

Higher Nationals – Assignment Front Sheet

Student Name/ID	Ngo Tri Tai/ GCD210330						
Unit Title	Unit 30: Application Development						
Assignment Number	Assignment 2	Assessor	Pham Thanh Son				
Submission Date	23/8/2024	Date Received 1st submission					
Re-submission Date	30/8/2024	Date Received 2nd submission					
Grading grid							
P4	P5	P6	M3	M4	M5	D2	D3
Assessor Feedback:							
<p>*Please note that constructive and useful feedback should allow students to understand:</p> <ul style="list-style-type: none"> a) Strengths of performance b) Limitations of performance c) Any improvements needed in future assessments <p>Feedback should be against the learning outcomes and assessment criteria to help students understand how these inform the process of judging the overall grade.</p> <p>Feedback should give full guidance to the students on how they have met the learning outcomes and assessment criteria.</p>							
Grade:	Assessor Signature:				Date:		
Resubmission Feedback:							
<p>*Please note resubmission feedback is focussed only on the resubmitted work</p>							
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.

When submitting evidence for assessment, each student must sign a declaration confirming that the work is their own.

Student name: Ngo Tri Tai		Assessor name: Pham Thanh Son	
Issue date:	Submission date: 30/8/2024	Submitted on:	
Programme: BTEC			
Unit 30: Application Development			
Assignment number and title: Assignment 2			

Plagiarism

Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalised. It is your responsibility to ensure that you understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

Student Declaration

Student declaration

I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I declare that the work submitted for assessment has been carried out without assistance other than that

which is acceptable according to the rules of the specification. I certify I have clearly referenced any sources and any artificial intelligence (AI) tools used in the work. I understand that making a false declaration is a form of malpractice.

Student signature:



Date: 23/8/2024

A. Develop a formal questionnaire designed to thoroughly evaluate your business application, problem definition statement, proposed solution, and development strategy. Utilize this questionnaire for a peer review, and make sure to document any feedback received.

We plan to collect feedback from both job seekers and employers after showcasing and explaining the features of the FPT Recruitment platform. By reviewing your recruitment application along with the provided details on problem identification, potential solutions, and strategy development, we've gathered some initial thoughts and comments. To further refine the platform, we've crafted the following questions aimed at gathering user feedback and suggestions for future improvements in the recruitment process. Your input will be crucial in enhancing the platform to better serve the needs of both job seekers and employers.

How would you rate your first reaction to our application? *

1 2 3 4 5

Very Bad ☐ ☐ ☐ ☐ ☐ Very Good

What do you like most about this application? *

☐ Stability

☐ Navigation

☐ Speed

☐ Functionality

What is your primary reason for using our application? *

☐ Looking for job opportunities

☐ Posting job vacancies

☐ Networking with professionals

☐ Exploring career paths

Figure 1: Questionnaire about business application 1

Would you recommend our job recruitment platform to others? *

☐ Definitely yes

☐ Not sure

☐ Definitely not

How would you rate the overall quality of the application? *

1 2 3 4 5

Very Low Quality ☐ ☐ ☐ ☐ ☐ Very High Quality

Thank you for taking the survey.




Figure 2: Questionnaire about business application 2

*** Collecting review feedback:**

After gathering feedback from 100 individuals, we've compiled statistics on how clients feel after using our system. The initial data focuses on consumers' first impressions and highlights what they like most about our platform.

