored in the database, including Candidate, Employer, TinTuyenDung, UngTuyen. The Controllers folder is where all of the user's controllers are located. These controllers are responsible for handling incoming requests from users and returning appropriate responses. Finally, the Views folder contains the user interface, which is responsible for rendering the HTML visible to the user. Overall, using ASP.NET Core in our online recruiting e-commerce application provides several key benefits, including scalability, robustness, and clear separation of functions.

ingredient. By following the MVC pattern and organizing the application into different folders and roles, we have created a platform that is easy to understand, maintain, and scale as needed.

wwwroot



The static file resources for your web application are kept in the "wwwroot" directory within an ASP.NET Core project. It specifically includes files that the user's browser will download when they visit my website, including CSS, JavaScript, pictures, fonts, and other files.

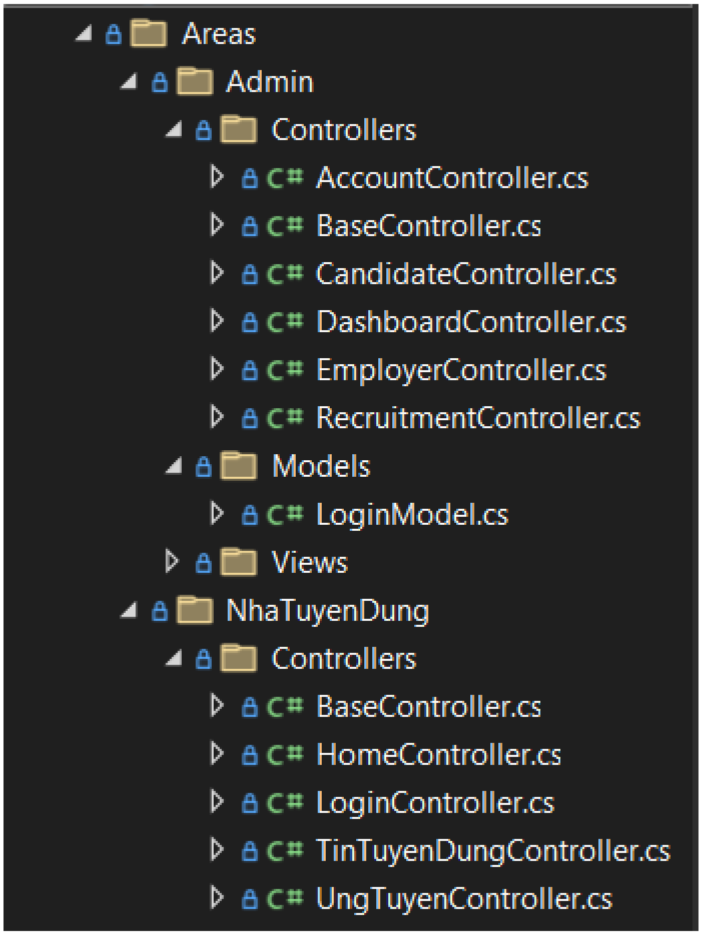
Here are some key points of the "wwwroot" directory:

-Configure root folder: In the "wwwroot" folder, you will usually find folders like "css", "js", "images" and "lib". This is where you store the respective files of each resource type.

Distinguish between dynamic and static resources: Files in the "wwwroot" directory are considered static resources, meaning they are downloaded and used by the browser directly from the client. This distinguishes them from resources, such as actions (actions) in controllers, which are processed by the encoding server before being sent to the client.

-Flexibility and easy management: Placing static file resources in the "wwwroot" folder makes managing and organizing your data clear and easy. This makes your code more readable and maintainable.

Area

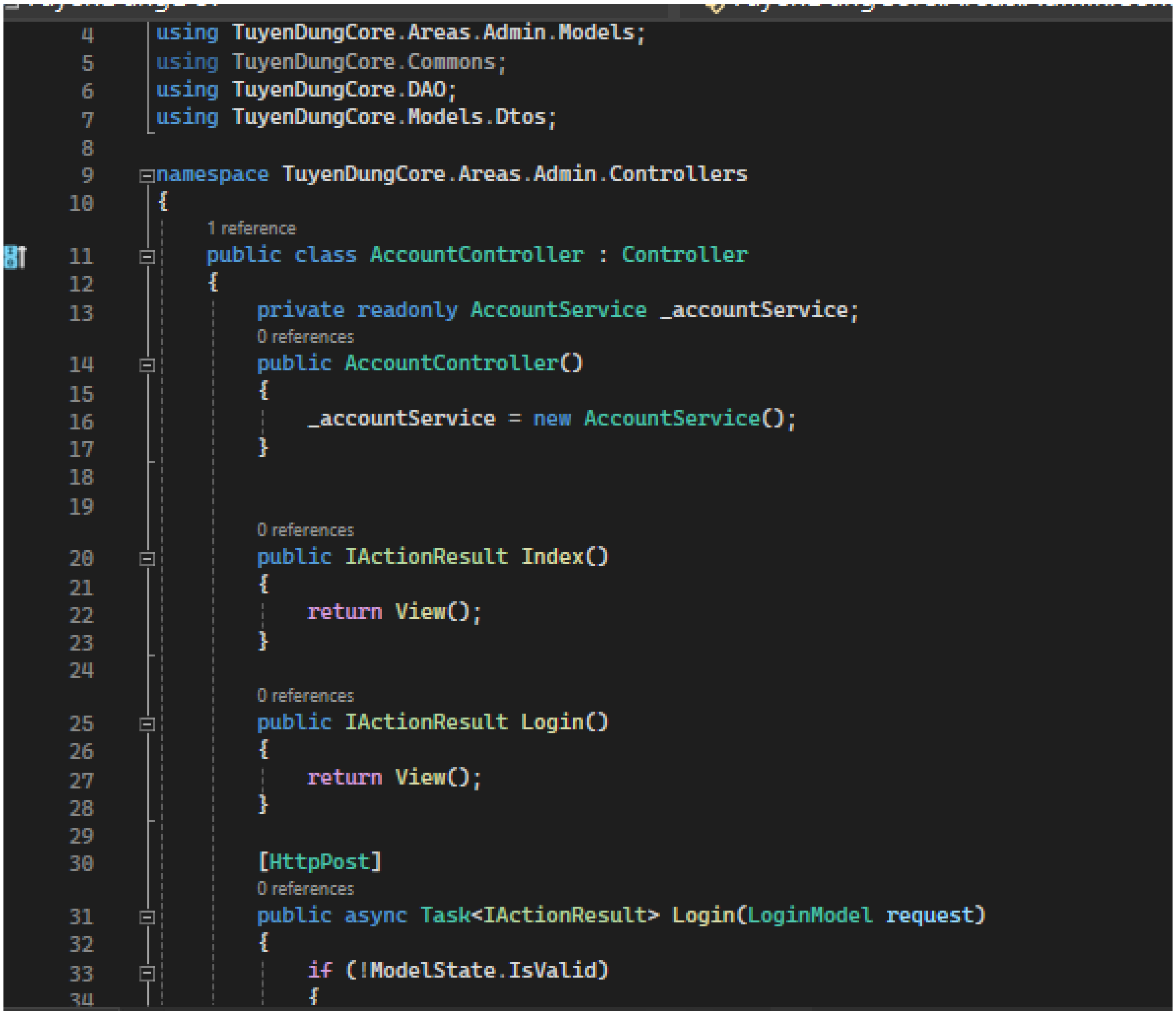


```
◢ Areas
  ◢ Admin
    ◢ Controllers
      ▷ C# AccountController.cs
      ▷ C# BaseController.cs
      ▷ C# CandidateController.cs
      ▷ C# DashboardController.cs
      ▷ C# EmployerController.cs
      ▷ C# RecruitmentController.cs
    ◢ Models
      ▷ C# LoginModel.cs
    ▷ Views
  ◢ NhaTuyenDung
    ◢ Controllers
      ▷ C# BaseController.cs
      ▷ C# HomeController.cs
      ▷ C# LoginController.cs
      ▷ C# TinTuyenDungController.cs
      ▷ C# UngTuyenController.cs
```

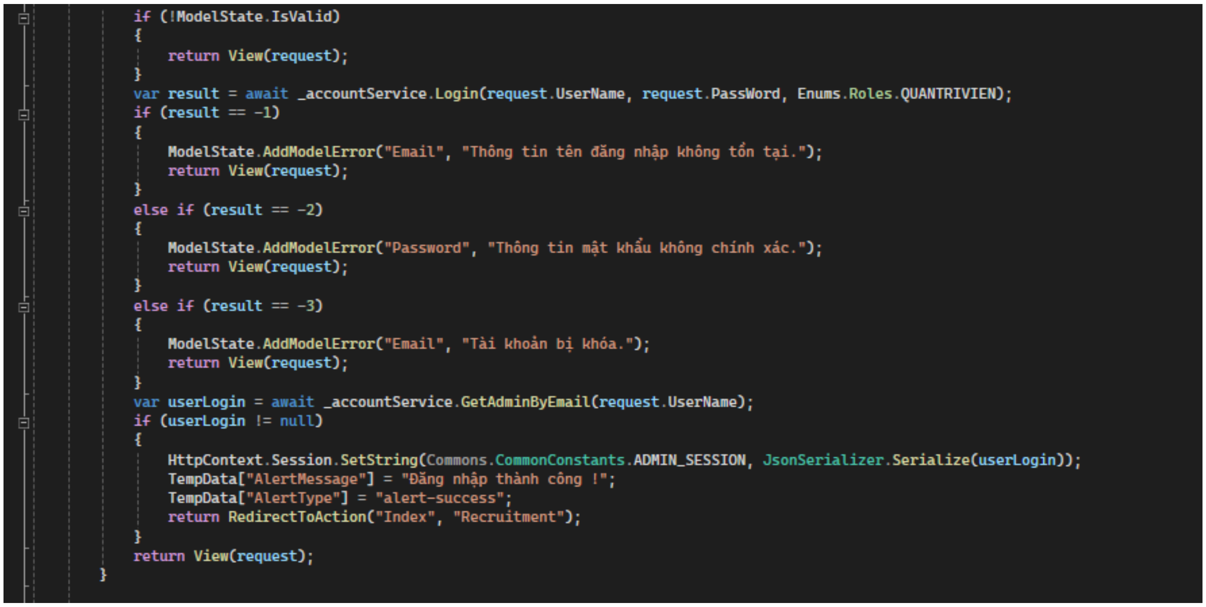
The "Area" section in ASP.NET Core is an application organization mechanism that divides my source code into smaller and organized parts, making it easier for you to manage your application as it gets larger. and more complicated.

Including Model-View-Controller of Admin and NhaTuyenDung

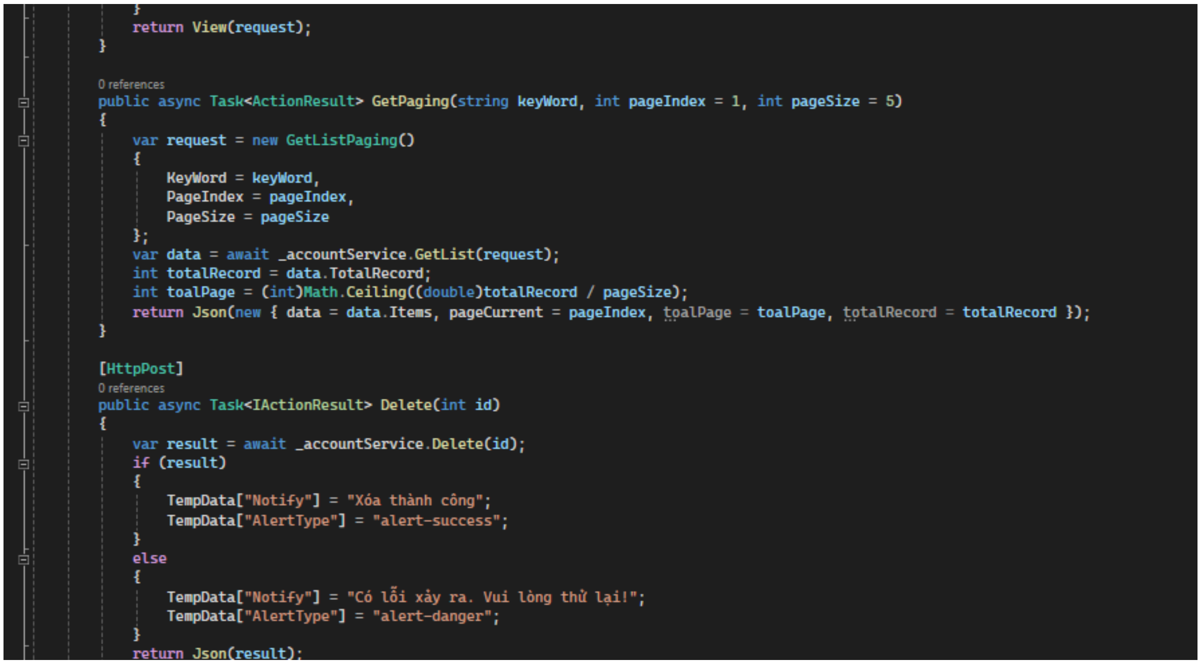
Account controller:



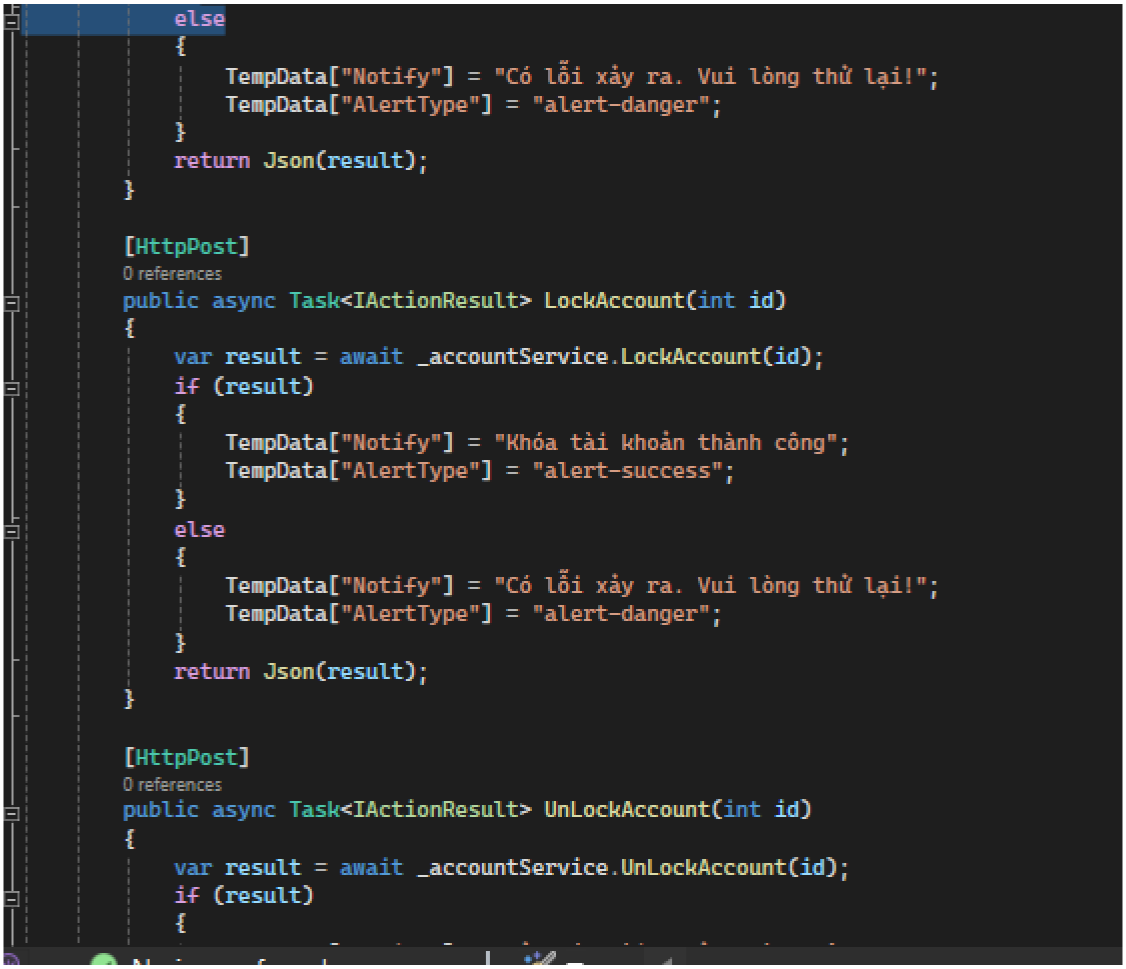
```
using TuyenDungCore.Areas.Admin.Models;
using TuyenDungCore.Commons;
using TuyenDungCore.DAO;
using TuyenDungCore.Models.Dtos;

namespace TuyenDungCore.Areas.Admin.Controllers
{
    1 reference
    public class AccountController : Controller
    {
        private readonly AccountService _accountService;
        0 references
        public AccountController()
        {
            _accountService = new AccountService();
        }


        0 references
        public IActionResult Index()
        {
            return View();
        }

        0 references
        public IActionResult Login()
        {
            return View();
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> Login(LoginModel request)
        {
            if (!ModelState.IsValid)
            {
```



```
        if (!ModelState.IsValid)
        {
            return View(request);
        }
        var result = await _accountService.Login(request.UserName, request.PassWord, Enums.Roles.QUANTRIVIEN);
        if (result == -1)
        {
            ModelState.AddModelError("Email", "Thông tin tên đăng nhập không tồn tại.");
            return View(request);
        }
        else if (result == -2)
        {
            ModelState.AddModelError("Password", "Thông tin mật khẩu không chính xác.");
            return View(request);
        }
        else if (result == -3)
        {
            ModelState.AddModelError("Email", "Tài khoản bị khóa.");
            return View(request);
        }
        var userLogin = await _accountService.GetAdminByEmail(request.UserName);
        if (userLogin != null)
        {
            HttpContext.Session.SetString(Commons.CommonConstants.ADMIN_SESSION, JsonSerializer.Serialize(userLogin));
            TempData["AlertMessage"] = "Đăng nhập thành công !";
            TempData["AlertType"] = "alert-success";
            return RedirectToAction("Index", "Recruitment");
        }
        return View(request);
    }
```



```
        }
        return View(request);
    }

    0 references
    public async Task<ActionResult> GetPaging(string keyWord, int pageIndex = 1, int pageSize = 5)
    {
        var request = new GetListPaging()
        {
            KeyWord = keyWord,
            PageIndex = pageIndex,
            PageSize = pageSize
        };
        var data = await _accountService.GetList(request);
        int totalRecord = data.TotalRecord;
        int toalPage = (int)Math.Ceiling((double)totalRecord / pageSize);
        return Json(new { data = data.Items, pageCurrent = pageIndex, toalPage = toalPage, totalRecord = totalRecord });
    }

    [HttpPost]
    0 references
    public async Task<IActionResult> Delete(int id)
    {
        var result = await _accountService.Delete(id);
        if (result)
        {
            TempData["Notify"] = "Xóa thành công";
            TempData["AlertType"] = "alert-success";
        }
        else
        {
            TempData["Notify"] = "Có lỗi xảy ra. Vui lòng thử lại!";
            TempData["AlertType"] = "alert-danger";
        }
        return Json(result);
```



```
            else
            {
                TempData["Notify"] = "Có lỗi xảy ra. Vui lòng thử lại!";
                TempData["AlertType"] = "alert-danger";
            }
            return Json(result);
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> LockAccount(int id)
        {
            var result = await _accountService.LockAccount(id);
            if (result)
            {
                TempData["Notify"] = "Khóa tài khoản thành công";
                TempData["AlertType"] = "alert-success";
            }
            else
            {
                TempData["Notify"] = "Có lỗi xảy ra. Vui lòng thử lại!";
                TempData["AlertType"] = "alert-danger";
            }
            return Json(result);
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> UnLockAccount(int id)
        {
            var result = await _accountService.UnLockAccount(id);
            if (result)
            {
```

This code is part of an ASP.NET Core web application in the administration area (Admin), which is responsible for handling requests related to user accounts in the system. Below is the function of each method in the AccountController controller:

-Index(): This method returns the main page of the admin area.

-Login(): This method returns the login page for the admin user.

-[HttpPost] Login(LoginModel request): This method handles the login request sent from the login form. It checks the login information, verifies the user information and then redirects to the main admin page or displays an error message if any.

GetPaging(string keyWord, int pageIndex = 1, int pageSize = 5): This method returns paged data of user accounts in the system.

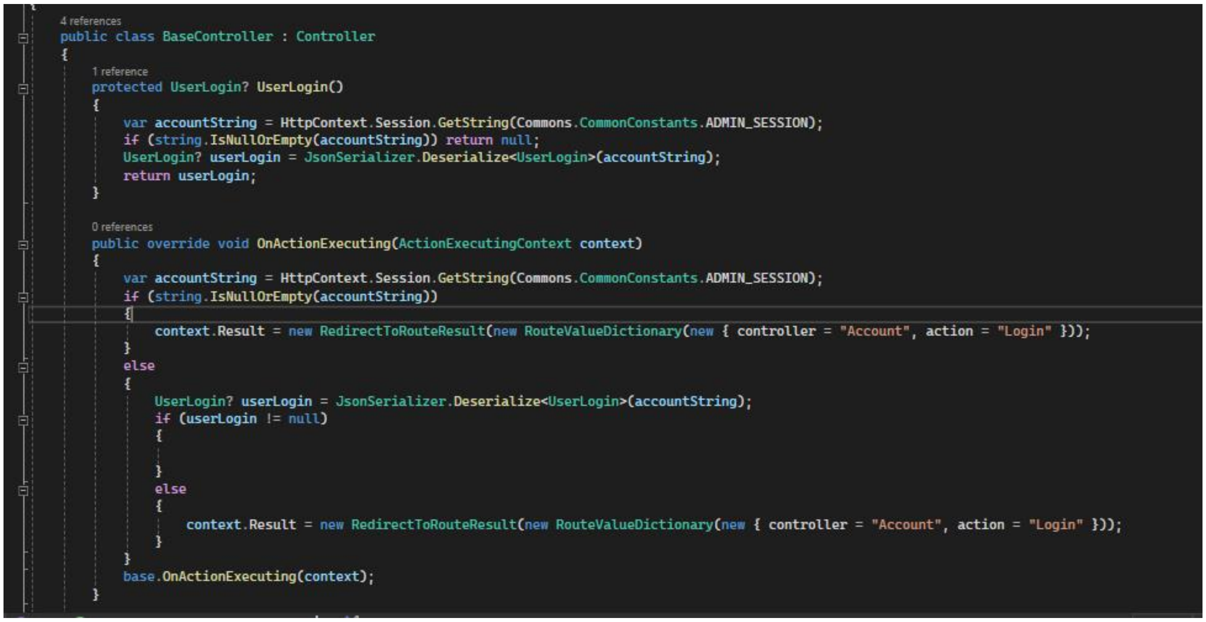
-[HttpPost] Delete(int id): This method handles a request to delete a user account from the system. It then sets TempData to display the resulting message to the user.

-[HttpPost] LockAccount(int id): This method handles a request to lock a user account.

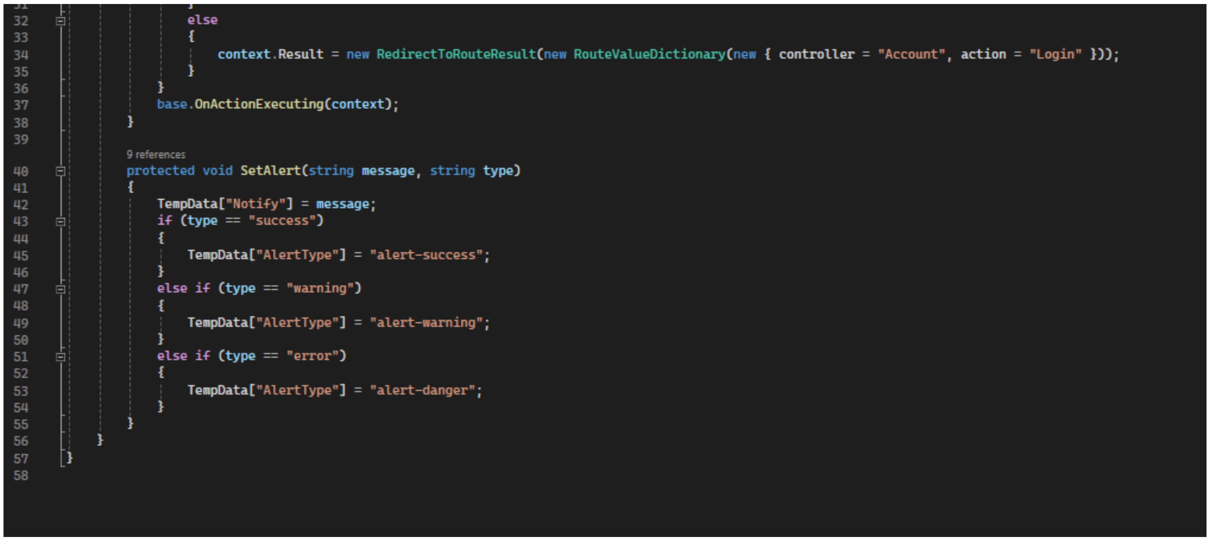
-[HttpPost] UnLockAccount(int id): This method handles a request to unlock a user account.

-Controller AccountController in the admin area is responsible for handling requests related to user account management in the system, including login, paging, deletion, locking and unlocking accounts.

BaseController



```
4 references
public class BaseController : Controller
{
    1 reference
    protected UserLogin? UserLogin()
    {
        var accountString = HttpContext.Session.GetString(Commons.CommonConstants.ADMIN_SESSION);
        if (string.IsNullOrEmpty(accountString)) return null;
        UserLogin? userLogin = JsonSerializer.Deserialize<UserLogin>(accountString);
        return userLogin;
    }

    0 references
    public override void OnActionExecuting(ActionExecutingContext context)
    {
        var accountString = HttpContext.Session.GetString(Commons.CommonConstants.ADMIN_SESSION);
        if (string.IsNullOrEmpty(accountString))
        {
            context.Result = new RedirectToRouteResult(new RouteValueDictionary(new { controller = "Account", action = "Login" }));
        }
        else
        {
            UserLogin? userLogin = JsonSerializer.Deserialize<UserLogin>(accountString);
            if (userLogin != null)
            {

            }
            else
            {
                context.Result = new RedirectToRouteResult(new RouteValueDictionary(new { controller = "Account", action = "Login" }));
            }
        }
        base.OnActionExecuting(context);
    }
```



```
                else
                {
                    context.Result = new RedirectToRouteResult(new RouteValueDictionary(new { controller = "Account", action = "Login" }));
                }
            }
            base.OnActionExecuting(context);
        }

        9 references
        protected void SetAlert(string message, string type)
        {
            TempData["Notify"] = message;
            if (type == "success")
            {
                TempData["AlertType"] = "alert-success";
            }
            else if (type == "warning")
            {
                TempData["AlertType"] = "alert-warning";
            }
            else if (type == "error")
            {
                TempData["AlertType"] = "alert-danger";
            }
        }
    }
}
```

This section of code defines a BaseController in the Admin area of the ASP.NET Core web application. BaseController is a base class for other controllers in the admin area and provides common functionality to them.

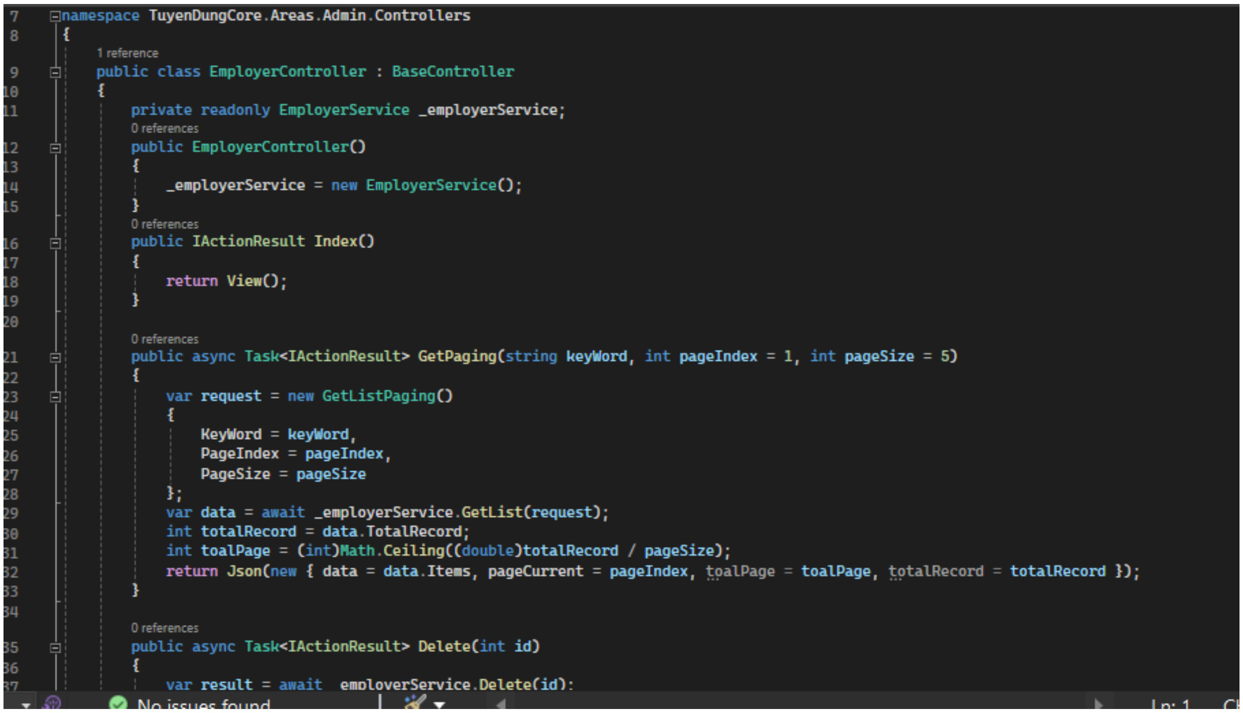
description of the functional parts of BaseController:

UserLogin(): This method is used to get logged in user information from session. It returns a UserLogin object containing user information, including login name, permissions, etc.

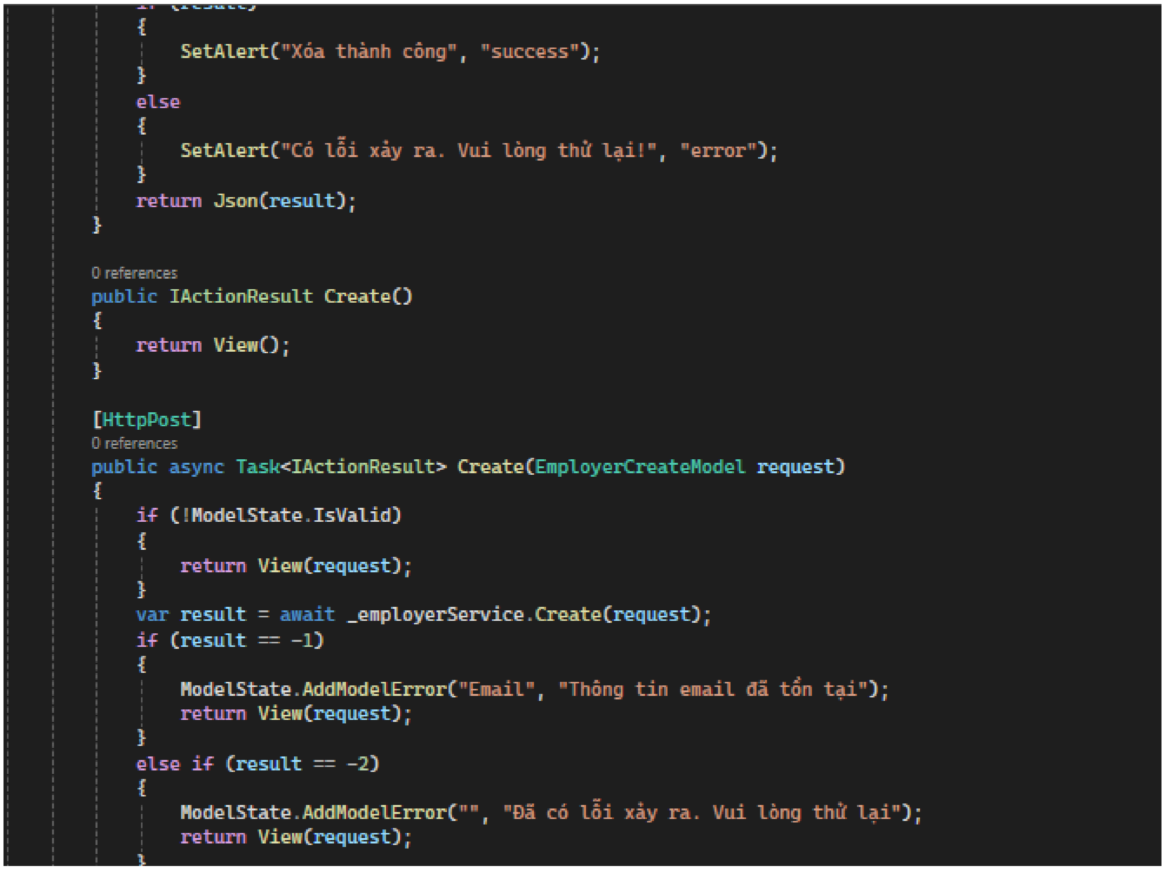
OnActionExecuting(ActionExecutingContext context): This method is overridden from the Controller base class and is called before an action in the controller is executed. In this method, BaseController checks if the user is logged in by checking the session. If there is no user information in the session, it will redirect the user to the login page.

SetAlert(string message, string type): This method is used to set the alert message (alert) into TempData to display to the user. TempData is a mechanism in ASP.NET Core to pass data from one action to another within the same request. This method takes a message and a warning type (success, warning, error) and puts them in TempData.

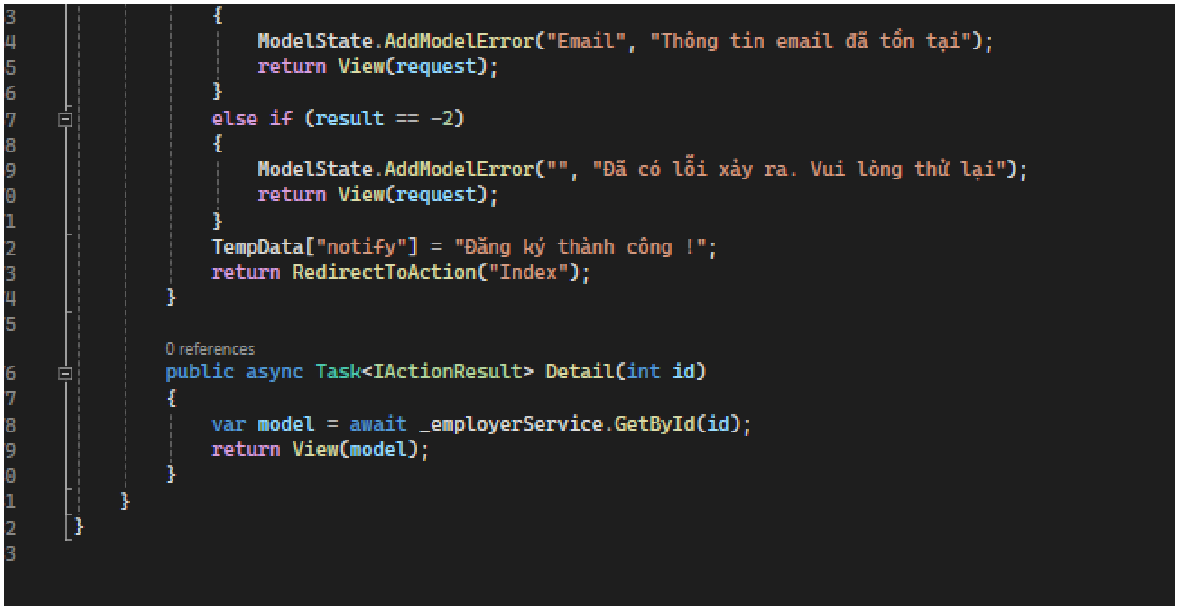
EmployerController



```
namespace TuyenDungCore.Areas.Admin.Controllers
{
    1 reference
    public class EmployerController : BaseController
    {
        private readonly EmployerService _employerService;
        0 references
        public EmployerController()
        {
            _employerService = new EmployerService();
        }
        0 references
        public IActionResult Index()
        {
            return View();
        }

        0 references
        public async Task<IActionResult> GetPaging(string keyWord, int pageIndex = 1, int pageSize = 5)
        {
            var request = new GetListPaging()
            {
                KeyWord = keyWord,
                PageIndex = pageIndex,
                PageSize = pageSize
            };
            var data = await _employerService.GetList(request);
            int totalRecord = data.TotalRecord;
            int toalPage = (int)Math.Ceiling((double)totalRecord / pageSize);
            return Json(new { data = data.Items, pageCurrent = pageIndex, toalPage = toalPage, totalRecord = totalRecord });
        }

        0 references
        public async Task<IActionResult> Delete(int id)
        {
            var result = await _employerService.Delete(id);
```



```
            if (result)
            {
                SetAlert("Xóa thành công", "success");
            }
            else
            {
                SetAlert("Có lỗi xảy ra. Vui lòng thử lại!", "error");
            }
            return Json(result);
        }

        0 references
        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> Create(EmployerCreateModel request)
        {
            if (!ModelState.IsValid)
            {
                return View(request);
            }
            var result = await _employerService.Create(request);
            if (result == -1)
            {
                ModelState.AddModelError("Email", "Thông tin email đã tồn tại");
                return View(request);
            }
            else if (result == -2)
            {
                ModelState.AddModelError("", "Đã có lỗi xảy ra. Vui lòng thử lại");
                return View(request);
            }
```



```
            {
                ModelState.AddModelError("Email", "Thông tin email đã tồn tại");
                return View(request);
            }
            else if (result == -2)
            {
                ModelState.AddModelError("", "Đã có lỗi xảy ra. Vui lòng thử lại");
                return View(request);
            }
            TempData["notify"] = "Đăng ký thành công !";
            return RedirectToAction("Index");
        }

        0 references
        public async Task<IActionResult> Detail(int id)
        {
            var model = await _employerService.GetById(id);
            return View(model);
        }
    }
}
```

This component is a controller in the TuyenDungCore software's ASP.NET Core MVC application. This controller is in charge of directing user requests and communicating with the RecruitService service in order to carry out tasks related to the maintenance of employer information.

-Constructor: Controller starts an instance of RecruitService for use in its methods.

-Index: Action return one view to display the main page of the employer management function.

-GetPaging: Processing this action requires paging the list of employers. It calls RecruitService's GetList method, which then returns the data as JSON to the browser.