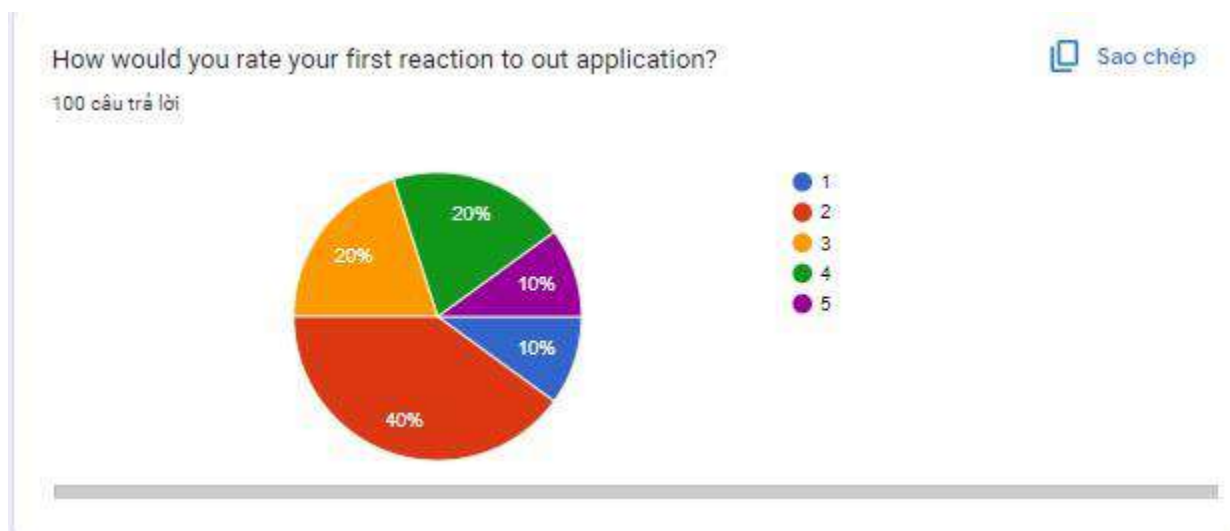


Figure 3: Result question 1

The most common rating assigned by respondents was , with 40 people selecting it. This accounts for 40% of the total responses, showing a clear preference for this rating among users. On the other hand, the ratings at the extremes—1 and 5—were the least chosen, with only 10 participants selecting each, representing 10% of the feedback. This suggests that opinions about the app are not highly polarized, with fewer users feeling either extremely positive or negative.

Overall, the majority of feedback was positive, with half of all respondents rating the app between 3 and 5. This reflects general satisfaction with the app, although there is still room for improvement to increase the number of higher ratings.

What do you like most about this application?

100 câu trả lời

Sao chép

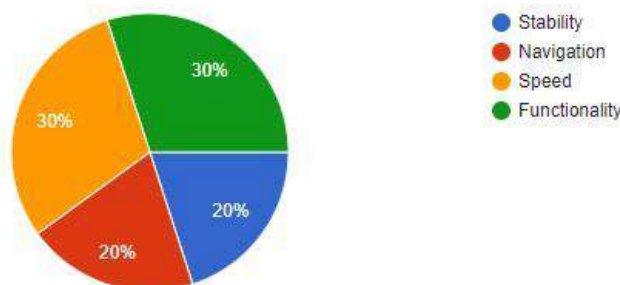


Figure 4: Result question 2

Speed and functionality were the most frequently mentioned attributes in user feedback, each receiving 30% of the total responses. This significant level of feedback highlights a strong appreciation for these aspects, emphasizing their crucial role in user satisfaction.

While stability and navigation were also positively received, they were mentioned less frequently, with each garnering 20% of the responses. This suggests that users consider these features moderately important.

The equal emphasis on speed and functionality suggests that users highly value a consistent and intuitive experience, indicating that these factors are the key drivers of satisfaction.

What is your primary reason for using our application?

100 câu trả lời

Sao chép

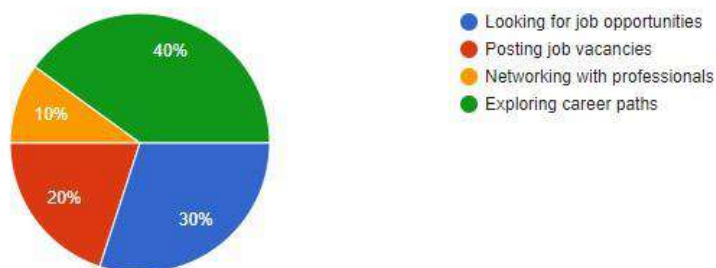


Figure 5: Result question 3

The feature centered on exploring career paths was the most popular among users, capturing 40% of the total responses. This notable share indicates a strong interest in career exploration as a key function of the platform.

Networking with professionals was the least favored feature, chosen by only 10% of users. This lower preference suggests that networking is not a primary focus for most users on the platform.

Overall, the majority of responses show a clear preference for features related to career exploration and job opportunities, making up 70% of the feedback. This trend highlights the platform's main value as a tool for career development and job search. While the platform offers various features, its primary strength and user engagement appear to be in helping individuals navigate their career paths and find job opportunities.