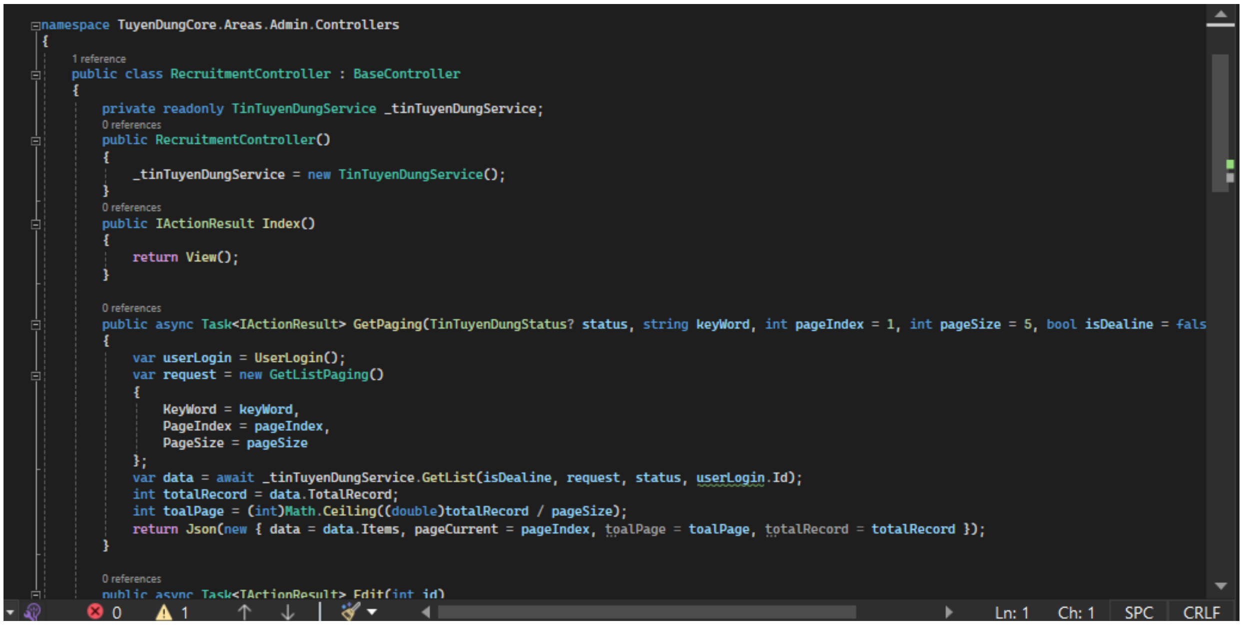
-Delete: This action processes a request to delete a recruiter. After deletion, it does not set an alert using the SetAlert method and returns the result of the delete operation as JSON.

-Create: This action returns a view to create a new employer. If the data is invalid, it will re-render the view with an error message.

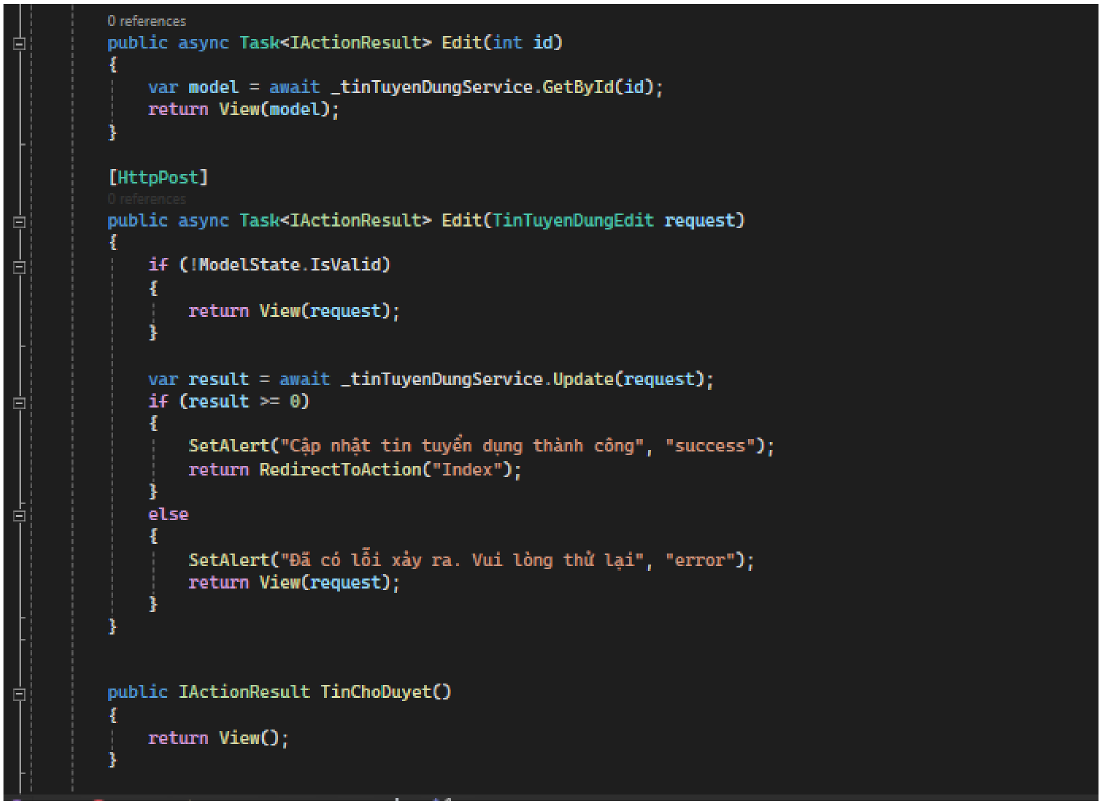
-Create (POST): Processing this action requires creating a new employer. If created successfully, it will redirect to the main page and set a notification via TempData.

-Details: This action returns a view showing details of an employer based on the specified ID.

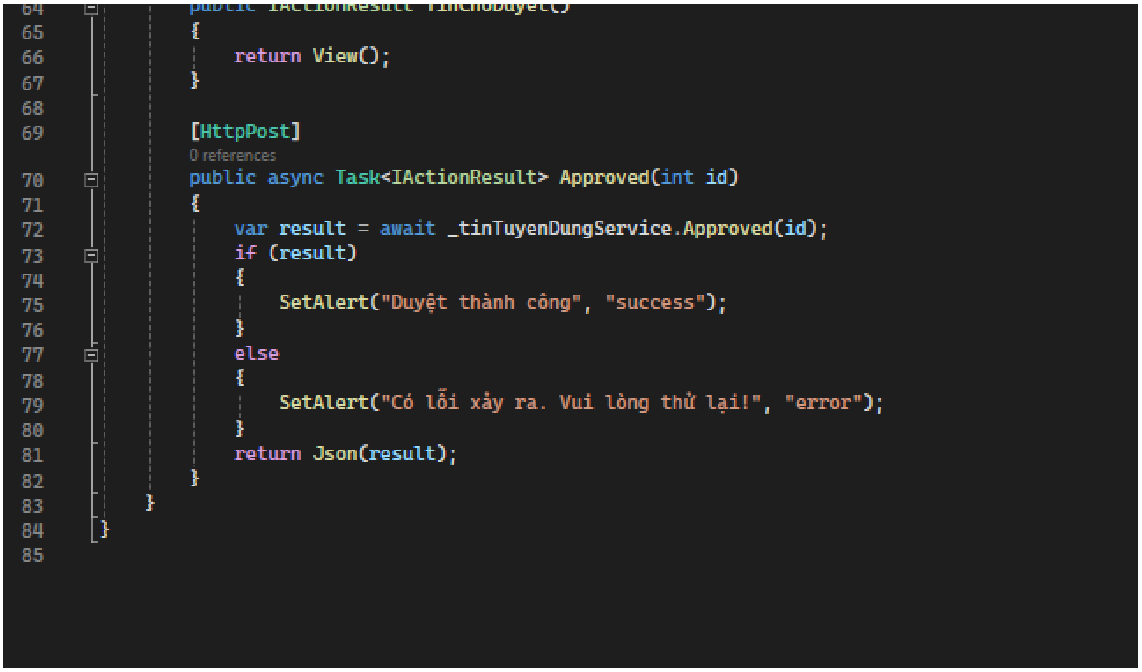
`RecruitmentController`



```
namespace TuyenDungCore.Areas.Admin.Controllers
{
    1 reference
    public class RecruitmentController : BaseController
    {
        private readonly TinTuyenDungService _tinTuyenDungService;
        0 references
        public RecruitmentController()
        {
            _tinTuyenDungService = new TinTuyenDungService();
        }
        0 references
        public IActionResult Index()
        {
            return View();
        }

        0 references
        public async Task<IActionResult> GetPaging(TinTuyenDungStatus? status, string keyWord, int pageIndex = 1, int pageSize = 5, bool isDealine = fals
        {
            var userLogin = UserLogin();
            var request = new GetListPaging()
            {
                KeyWord = keyWord,
                PageIndex = pageIndex,
                PageSize = pageSize
            };
            var data = await _tinTuyenDungService.GetList(isDealine, request, status, userLogin.Id);
            int totalRecord = data.TotalRecord;
            int toalPage = (int)Math.Ceiling((double)totalRecord / pageSize);
            return Json(new { data = data.Items, pageCurrent = pageIndex, toalPage = toalPage, totalRecord = totalRecord });
        }

        0 references
        public async Task<IActionResult> Edit(int id)
```



```
        0 references
        public async Task<IActionResult> Edit(int id)
        {
            var model = await _tinTuyenDungService.GetById(id);
            return View(model);
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> Edit(TinTuyenDungEdit request)
        {
            if (!ModelState.IsValid)
            {
                return View(request);
            }

            var result = await _tinTuyenDungService.Update(request);
            if (result >= 0)
            {
                SetAlert("Cập nhật tin tuyển dụng thành công", "success");
                return RedirectToAction("Index");
            }
            else
            {
                SetAlert("Đã có lỗi xảy ra. Vui lòng thử lại", "error");
                return View(request);
            }
        }


        public IActionResult TinChoDuyet()
        {
            return View();
        }
```



```
        public IActionResult TinChoDuyet()
        {
            return View();
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> Approved(int id)
        {
            var result = await _tinTuyenDungService.Approved(id);
            if (result)
            {
                SetAlert("Duyệt thành công", "success");
            }
            else
            {
                SetAlert("Có lỗi xảy ra. Vui lòng thử lại!", "error");
            }
            return Json(result);
        }
    }
}
```

This is another controller in your ASP.NET Core MVC application, used to manage job postings.

-Constructor: Similar to the previous controller, this is where you initialize an instance of TinTuyenDungService to use in the controller methods.

-Index: This action returns a view to display the main page of the job posting management function.

-GetPaging: This action handles the request to paginate the list of job postings. It calls TinTuyenDungService's GetList method, then returns the data as JSON to the browser.

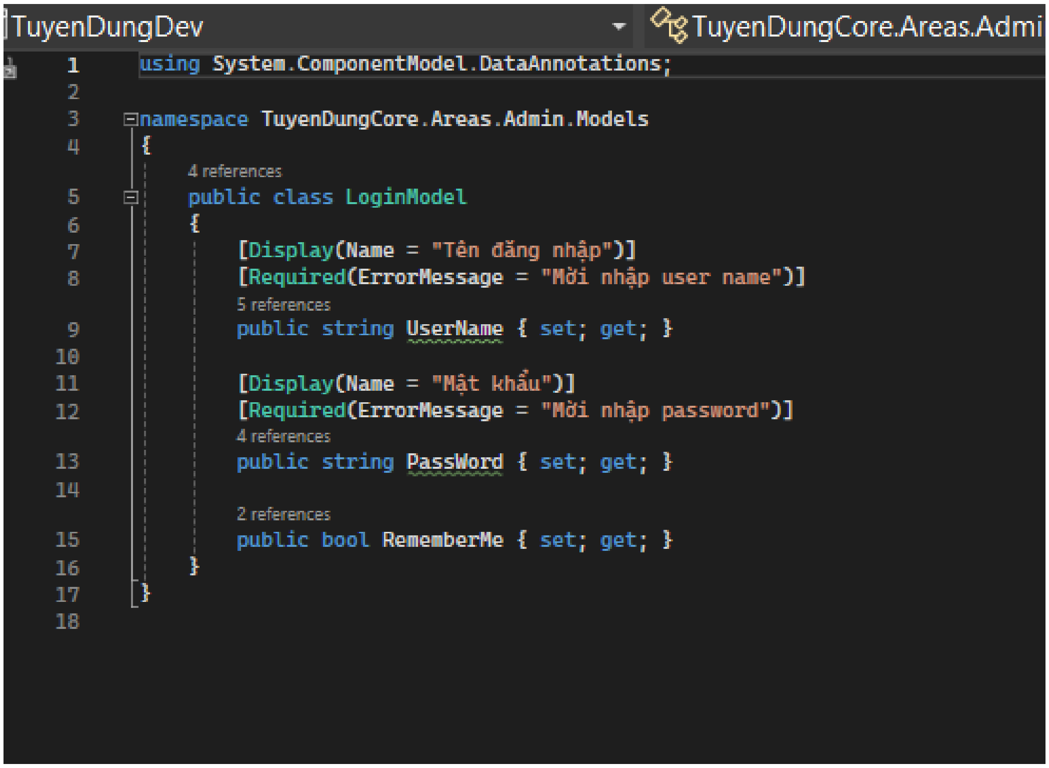
-Edit: This action returns a view that allows editing information of a job posting based on the specified ID.

-Edit (POST): This action processes a request to update information of a job posting. If the update is successful, it redirects to the main page and sets a success message, otherwise if there is an error, it redisplays the edit view with the error message.

-TinChoDuyet: This action returns a view that allows browsing job postings.

-Approved (POST): This action processes a request to approve a job posting. After browsing, it sets up a message and returns the result of the browsing operation as JSON.

LoginModel



```
using System.ComponentModel.DataAnnotations;

namespace TuyenDungCore.Areas.Admin.Models
{
    4 references
    public class LoginModel
    {
        [Display(Name = "Tên đăng nhập")]
        [Required(ErrorMessage = "Mời nhập user name")]
        5 references
        public string UserName { set; get; }

        [Display(Name = "Mật khẩu")]
        [Required(ErrorMessage = "Mời nhập password")]
        4 references
        public string PassWord { set; get; }

        2 references
        public bool RememberMe { set; get; }
    }
}
```

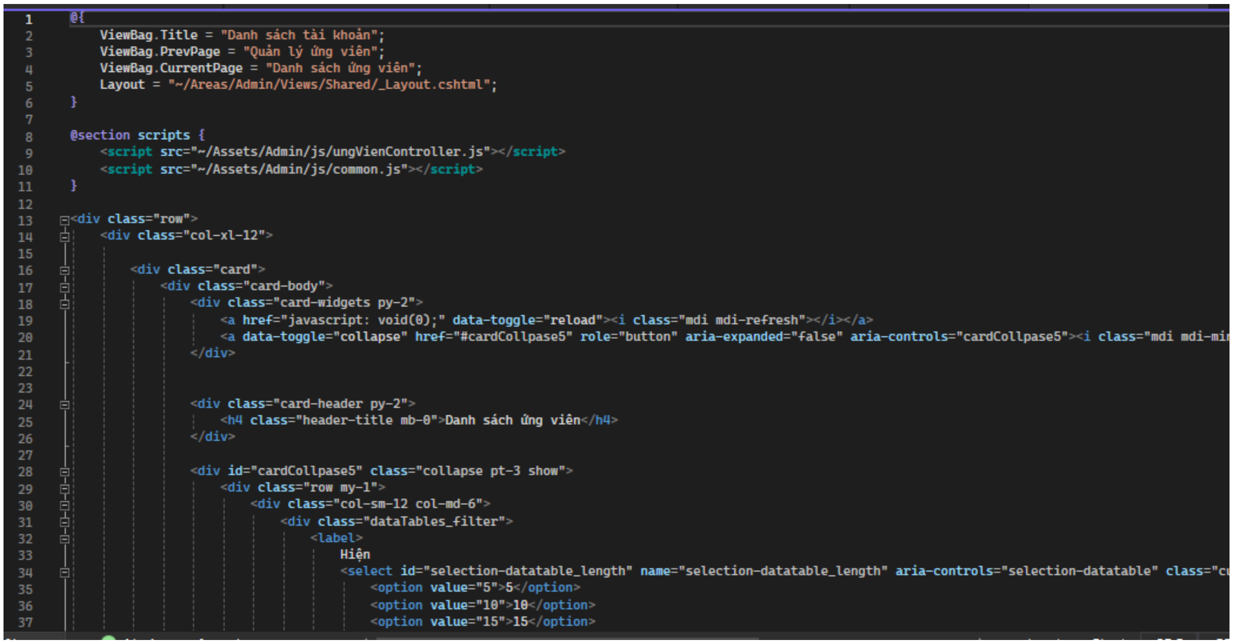
This is a model used in the admin interface part of my application, to hold user login information.

-UserName: This attribute is the user's login name. The Display property is used to define the display name for this field on the user interface. The Required attribute specifies that this field is required and an error message will be displayed if the user does not enter a value for this field.

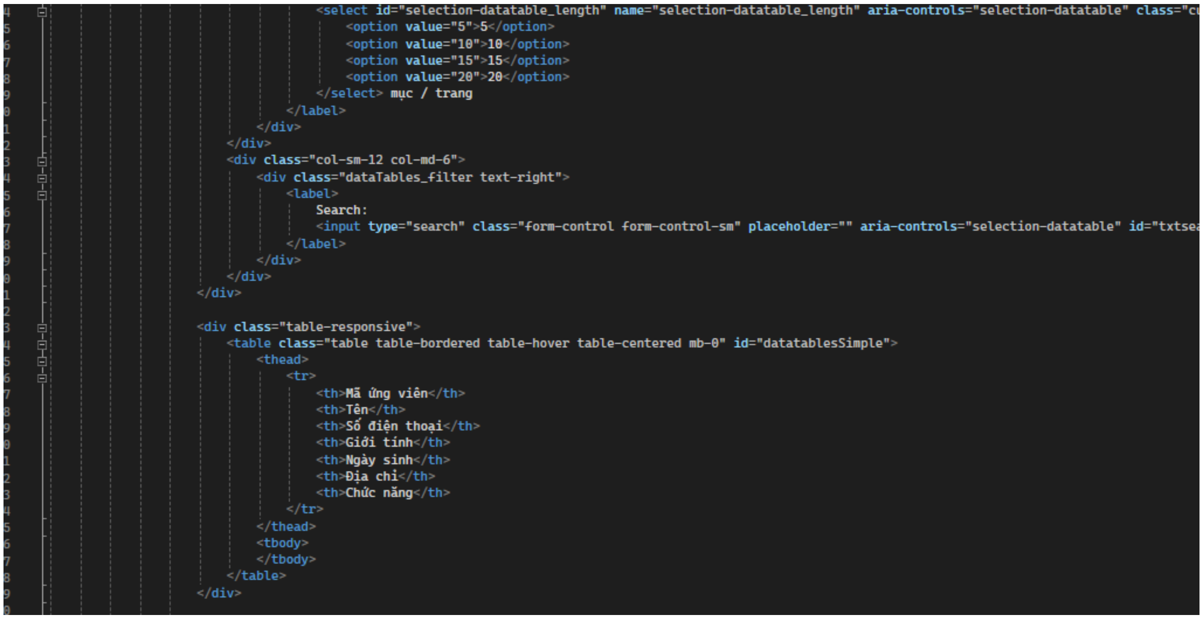
-PassWord: This attribute is the user's password. Similar to UserName, Display is used to define the display name for this field on the user interface. The Required attribute also specifies that this field is required and an error message will be displayed if the user does not enter a value for this field.

-RememberMe: This attribute is typically used to allow users to select "Remember Me" to maintain their login session.

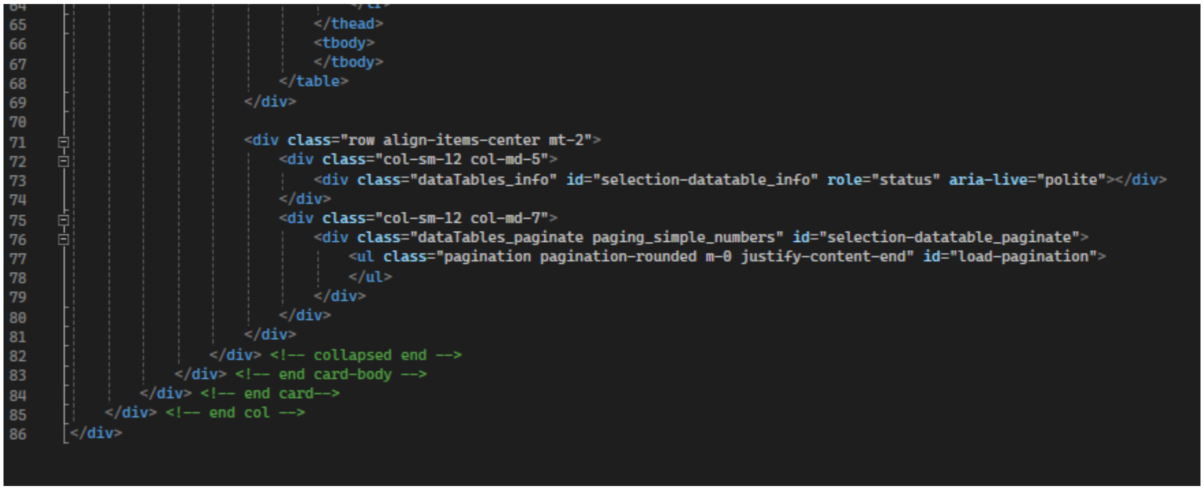
View-Candidate-Index



```
@{
    ViewBag.Title = "Danh sách tài khoản";
    ViewBag.PrevPage = "Quản lý ứng viên";
    ViewBag.CurrentPage = "Danh sách ứng viên";
    Layout = "~/Areas/Admin/Views/Shared/_Layout.cshtml";
}

@section scripts {
    <script src="~/Assets/Admin/js/ungVienController.js"></script>
    <script src="~/Assets/Admin/js/common.js"></script>
}

<div class="row">
    <div class="col-xl-12">

        <div class="card">
            <div class="card-body">
                <div class="card-widgets py-2">
                    <a href="javascript: void(0);" data-toggle="reload"><i class="mdi mdi-refresh"></i></a>
                    <a data-toggle="collapse" href="#cardCollpase5" role="button" aria-expanded="false" aria-controls="cardCollpase5"><i class="mdi mdi-mi
                </div>


                <div class="card-header py-2">
                    <h4 class="header-title mb-0">Danh sách ứng viên</h4>
                </div>

                <div id="cardCollpase5" class="collapse pt-3 show">
                    <div class="row my-1">
                        <div class="col-sm-12 col-md-6">
                            <div class="dataTables_filter">
                                <label>
                                    Hiện
                                    <select id="selection-datatable_length" name="selection-datatable_length" aria-controls="selection-datatable" class="cu
                                        <option value="5">5</option>
                                        <option value="10">10</option>
                                        <option value="15">15</option>
```



```
                <select id="selection-datatable_length" name="selection-datatable_length" aria-controls="selection-datatable" class="cu
                    <option value="5">5</option>
                    <option value="10">10</option>
                    <option value="15">15</option>
                    <option value="20">20</option>
                </select> mục / trang
            </label>
        </div>
    </div>
    <div class="col-sm-12 col-md-6">
        <div class="dataTables_filter text-right">
            <label>
                Search:
                <input type="search" class="form-control form-control-sm" placeholder="" aria-controls="selection-datatable" id="txtsea
            </label>
        </div>
    </div>
</div>

<div class="table-responsive">
    <table class="table table-bordered table-hover table-centered mb-0" id="datatablesSimple">
        <thead>
            <tr>
                <th>Mã ứng viên</th>
                <th>Tên</th>
                <th>Số điện thoại</th>
                <th>Giới tính</th>
                <th>Ngày sinh</th>
                <th>Địa chỉ</th>
                <th>Chức năng</th>
            </tr>
        </thead>
        <tbody>
        </tbody>
    </table>
</div>
```



```
                            </thead>
                            <tbody>
                            </tbody>
                        </table>
                    </div>

                    <div class="row align-items-center mt-2">
                        <div class="col-sm-12 col-md-5">
                            <div class="dataTables_info" id="selection-datatable_info" role="status" aria-live="polite"></div>
                        </div>
                        <div class="col-sm-12 col-md-7">
                            <div class="dataTables_paginate paging_simple_numbers" id="selection-datatable_paginate">
                                <ul class="pagination pagination-rounded m-0 justify-content-end" id="load-pagination">
                                </ul>
                            </div>
                        </div>
                    </div>
                </div> <!-- collapsed end -->
            </div> <!-- end card-body -->
        </div> <!-- end card-->
    </div> <!-- end col -->
</div>
```

This Razor view appears to be a page for displaying a list of candidates -. Let's break down its structure and functionality:

-ViewBag: The ViewBag is used to pass data from the controller to the view. In this case, it sets the title of the page (ViewBag.Title), the previous page link (ViewBag.PrevPage), and the current page (ViewBag.CurrentPage). These values are then used within the view.

Layout: Specifies the layout file (_Layout.cshtml) to be used for this view. Layout files typically contain shared HTML structure and scripts across multiple views.

-Scripts Section: Defines scripts to be included at the end of the view. In this case, it includes two JavaScript files: ungVienController.js and common.js. These scripts likely contain client-side logic for interacting with and manipulating the candidate data.

-HTML Structure: The view is divided into several sections:

Card: The main content area is enclosed within a Bootstrap card (<div class="card">). This card contains the list of candidates.

Header: The header of the card displays the title "Danh sách ứng viên" (List of candidates).

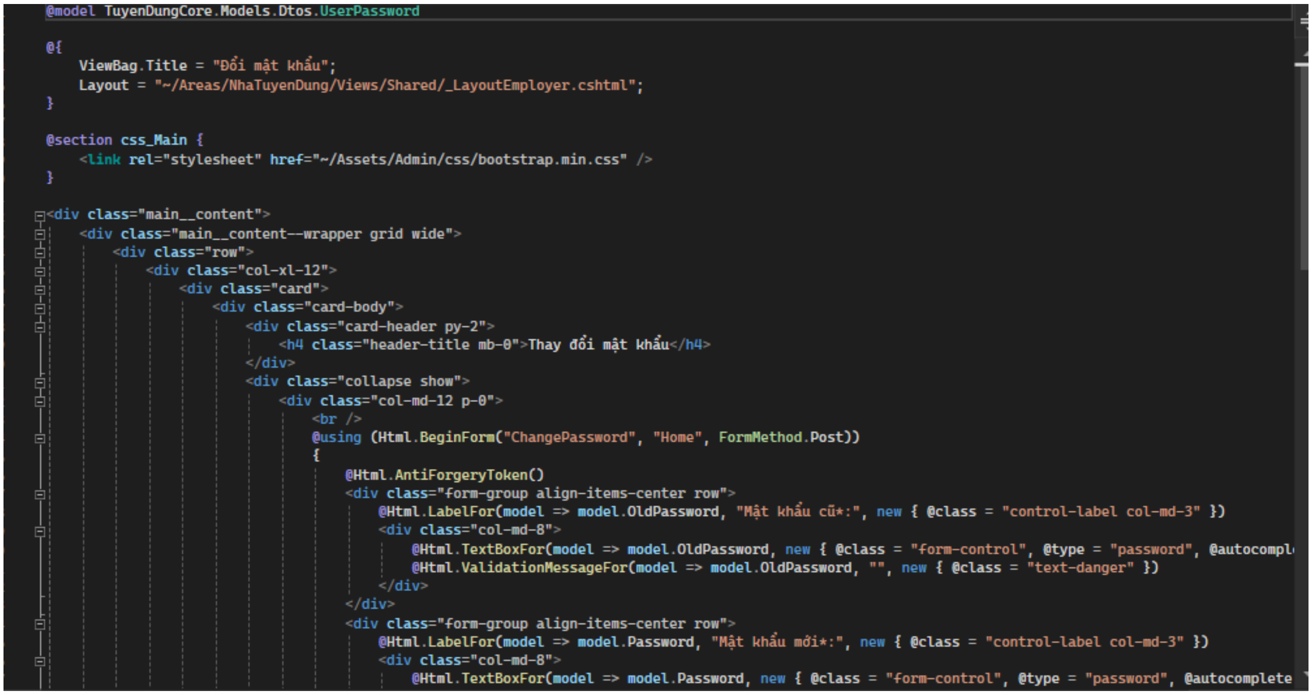
-Search and Pagination: Below the header, there are search and pagination controls. Users can search for candidates using a search input field (<input type="search">). Pagination controls allow users to navigate between pages of candidate data.

-Table: The candidate data is displayed in a table (<table>). Each row in the table represents a candidate, with columns for candidate details such as ID, name, phone number, gender, date of birth, address, and functions (possibly actions such as edit or delete).

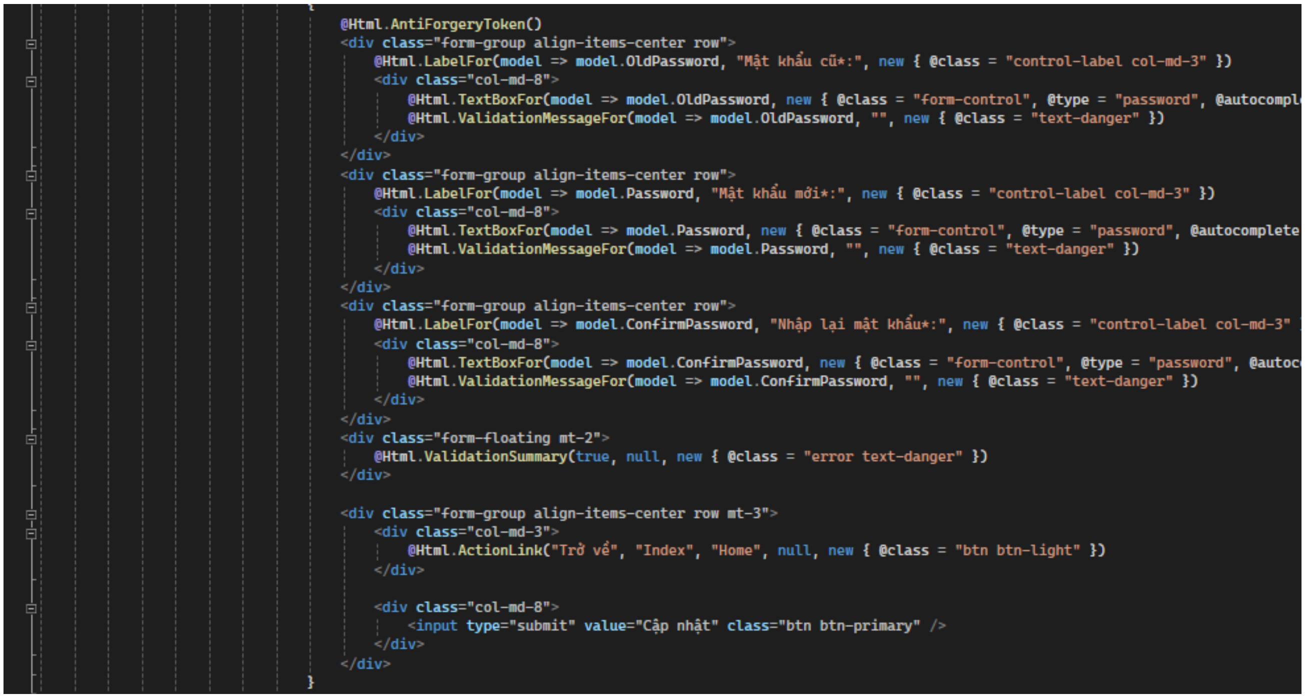
-Data Binding: The candidate data is dynamically loaded into the table body (<tbody>) using JavaScript. This likely involves making AJAX requests to the server to fetch candidate data.

-JavaScript Interactions: JavaScript files (ungVienController.js and common.js) are included to handle interactions such as fetching candidate data, filtering, pagination, and any other client-side functionality.

View-Home-ChangePassword



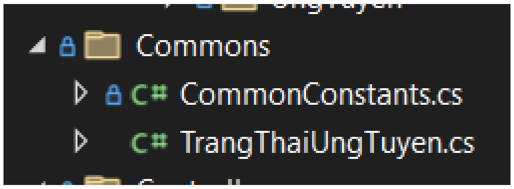
```
@model TuyenDungCore.Models.Dtos.UserPassword

@{
    ViewBag.Title = "Đổi mật khẩu";
    Layout = "~/Areas/NhaTuyenDung/Views/Shared/_LayoutEmployer.cshtml";
}

@section css_Main {
    <link rel="stylesheet" href="~/Assets/Admin/css/bootstrap.min.css" />
}

<div class="main__content">
    <div class="main__content--wrapper grid wide">
        <div class="row">
            <div class="col-xl-12">
                <div class="card">
                    <div class="card-body">
                        <div class="card-header py-2">
                            <h4 class="header-title mb-0">Thay đổi mật khẩu</h4>
                        </div>
                        <div class="collapse show">
                            <div class="col-md-12 p-0">
                                <br />
                                @using (Html.BeginForm("ChangePassword", "Home", FormMethod.Post))
                                {
                                    @Html.AntiForgeryToken()
                                    <div class="form-group align-items-center row">
                                        @Html.LabelFor(model => model.OldPassword, "Mật khẩu cũ*:", new { @class = "control-label col-md-3" })
                                        <div class="col-md-8">
                                            @Html.TextBoxFor(model => model.OldPassword, new { @class = "form-control", @type = "password", @autocompl
                                            @Html.ValidationMessageFor(model => model.OldPassword, "", new { @class = "text-danger" })
                                        </div>
                                    </div>
                                    <div class="form-group align-items-center row">
                                        @Html.LabelFor(model => model.Password, "Mật khẩu mới*:", new { @class = "control-label col-md-3" })
                                        <div class="col-md-8">
                                            @Html.TextBoxFor(model => model.Password, new { @class = "form-control", @type = "password", @autocomplete
```



```
                                {
                                    @Html.AntiForgeryToken()
                                    <div class="form-group align-items-center row">
                                        @Html.LabelFor(model => model.OldPassword, "Mật khẩu cũ*:", new { @class = "control-label col-md-3" })
                                        <div class="col-md-8">
                                            @Html.TextBoxFor(model => model.OldPassword, new { @class = "form-control", @type = "password", @autocompl
                                            @Html.ValidationMessageFor(model => model.OldPassword, "", new { @class = "text-danger" })
                                        </div>
                                    </div>
                                    <div class="form-group align-items-center row">
                                        @Html.LabelFor(model => model.Password, "Mật khẩu mới*:", new { @class = "control-label col-md-3" })
                                        <div class="col-md-8">
                                            @Html.TextBoxFor(model => model.Password, new { @class = "form-control", @type = "password", @autocomplete
                                            @Html.ValidationMessageFor(model => model.Password, "", new { @class = "text-danger" })
                                        </div>
                                    </div>
                                    <div class="form-group align-items-center row">
                                        @Html.LabelFor(model => model.ConfirmPassword, "Nhập lại mật khẩu*:", new { @class = "control-label col-md-3"
                                        <div class="col-md-8">
                                            @Html.TextBoxFor(model => model.ConfirmPassword, new { @class = "form-control", @type = "password", @autoc
                                            @Html.ValidationMessageFor(model => model.ConfirmPassword, "", new { @class = "text-danger" })
                                        </div>
                                    </div>
                                    <div class="form-floating mt-2">
                                        @Html.ValidationSummary(true, null, new { @class = "error text-danger" })
                                    </div>

                                    <div class="form-group align-items-center row mt-3">
                                        <div class="col-md-3">
                                            @Html.ActionLink("Trở về", "Index", "Home", null, new { @class = "btn btn-light" })
                                        </div>

                                        <div class="col-md-8">
                                            <input type="submit" value="Cập nhật" class="btn btn-primary" />
                                        </div>
                                    </div>
                                }
```

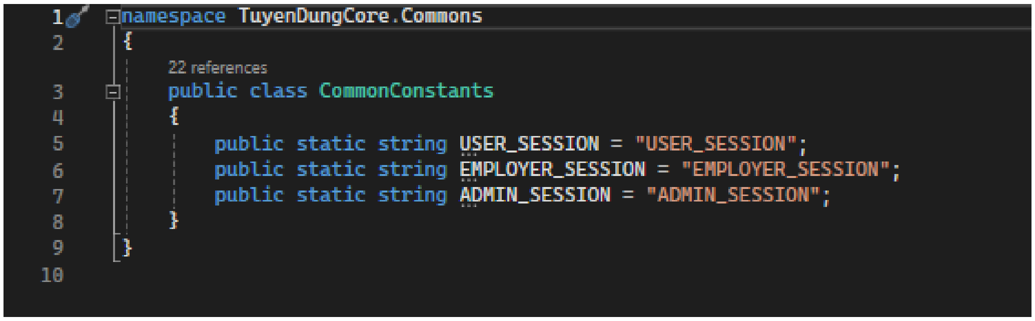
this view provides a user interface for users to change their password, with validation for each field and navigation buttons for convenience.

Common



Commons
CommonConstants.cs
TrangThaiUngTuyen.cs

This is a common part of the system to make it more convenient to call functions without having to rewrite code many times.



```
namespace TuyenDungCore.Commons
{
    22 references
    public class CommonConstants
    {
        public static string USER_SESSION = "USER_SESSION";
        public static string EMPLOYER_SESSION = "EMPLOYER_SESSION";
        public static string ADMIN_SESSION = "ADMIN_SESSION";
    }
}
```

This CommonConstants class defines constant strings that represent session keys commonly used in the TuyenDungCore application:

USER_SESSION: Represents the session key for storing user-related information.

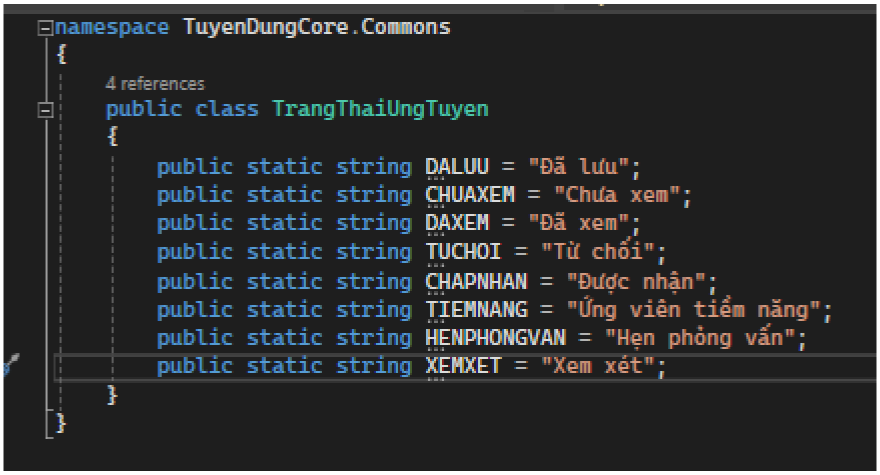
EMPLOYER_SESSION: Represents the session key for storing employer-related information.

ADMIN_SESSION: Represents the session key for storing admin-related information.

By using constants like these, you ensure consistency and avoid hardcoding session key strings throughout your application. This makes your code more maintainable and reduces the likelihood of errors caused by typos or inconsistencies.

DAO(Data Access Object)

This place is where services that interact with the database are stored, similar to the Repository.