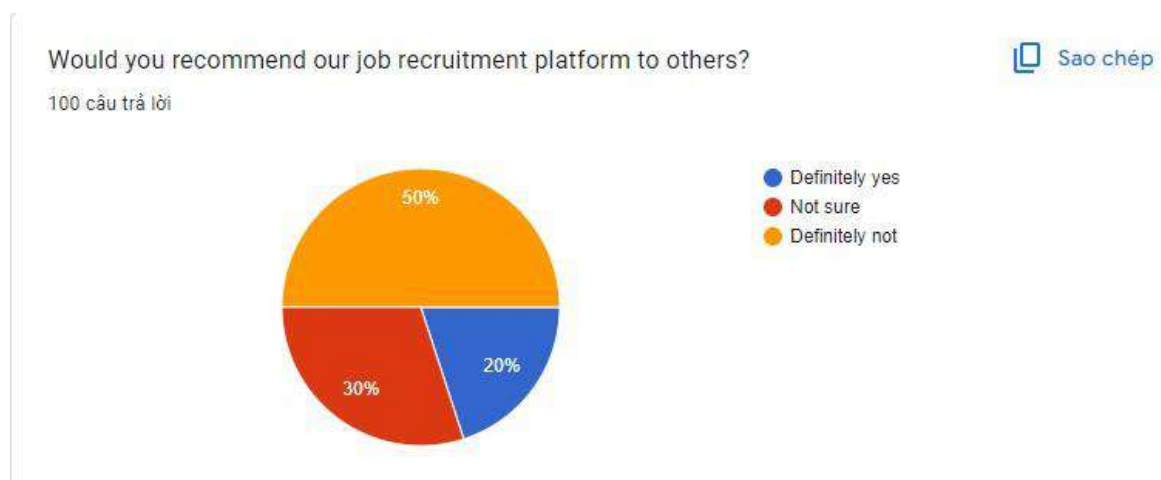


Figure 6: Result question 4

According to the feedback collected, half of the respondents, accounting for 50% of the total, stated they would "Definitely not" recommend the job recruitment platform, indicating a high level of dissatisfaction among users.

On the other hand, only 20% of participants expressed that they would "Definitely yes" recommend the platform. Additionally, 30% were "Not sure" about recommending it, bringing these two groups to a combined total of 50%. These statistics reveal a divide in user satisfaction, with a minority showing approval while a larger proportion remains uncertain or dissatisfied with the platform.

The feedback indicates a divided user base, with a substantial portion clearly dissatisfied, while others are either uncertain or satisfied. This divide underscores the opportunity for platform improvements to boost user satisfaction and increase the likelihood of recommendations.

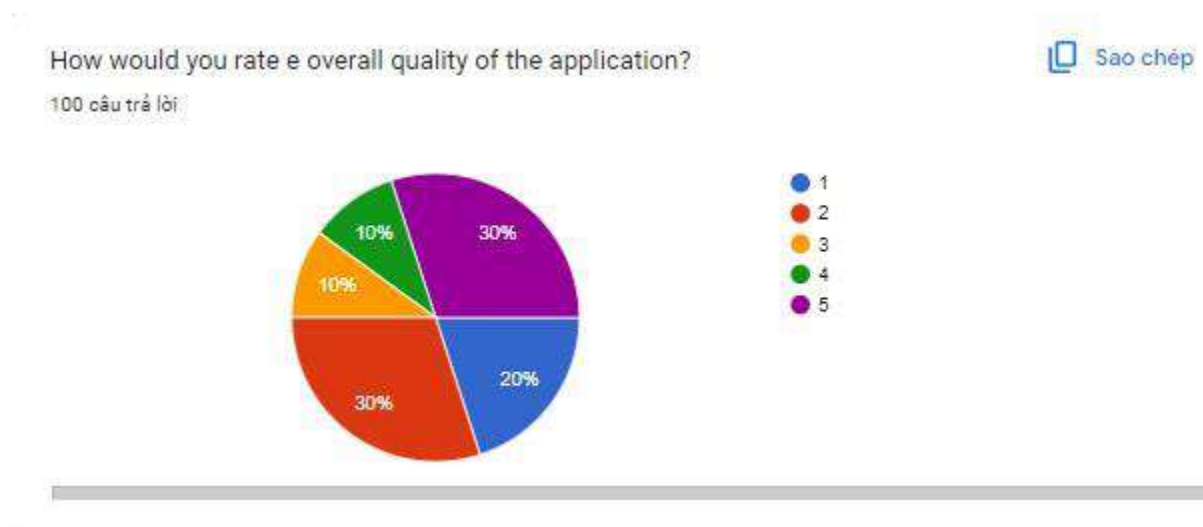


Figure 7: Result question 5

The most frequent ratings were 2 and 5, each chosen by 30% of respondents. This points to a clear divide in user perceptions, with these ratings reflecting significant but contrasting views.

Ratings of 3 and 4 were less common, each making up just 10% of the responses. This suggests that fewer users felt neutral about the application, indicating a polarization in user opinions.

The feedback on the application reveals polarized opinions, with equal numbers of users rating it at both the high and low ends of the quality spectrum. This polarization points to mixed user experiences, highlighting the need for a thorough investigation into the reasons behind these varied perceptions to identify areas for improvement.