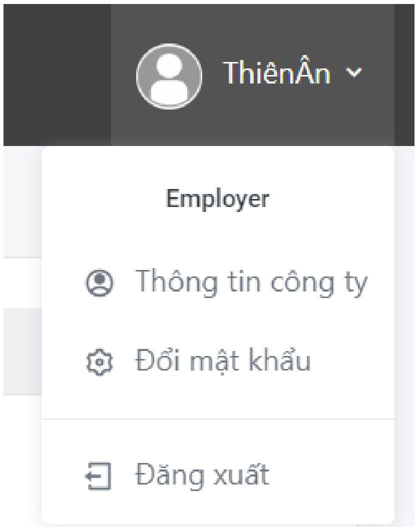


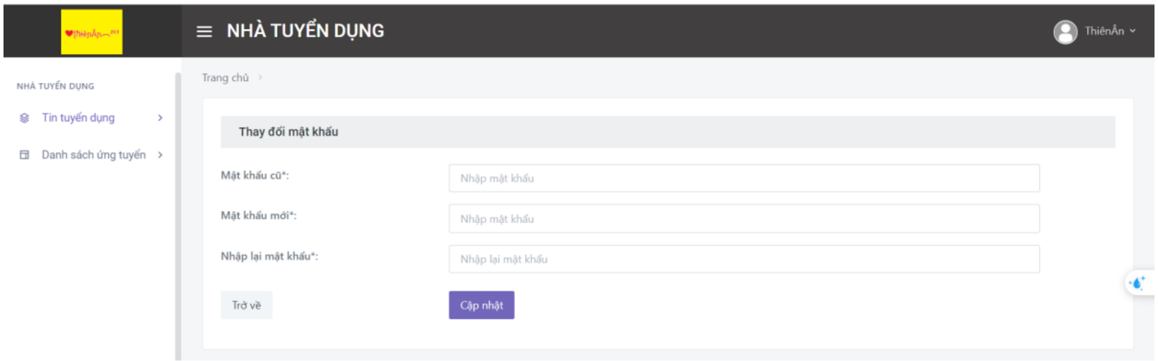
```
namespace TuyenDungCore.Commons
{
    4 references
    public class TrangThaiUngTuyen
    {
        public static string DALUU = "Đã lưu";
        public static string CHUAXEM = "Chưa xem";
        public static string DAXEM = "Đã xem";
        public static string TUCHOI = "Từ chối";
        public static string CHAPNHAN = "Được nhận";
        public static string TIEMNANG = "Ứng viên tiềm năng";
        public static string HENPHONGVAN = "Hẹn phỏng vấn";
        public static string XEMXET = "Xem xét";
    }
}
```

-Application statuses

Required functions:

- The system shall provide a secure login mechanism for both employers and job seekers.



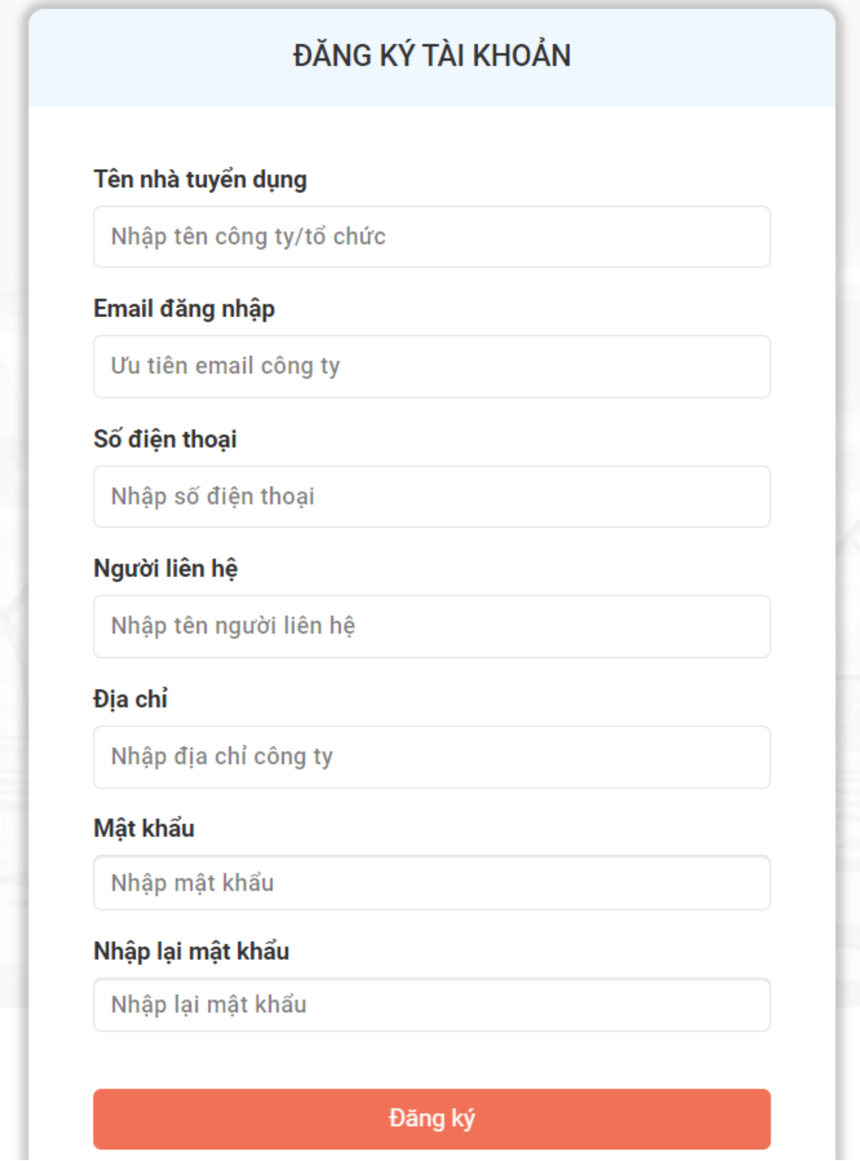


Thay đổi mật khẩu

Mật khẩu cũ*: Nhập mật khẩu

Mật khẩu mới*: Nhập mật khẩu

Nhập lại mật khẩu*: Nhập lại mật khẩu



ĐĂNG KÝ TÀI KHOẢN

Tên nhà tuyển dụng

Nhập tên công ty/tổ chức

Email đăng nhập

Ưu tiên email công ty

Số điện thoại

Nhập số điện thoại

Người liên hệ

Nhập tên người liên hệ

Địa chỉ

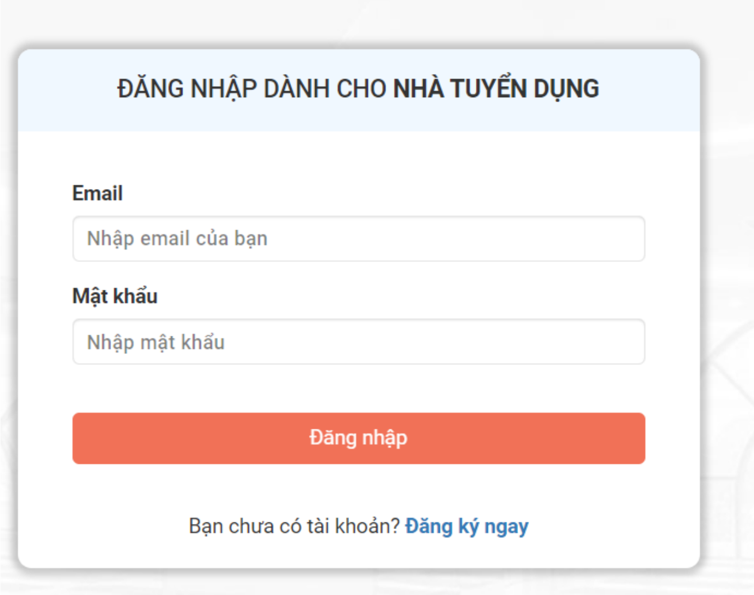
Nhập địa chỉ công ty

Mật khẩu

Nhập mật khẩu

Nhập lại mật khẩu

Nhập lại mật khẩu



ĐĂNG NHẬP DÀNH CHO NHÀ TUYỂN DỤNG

Email

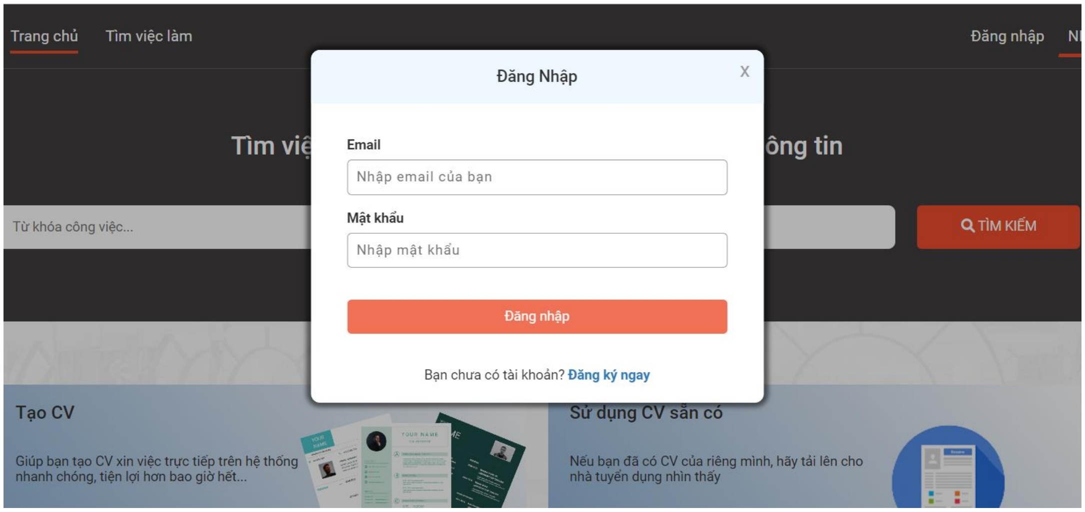
Nhập email của bạn

Mật khẩu

Nhập mật khẩu

Đăng nhập

Bạn chưa có tài khoản? **Đăng ký ngay**



Đăng Nhập

X

Email

Nhập email của bạn

Mật khẩu

Nhập mật khẩu

Đăng nhập

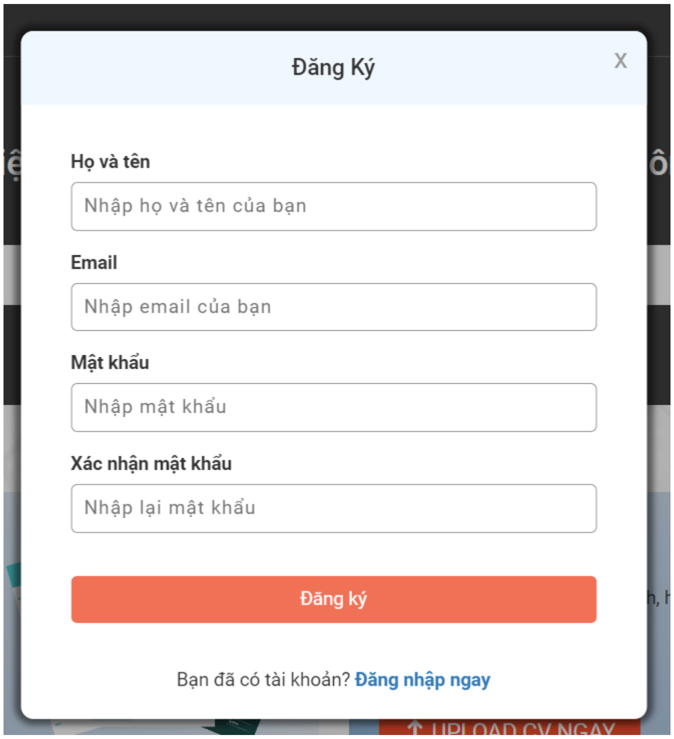
Bạn chưa có tài khoản? **Đăng ký ngay**

Tạo CV

Giúp bạn tạo CV xin việc trực tiếp trên hệ thống nhanh chóng, tiện lợi hơn bao giờ hết...

Sử dụng CV sẵn có

Nếu bạn đã có CV của riêng mình, hãy tải lên cho nhà tuyển dụng nhìn thấy



Đăng Ký

Họ và tên

Nhập họ và tên của bạn

Email

Nhập email của bạn

Mật khẩu

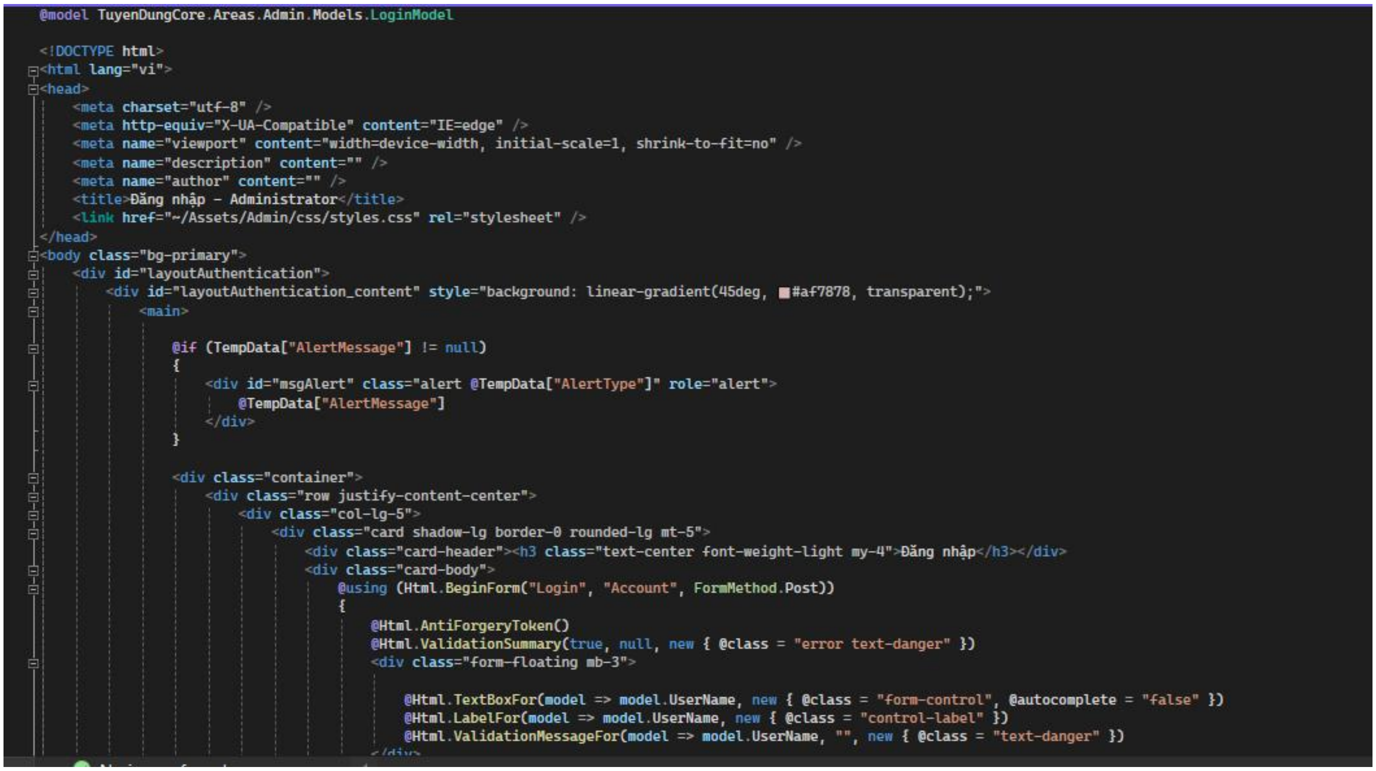
Nhập mật khẩu

Xác nhận mật khẩu

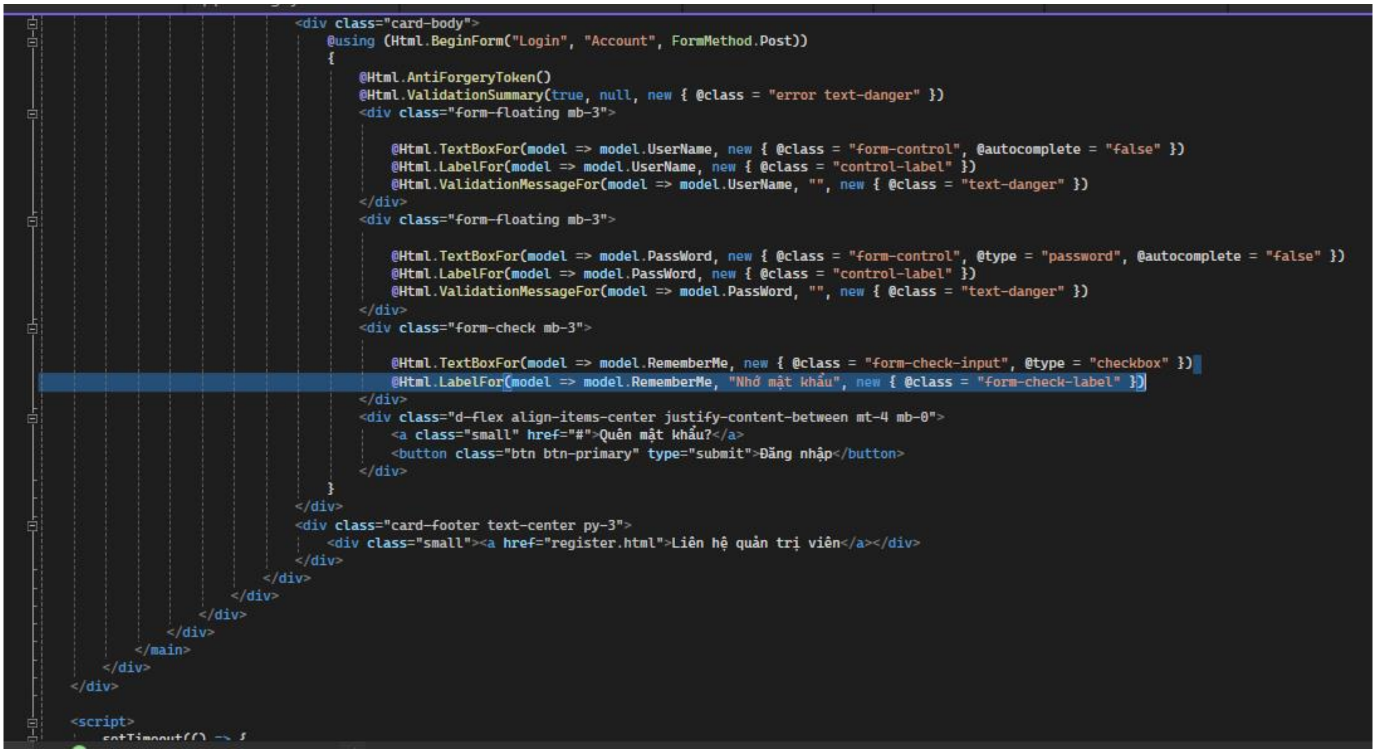
Nhập lại mật khẩu

Bạn đã có tài khoản? Đăng nhập ngay

Code:



```
@model TuyenDungCore.Areas.Admin.Models.LoginModel

<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <title>Đăng nhập - Administrator</title>
    <link href="~/Assets/Admin/css/styles.css" rel="stylesheet" />
</head>
<body class="bg-primary">
    <div id="layoutAuthentication">
        <div id="layoutAuthentication_content" style="background: linear-gradient(45deg, ■#af7878, transparent);">
            <main>

                @if (TempData["AlertMessage"] != null)
                {
                    <div id="msgAlert" class="alert @TempData["AlertType"]" role="alert">
                        @TempData["AlertMessage"]
                    </div>
                }

                <div class="container">
                    <div class="row justify-content-center">
                        <div class="col-lg-5">
                            <div class="card shadow-lg border-0 rounded-lg mt-5">
                                <div class="card-header"><h3 class="text-center font-weight-light my-4">Đăng nhập</h3></div>
                                <div class="card-body">
                                    @using (Html.BeginForm("Login", "Account", FormMethod.Post))
                                    {
                                        @Html.AntiForgeryToken()
                                        @Html.ValidationSummary(true, null, new { @class = "error text-danger" })
                                        <div class="form-floating mb-3">

                                            @Html.TextBoxFor(model => model.UserName, new { @class = "form-control", @autocomplete = "false" })
                                            @Html.LabelFor(model => model.UserName, new { @class = "control-label" })
                                            @Html.ValidationMessageFor(model => model.UserName, "", new { @class = "text-danger" })
                                        </div>
```



```
                                <div class="card-body">
                                    @using (Html.BeginForm("Login", "Account", FormMethod.Post))
                                    {
                                        @Html.AntiForgeryToken()
                                        @Html.ValidationSummary(true, null, new { @class = "error text-danger" })
                                        <div class="form-floating mb-3">

                                            @Html.TextBoxFor(model => model.UserName, new { @class = "form-control", @autocomplete = "false" })
                                            @Html.LabelFor(model => model.UserName, new { @class = "control-label" })
                                            @Html.ValidationMessageFor(model => model.UserName, "", new { @class = "text-danger" })
                                        </div>
                                        <div class="form-floating mb-3">

                                            @Html.TextBoxFor(model => model.PassWord, new { @class = "form-control", @type = "password", @autocomplete = "false" })
                                            @Html.LabelFor(model => model.PassWord, new { @class = "control-label" })
                                            @Html.ValidationMessageFor(model => model.PassWord, "", new { @class = "text-danger" })
                                        </div>
                                        <div class="form-check mb-3">

                                            @Html.TextBoxFor(model => model.RememberMe, new { @class = "form-check-input", @type = "checkbox" })
                                            @Html.LabelFor(model => model.RememberMe, "Nhớ mật khẩu", new { @class = "form-check-label" })
                                        </div>
                                        <div class="d-flex align-items-center justify-content-between mt-4 mb-0">
                                            <a class="small" href="#">Quên mật khẩu?</a>
                                            <button class="btn btn-primary" type="submit">Đăng nhập</button>
                                        </div>
                                    }
                                </div>
                                <div class="card-footer text-center py-3">
                                    <div class="small"><a href="register.html">Liên hệ quản trị viên</a></div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </main>
        </div>
    </div>

    <script>
        setTimeout(() => {
```

This code is an HTML page and the Razor view is used to display the login interface for the administrator. Let's go through each section:

-Model Declaration: First, the page declares a model used to bind the data of the login form.

-HTML Head Section: This section contains meta tags and links to identify information about the page, including charset, compatibility, viewport and description.

-CSS Stylesheet: The page uses a CSS file to customize the interface, linked via the <link> tag.

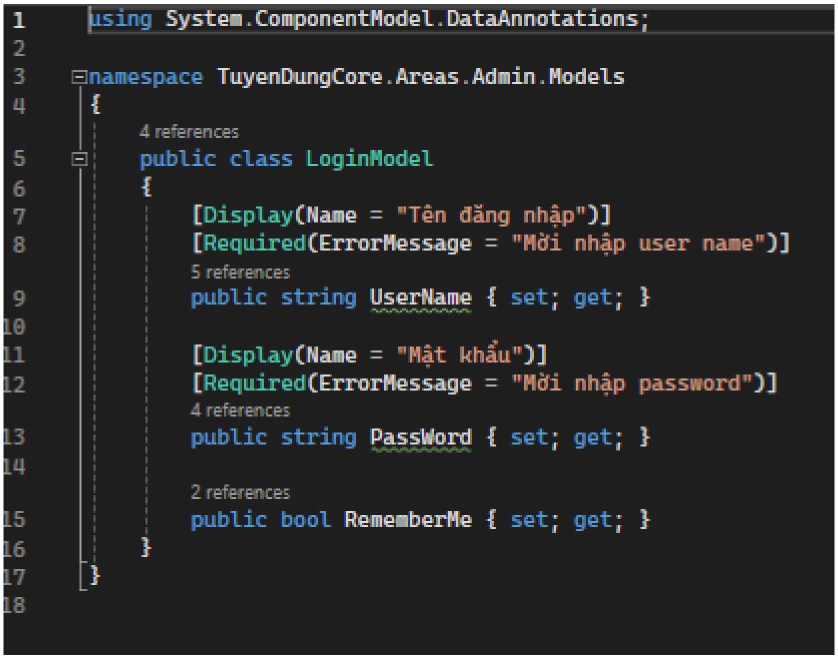
-Body Section: This section contains the content of the page.

-Alert Message: If there is a notification from TempData, it will be displayed here. The notification is displayed as a div with a class depending on the notification type, and the content of the notification is displayed from TempData["AlertMessage"].

-Form: The login form is placed in a div with class card and HTML elements such as form, input, and button to enter login information and perform login.

-JavaScript Timeout: This JavaScript snippet is used to automatically hide the notification after a certain amount of time (here 2 seconds) using jQuery's setTimeout() and fadeOut() functions.

In short, this code creates a login page for the administrator with a login form, and displays notifications (if any) from TempData when necessary.



```
using System.ComponentModel.DataAnnotations;

namespace TuyenDungCore.Areas.Admin.Models
{
    4 references
    public class LoginModel
    {
        [Display(Name = "Tên đăng nhập")]
        [Required(ErrorMessage = "Mời nhập user name")]
        5 references
        public string UserName { set; get; }

        [Display(Name = "Mật khẩu")]
        [Required(ErrorMessage = "Mời nhập password")]
        4 references
        public string PassWord { set; get; }

        2 references
        public bool RememberMe { set; get; }
    }
}
```

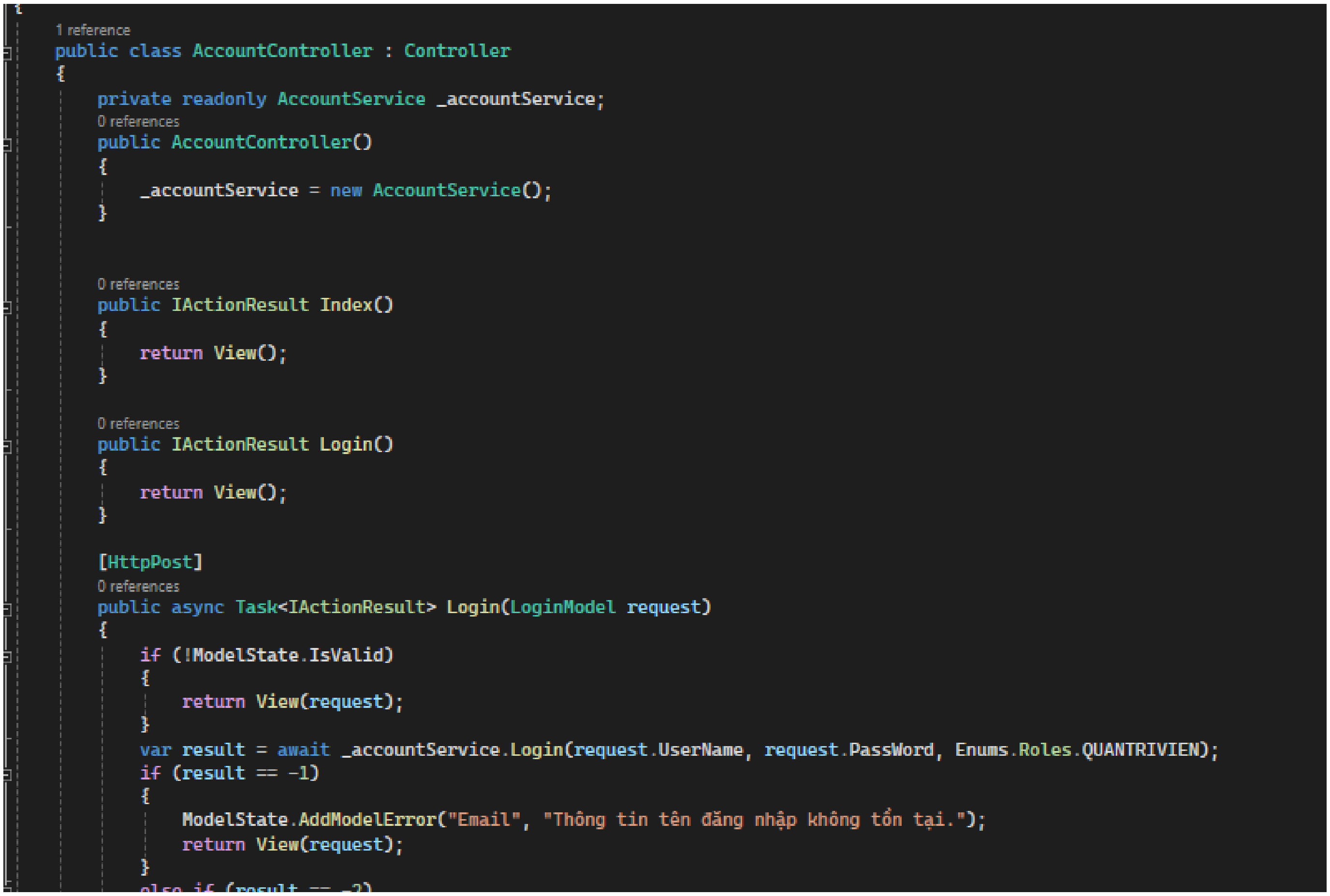
In this code, you define a LoginModel model in the TuyenDungCore.Areas.Admin.Models namespace. This is a class that describes the data used in the admin page's login form. Below are the components of the model:

-UserName: This attribute represents the user's login name. The Display property is used to specify the display name for this field on the user interface. The Required attribute specifies that this field is required and an error message will be displayed if the user does not enter a value for this field.

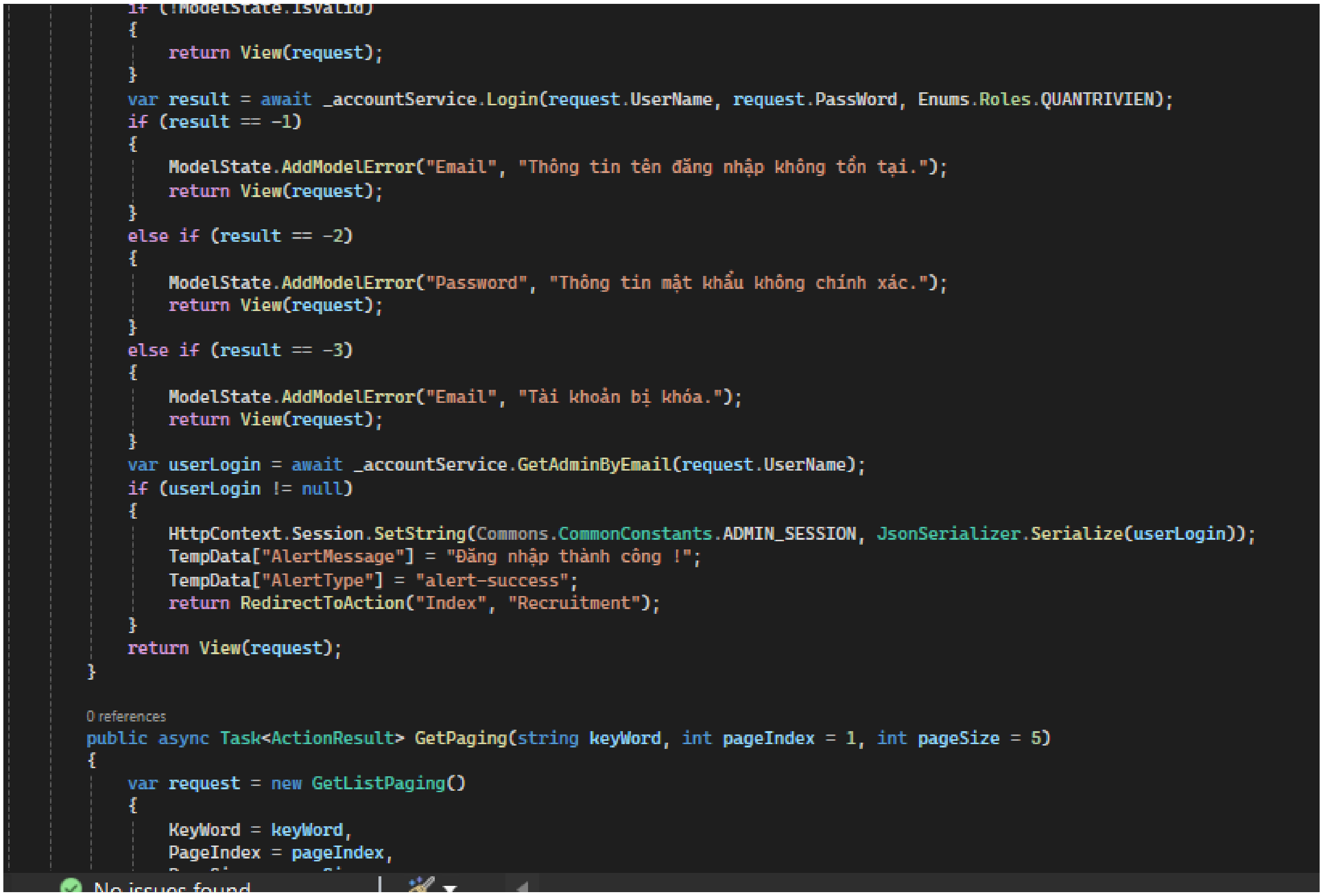
-PassWord: This property represents the user's password. Similar to UserName, Display is used to specify the display name for this field on the user interface. The Required attribute specifies that this field is required and an error message will be displayed if the user does not enter a value for this field.

-RememberMe: This attribute is typically used to allow users to select "Remember me" to maintain their login session.

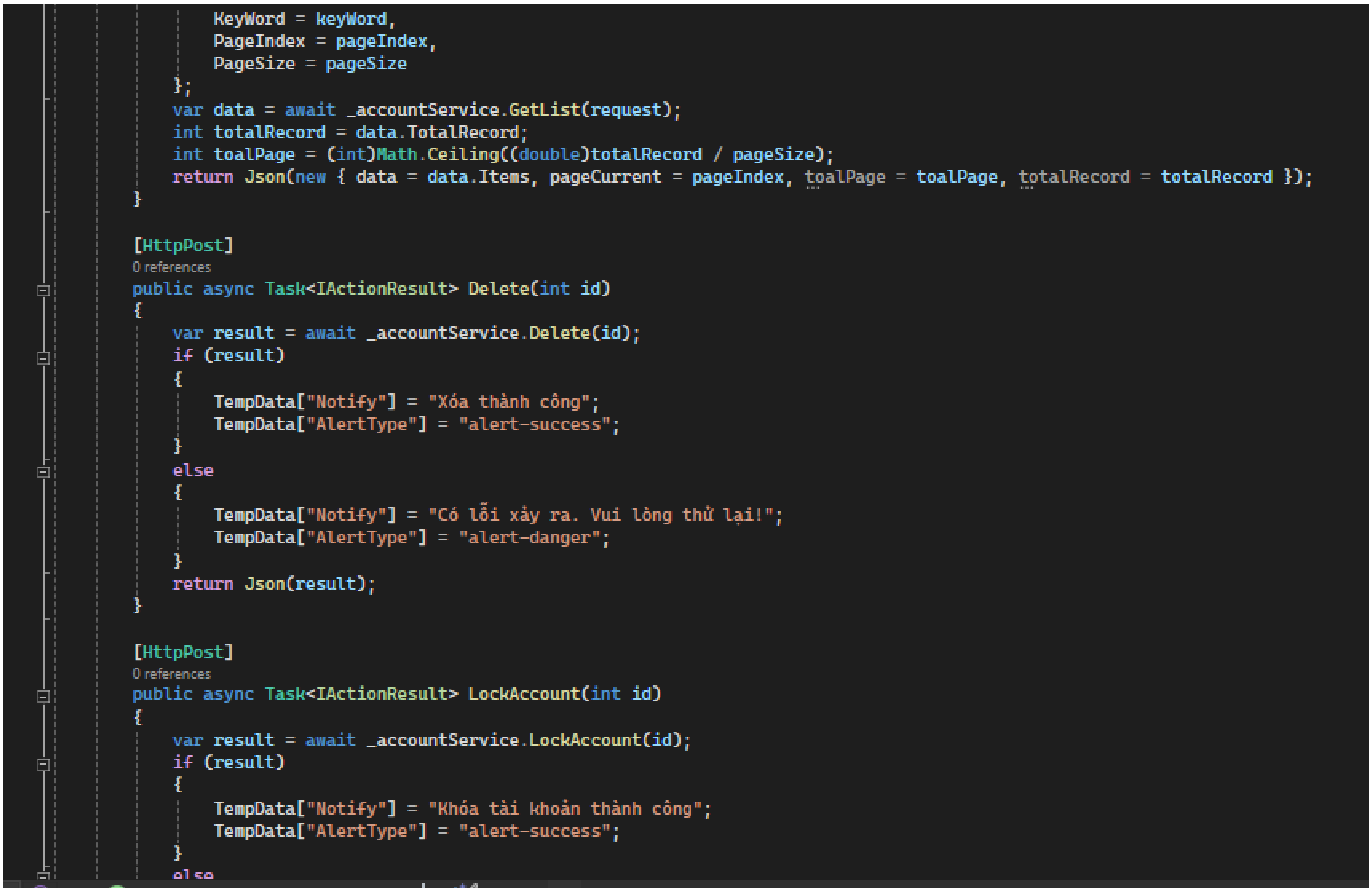
The code also uses properties of System.ComponentModel.DataAnnotations to define validation rules and display names for fields in the form. This helps control the data being entered and provides more detailed information to the user about errors if any.



```
{
    1 reference
    public class AccountController : Controller
    {
        private readonly AccountService _accountService;
        0 references
        public AccountController()
        {
            _accountService = new AccountService();
        }


        0 references
        public IActionResult Index()
        {
            return View();
        }

        0 references
        public IActionResult Login()
        {
            return View();
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> Login(LoginModel request)
        {
            if (!ModelState.IsValid)
            {
                return View(request);
            }
            var result = await _accountService.Login(request.UserName, request.PassWord, Enums.Roles.QUANTRIVIEN);
            if (result == -1)
            {
                ModelState.AddModelError("Email", "Thông tin tên đăng nhập không tồn tại.");
                return View(request);
            }
            else if (result == -2)
```



```
        if (!ModelState.IsValid)
        {
            return View(request);
        }
        var result = await _accountService.Login(request.UserName, request.PassWord, Enums.Roles.QUANTRIVIEN);
        if (result == -1)
        {
            ModelState.AddModelError("Email", "Thông tin tên đăng nhập không tồn tại.");
            return View(request);
        }
        else if (result == -2)
        {
            ModelState.AddModelError("Password", "Thông tin mật khẩu không chính xác.");
            return View(request);
        }
        else if (result == -3)
        {
            ModelState.AddModelError("Email", "Tài khoản bị khóa.");
            return View(request);
        }
        var userLogin = await _accountService.GetAdminByEmail(request.UserName);
        if (userLogin != null)
        {
            HttpContext.Session.SetString(Commons.CommonConstants.ADMIN_SESSION, JsonSerializer.Serialize(userLogin));
            TempData["AlertMessage"] = "Đăng nhập thành công !";
            TempData["AlertType"] = "alert-success";
            return RedirectToAction("Index", "Recruitment");
        }
        return View(request);
    }

    0 references
    public async Task<ActionResult> GetPaging(string keyWord, int pageIndex = 1, int pageSize = 5)
    {
        var request = new GetListPaging()
        {
            KeyWord = keyWord,
            PageIndex = pageIndex,
```



```
                KeyWord = keyWord,
                PageIndex = pageIndex,
                PageSize = pageSize
            };
            var data = await _accountService.GetList(request);
            int totalRecord = data.TotalRecord;
            int toalPage = (int)Math.Ceiling((double)totalRecord / pageSize);
            return Json(new { data = data.Items, pageCurrent = pageIndex, toalPage = toalPage, totalRecord = totalRecord });
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> Delete(int id)
        {
            var result = await _accountService.Delete(id);
            if (result)
            {
                TempData["Notify"] = "Xóa thành công";
                TempData["AlertType"] = "alert-success";
            }
            else
            {
                TempData["Notify"] = "Có lỗi xảy ra. Vui lòng thử lại!";
                TempData["AlertType"] = "alert-danger";
            }
            return Json(result);
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> LockAccount(int id)
        {
            var result = await _accountService.LockAccount(id);
            if (result)
            {
                TempData["Notify"] = "Khóa tài khoản thành công";
                TempData["AlertType"] = "alert-success";
            }
            else
```

AccountController is a controller in an ASP.NET Core application. Below is a description of the controller methods:

Constructor: In the constructor, AccountController initializes an object of AccountService to interact with account-related data.

Index Action: This method returns a view to display the main page of the admin site.

Login Action (GET): This method returns a view to display the login form.

Login Action (POST): This method is called when the user submits the login form. It checks the validity of the input data and then calls the Login method from AccountService. If the login is successful, it saves the user information into the session and redirects the user to the main admin page (Index action of RecruitmentController). Otherwise, it displays the corresponding error messages.

GetPaging Action: This method is used to get paging data from AccountService and return the data as JSON for use in AJAX requests.

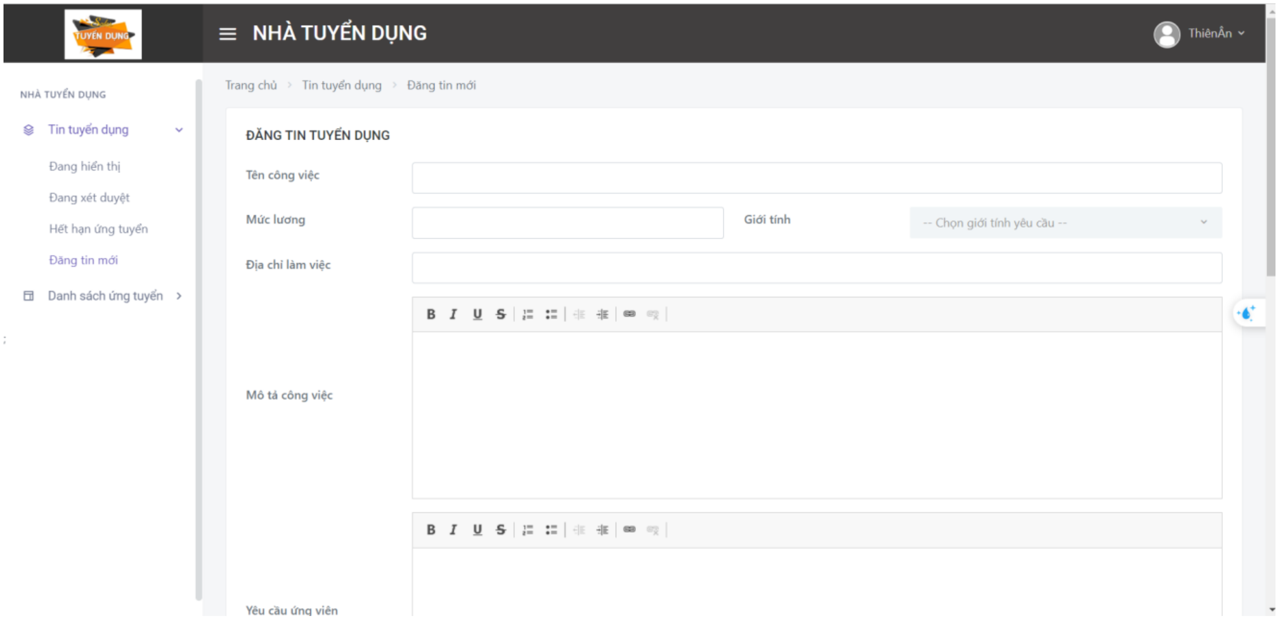
Delete Action: This method is used to delete an account. It calls the Delete method from AccountService and then sets up notifications based on the results.

LockAccount Action: This method is used to lock an account. It calls the LockAccount method from AccountService and then sets up notifications based on the results.

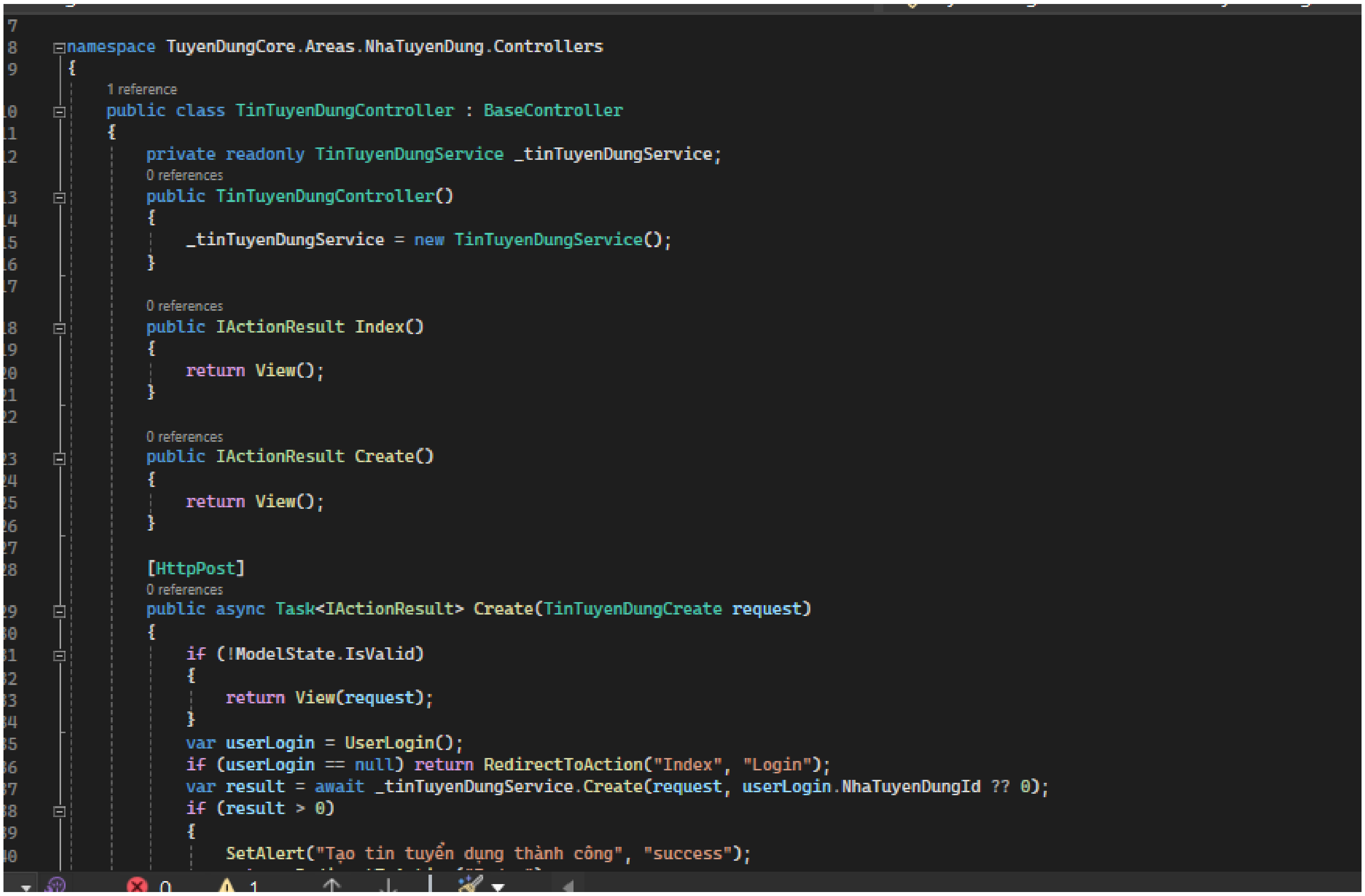
UnLockAccount Action: This method is used to unlock an account. It calls the UnLockAccount method from AccountService and then sets up notifications based on the results.

These methods handle requests from the user interface and interact with the service to perform account-related operations. They then return the results to the user interface via view or JSON.

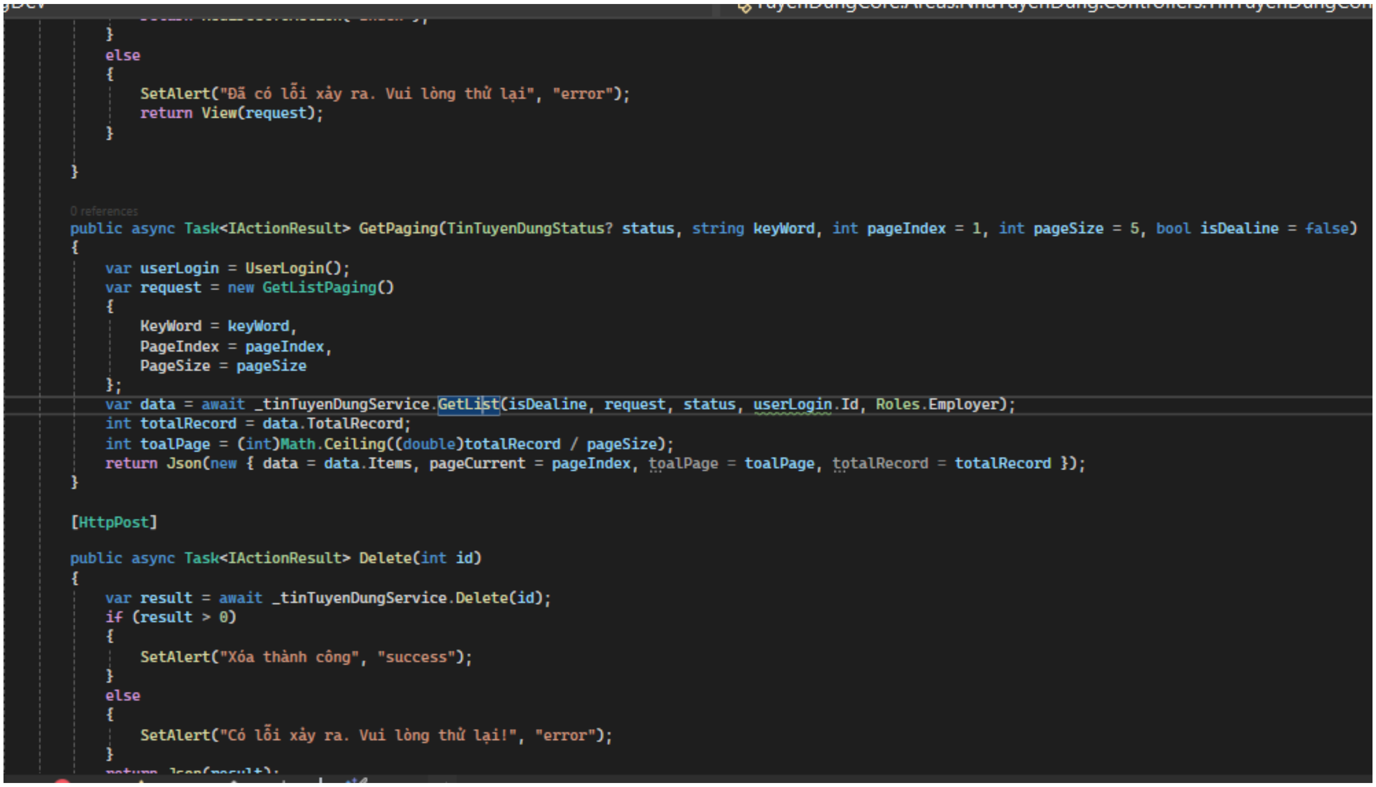
Employers shall have the ability to create and post job listings.



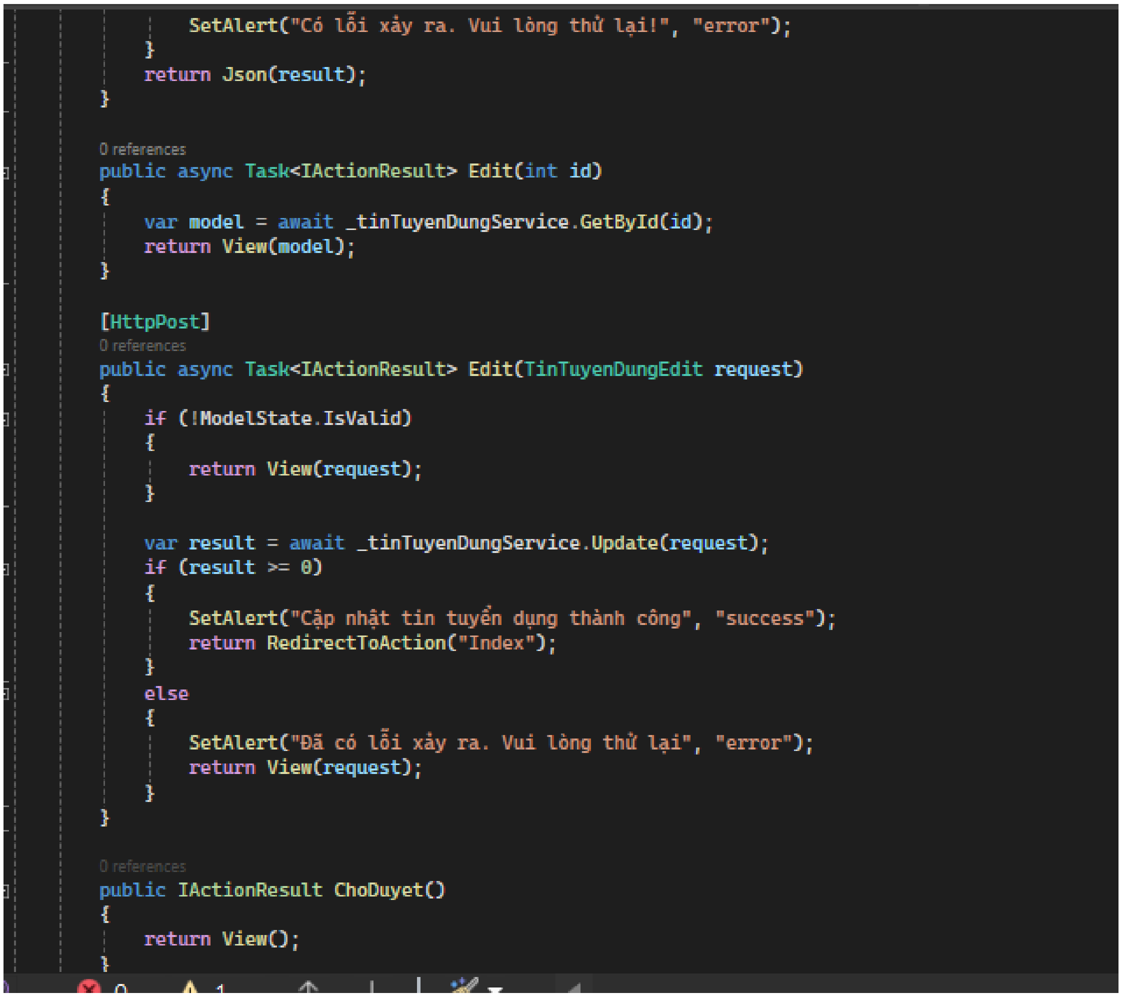
Code:



```
namespace TuyenDungCore.Areas.NhaTuyenDung.Controllers
{
    1 reference
    public class TinTuyenDungController : BaseController
    {
        private readonly TinTuyenDungService _tinTuyenDungService;
        0 references
        public TinTuyenDungController()
        {
            _tinTuyenDungService = new TinTuyenDungService();
        }

        0 references
        public IActionResult Index()
        {
            return View();
        }

        0 references
        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> Create(TinTuyenDungCreate request)
        {
            if (!ModelState.IsValid)
            {
                return View(request);
            }
            var userLogin = UserLogin();
            if (userLogin == null) return RedirectToAction("Index", "Login");
            var result = await _tinTuyenDungService.Create(request, userLogin.NhaTuyenDungId ?? 0);
            if (result > 0)
            {
                SetAlert("Tạo tin tuyển dụng thành công", "success");
```



```
            return RedirectToAction("Index");
        }
        else
        {
            SetAlert("Đã có lỗi xảy ra. Vui lòng thử lại", "error");
            return View(request);
        }

    }

    0 references
    public async Task<IActionResult> GetPaging(TinTuyenDungStatus? status, string keyWord, int pageIndex = 1, int pageSize = 5, bool isDealine = false)
    {
        var userLogin = UserLogin();
        var request = new GetListPaging()
        {
            KeyWord = keyWord,
            PageIndex = pageIndex,
            PageSize = pageSize
        };
        var data = await _tinTuyenDungService.GetList(isDealine, request, status, userLogin.Id, Roles.Employer);
        int totalRecord = data.TotalRecord;
        int toalPage = (int)Math.Ceiling((double)totalRecord / pageSize);
        return Json(new { data = data.Items, pageCurrent = pageIndex, toalPage = toalPage, totalRecord = totalRecord });
    }

    [HttpPost]

    public async Task<IActionResult> Delete(int id)
    {
        var result = await _tinTuyenDungService.Delete(id);
        if (result > 0)
        {
            SetAlert("Xóa thành công", "success");
        }
        else
        {
            SetAlert("Có lỗi xảy ra. Vui lòng thử lại!", "error");
        }
        return Json(result);
```



```
            SetAlert("Có lỗi xảy ra. Vui lòng thử lại!", "error");
        }
        return Json(result);
    }

    0 references
    public async Task<IActionResult> Edit(int id)
    {
        var model = await _tinTuyenDungService.GetById(id);
        return View(model);
    }

    [HttpPost]
    0 references
    public async Task<IActionResult> Edit(TinTuyenDungEdit request)
    {
        if (!ModelState.IsValid)
        {
            return View(request);
        }

        var result = await _tinTuyenDungService.Update(request);
        if (result >= 0)
        {
            SetAlert("Cập nhật tin tuyển dụng thành công", "success");
            return RedirectToAction("Index");
        }
        else
        {
            SetAlert("Đã có lỗi xảy ra. Vui lòng thử lại", "error");
            return View(request);
        }
    }

    0 references
    public IActionResult ChoDuyet()
    {
        return View();
    }
```

In this code, TinTuyenDungController is a controller in the ASP.NET Core application, located in the NhaTuyenDung area. Below is a description of the controller methods:

-Constructor: In the constructor, TinTuyenDungController initializes an object of TinTuyenDungService to interact with data related to job postings.

-Index Action: This method returns a view to display the list of job postings.

-Create Action (GET): This method returns a view to display the new job posting creation form.

-Create Action (POST): This method is called when the user submits a form to create a new job posting. It checks the validity of the input data and then calls the Create method from TinTuyenDungService to create a new job posting.

-GetPaging Action: This method is used to get paging data from TinTuyenDungService and returns the data as JSON for use in AJAX requests.

-Delete Action: This method is used to delete a job posting. It calls the Delete method from TinTuyenDungService and then sets up notifications based on the results.

-Edit Action (GET): This method returns a view to display the job posting editing form.

-Edit Action (POST): This method is called when the user submits the job posting editing form. It checks the validity of the input data and then calls the Update method from TinTuyenDungService to update the job posting.

-ChoDuyet Action: This method returns a view to display the list of job postings waiting for approval.

-HetHan Action: This method returns a view to display a list of expired job postings.

These methods handle requests from the user interface and interact with the service to perform operations related to the job posting. They then return the results to the user interface via view or JSON.

1. Application Evaluation

TODO:

- *Review the performance of the application*
- *Conclude whether the application adapts all requirements or it needs to be improved later*
- *Analyse the factors that influence the performance of the application*
- *Evaluate the strengths and weaknesses of the application*

1. Review the performance of the application

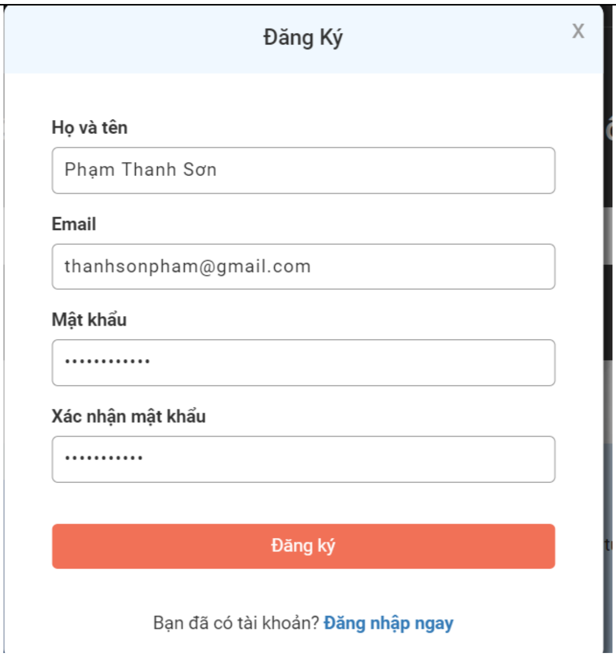
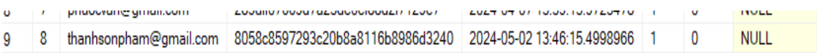
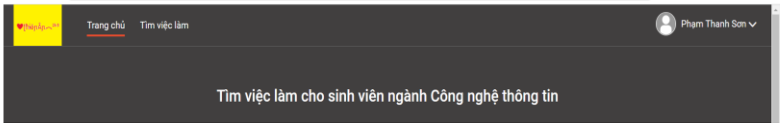
Test case

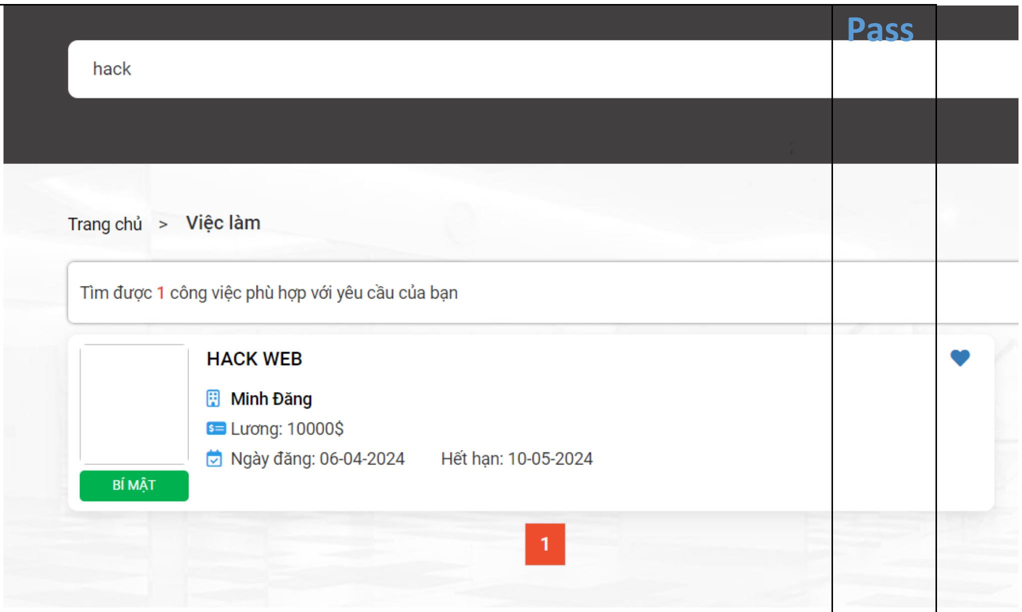
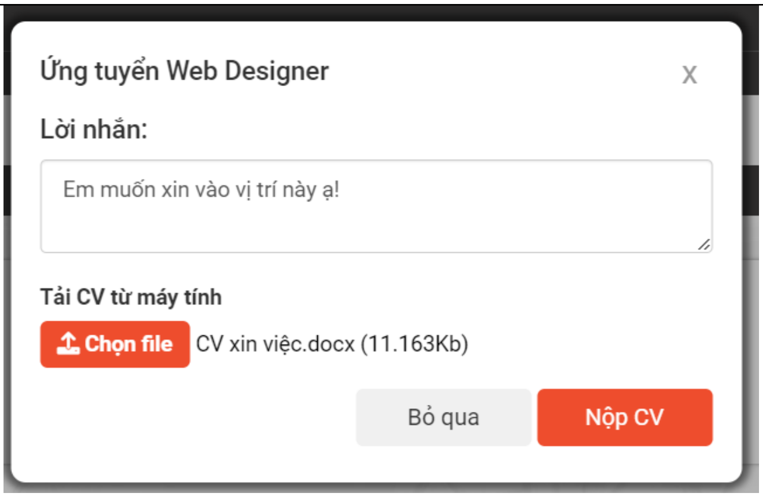
No	Name	Description
1	Register an account	After users input their information into the corresponding fields and click on the register button, the system will add a new row to the table user in the database.
2	Login by an account	After users input their information into the corresponding fields and click on the login button, they can access the system.
3	Search Jobs	After selecting a category and clicking the filter button, the user will see all the jobs whose category matches the job seeker's desired category.
4	Upload CV	After choosing the job that job seekers want, they can upload their CV file to the employer
5	Edit Profile	Users can customize their profile.
6	Change Password	Users can change their password.
7	Post news	Employers can post job advertisements.
8	Edit and Delete Post	Employers can edit and delete

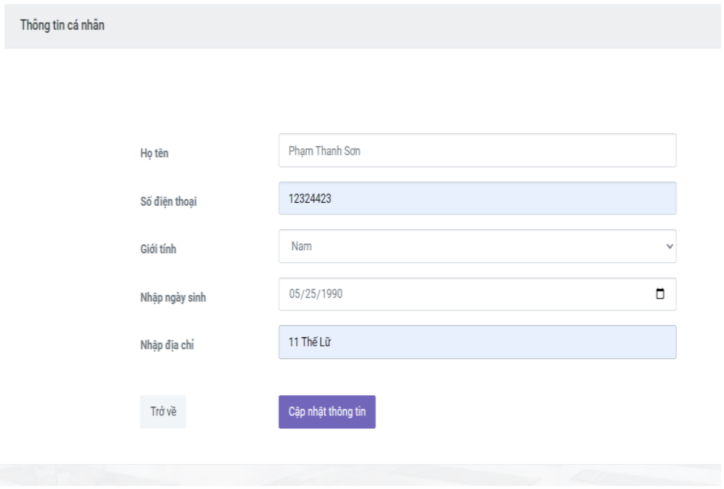
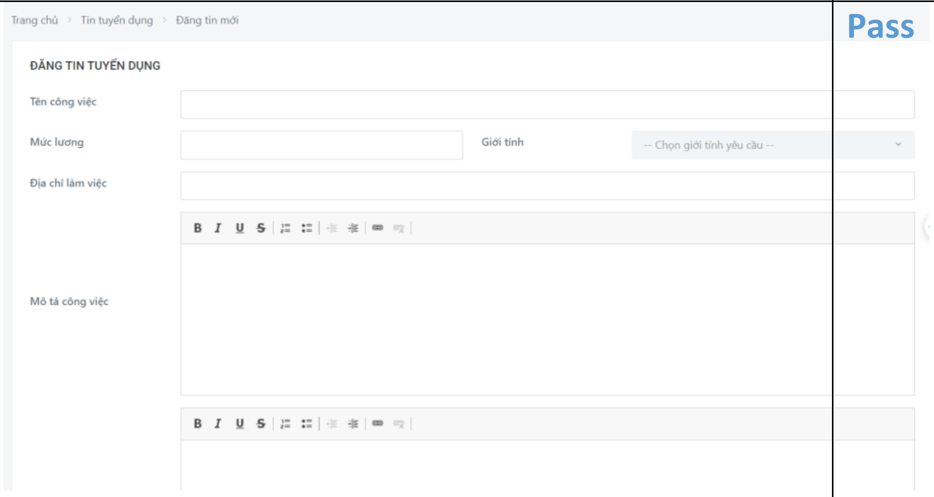
		the posts they post.
9	Browse application	Employers can approve job seekers' job requests
10	View information on accounts	Admin can view information about all accounts in the system.
11	Change the password of an account	Admin can change the password for any account by inputting the new password and clicking on the save button
12	Create candidates and recruiters	Admin can create candidates and employers
13	Manage category lists	Admin can manage the Recruitment category list
14	Browse the Recruitment category list	Admin can browse the list of Recruitment category

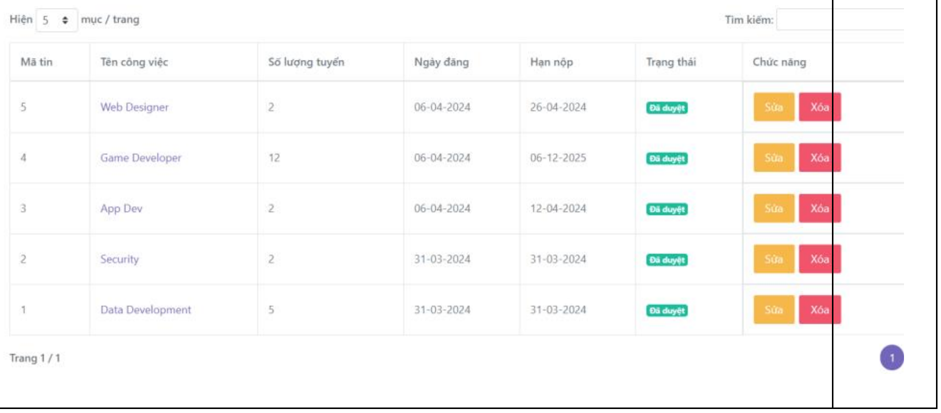
Test result

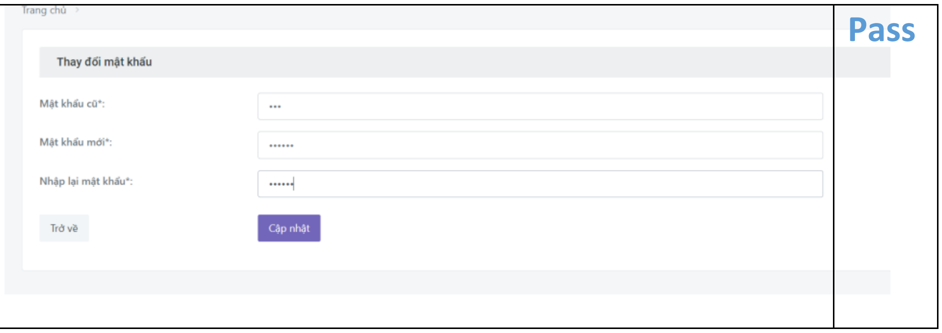
Test case	Name	Expected result	Actual result	Status

1	Register an account	After users click on the register button, a new record of table users is added	 Đăng Ký X Họ và tên Phạm Thanh Sơn Email thanhsonpham@gmail.com Mật khẩu .... Xác nhận mật khẩu .... Đăng ký Bạn đã có tài khoản? Đăng nhập ngay  9 8 thanhsonpham@gmail.com 8058c8597293c20b8a8116b8986d3240 2024-05-02 13:46:15.4998966 1 0 NULL	Pass
2	Login into the system	After clicking on the login button, users will see their name on the web page	 Tìm việc làm cho sinh viên ngành Công nghệ thông tin	Pass

3			 hack Trang chủ > Việc làm Tìm được 1 công việc phù hợp với yêu cầu của bạn HACK WEB Minh Đăng Lương: 10000$ Ngày đăng: 06-04-2024 Hết hạn: 10-05-2024 BÍ MẬT 1	Pass
4	Upload CV	Upload cv to apply for a job.	 Ứng tuyển Web Designer X Lời nhắn: Em muốn xin vào vị trí này ạ! Tải CV từ máy tính Chọn file CV xin việc.docx (11.163Kb) Bỏ qua Nộp CV	Pass

<table>
<tr><td>5</td><td>Update Profile</td><td>Update your Profile</td><td>
Thông tin cá nhân
Họ tên: Phạm Thanh Sơn
Số điện thoại: 12324423
Giới tính: Nam
Nhập ngày sinh: 05/25/1990
Nhập địa chỉ: 11 Thế Lữ
 </td><td>Pass</td></tr>
<tr><td>6</td><td>Create news</td><td>Create news</td><td>

ĐĂNG TIN TUYỂN DỤNG
Tên công việc:
Mức lương:
Giới tính: -- Chọn giới tính yêu cầu --
Địa chỉ làm việc:
Mô tả công việc:</td><td>Pass</td></tr>
<tr><td></td><td></td><td></td><td>
Hiện 5 mục / trang
Tìm kiếm:
<table>
<tr><th>Mã tin</th><th>Tên công việc</th><th>Số lượng tuyển</th><th>Ngày đăng</th><th>Hạn nộp</th><th>Trạng thái</th><th>Chức năng</th></tr>
<tr><td>5</td><td>Web Designer</td><td>2</td><td>06-04-2024</td><td>26-04-2024</td><td>Đã duyệt</td><td>Sửa Xóa</td></tr>
<tr><td>4</td><td>Game Developer</td><td>12</td><td>06-04-2024</td><td>06-12-2025</td><td>Đã duyệt</td><td>Sửa Xóa</td></tr>
<tr><td>3</td><td>App Dev</td><td>2</td><td>06-04-2024</td><td>12-04-2024</td><td>Đã duyệt</td><td>Sửa Xóa</td></tr>
<tr><td>2</td><td>Security</td><td>2</td><td>31-03-2024</td><td>31-03-2024</td><td>Đã duyệt</td><td>Sửa Xóa</td></tr>
<tr><td>1</td><td>Data Development</td><td>5</td><td>31-03-2024</td><td>31-03-2024</td><td>Đã duyệt</td><td>Sửa Xóa</td></tr>
</table>
Trang 1 / 1
 </td><td></td></tr>
</table>

7	Reset Password	Reset passwword	 Trang chủ > Thay đổi mật khẩu Mật khẩu cũ*: ••• Mật khẩu mới*: •••••• Nhập lại mật khẩu*: •••••• 	Pass

Conclude whether the application adapts all requirements, or it needs to be improved later.

Based on an assessment of the specific requirements around web pages, entity models, controllers, application features, and HTML/CSS usage, it can be concluded that the application meets all requirements. mandatory requirements and adjust most of the desired requirements.

This app has 18 web pages (views) that cover all the streams and features needed by the users. It has 7 independent entity models including Job, Category, Job Seeker, Employer, User, Role and User Role to store data efficiently. There are 9 controllers deployed to handle requests and responses for all roles. Although the app applies HTML5 and CSS3 as intended to structure content and styling, the CSS3 stylesheets can be further optimized. Overall, the application achieves all the required requirements around web pages, entity models, and controllers as expected. But CSS stylesheets require further improvements to fully accommodate the specifications. With these upgrades, the application can reach its maximum potential.

Additionally, Bootstrap is used to speed up the process of building interfaces for applications and to ensure that a website's design is consistent and responsive across devices and screen sizes of different sizes.

Regarding the functions that customers require, our system meets all of them:

- Yes Home Screen will be the default screen displayed when handling user login information and user logout options.

- The Job List screen displays all jobs that the customer can select.

➢ There is a Job type detail screen that helps display detailed information about a specific job

book.

- There is a Profile Screen which will be used to display his/her profile.

- There is a Registration screen that will be used to register customers.

➢ Has a Jobs Screen used to add, update, search or delete jobs.

Each book must have a specific Category.

➢ There is a Category Request Screen that will be used to make a request to the administrator to add a new work category if it does not exist.

- There is a recording screen that will be used to display all the recordings requested by the customer.

- There is a job seeker screen and a Recruiter screen which will be used to display accounts

information of all Customers. It will be possible to reset the Customer's password if necessary.

- Has a Catalog Approval Screen that allows administrators to approve or reject new jobs

Category requests are made by the Employer.

Overall, our application meets all the requirements of Job Seekers, but they are not enough to optimize the recruitment process. So if we have a lot of time and

opportunity, we will definitely release a new update with more functions that will make the owner's process management more convenient and easier.

Analyze the factors that influence the performance of the application

There are many factors that can affect the performance of an ecommerce website, including the quality of the code used to build it. Overall, below are some essential factors that need to be checked and researched to make the application have better performance:

➢ Website design: The design of the website can have a significant impact on its performance. A well-designed, easy-to-navigate, and visually appealing website can motivate users

spend more time on more websites.

- Website speed: The speed of a website is a determining factor in its success. Slow load times can lead to frustration and cause users to leave the site. Optimizing images, using caching techniques, and minimizing the use of external resources can help improve site speed.

- Server performance: The performance of the server hosting the website is also important. Slow or overloaded servers can cause the website to crash or become unresponsive, leading to poor user experience. experience.

➢ Mobile Optimization: As more and more users access e-commerce websites on mobile devices. necessary to ensure that the website is optimized for mobile use. This includes using technically responsive design and optimizing images and other resources for smaller screens.

➢ Code quality: The quality of the code used to build a website can have a significant impact on its performance. Well-written, optimized code can improve site speed and reduce risk

crashes or other technical issues.

- User Experience: Overall user experience is crucial to the success of an e-commerce website.

This includes factors such as website design, ease of use, product and customer information.

service. Specifically, to ensure that visitors enjoy their experience, make sure the website has a clean and attractive look. Ensure that each page is organized in a logical and user-friendly manner

Don't get lost while navigating through the website.

Evaluate the strengths and weaknesses of the application

Strength:

One of the biggest strengths of this application is its user-friendly interface. The interface is designed in such a way that it is easy to navigate and users can quickly find what they are looking for. In addition, the App is equipped with various functions such as add, edit, delete, search and help users easily manage their work.

Another strength of this app is that it meets the needs of all three types of users, i.e. job seekers, recruiters, and administrators. Each type of user has its own set of features, and the application fully meets the requirements of each type of user. For example, customers can browse job categories, apply for jobs and upload their cvs, while recruiters can manage their job postings and administrators can manage activity general of the application.

Weaknesses:

One weakness of this application is that it still has some response errors. This means that the website may not display correctly on all devices and users may have difficulty accessing it. This can be a major problem as it can lead to reduced user engagement and lead to loss of potential customers.

Another weakness of this application is that it may not be able to handle large numbers of visits. This means that if a large number of users try to access the website at the same time, it may crash or become slow, leading to a poor user experience. This can be a significant drawback.

Finally, the app is not completely safe in terms of security. This can be a significant concern as users may not feel comfortable sharing their personal and financial information on an uncertain website. This can lead to loss of customer trust, which can have a long-term negative impact on the success of the application.

Index of comments

1.1 1. What is good

- You can develop a functional business application based on a specified business problem.(P5)

- You can review the performance of your business application against the Problem Definition Statement and initial requirements. (P6)

2. What is not good

You should correct the questionaires