B. Develop a functional business application based on a specified business problem:

I. Folder structure of the application:

Overall structure of the project: This folder contains the whole code source for the FPTJobMatch project.

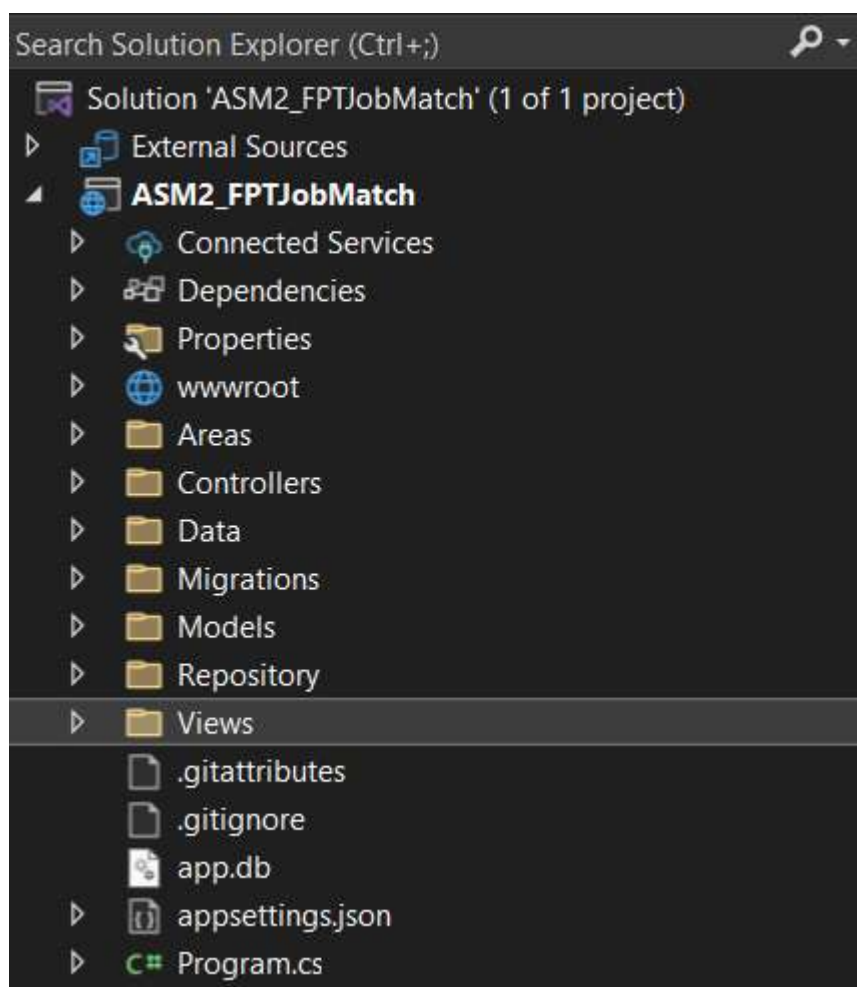


Figure 8: Folder Structure 1

This folder, located within the Area folder, contains files used for managing user login, registration, and handling user information.

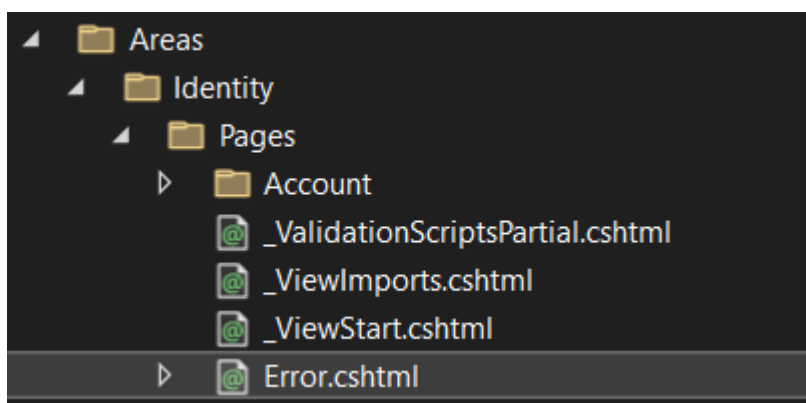


Figure 9: Folder Structure 2

This folder, known as the Controllers folder, contains various controller files such as ApplicationController, AppRolesController, CategoryController, EmployerController, HomeController, JobController, and JobSeekerController. These files are responsible for handling the logic behind event tasks and managing navigation between view pages. Each file is associated with specific user-defined roles.

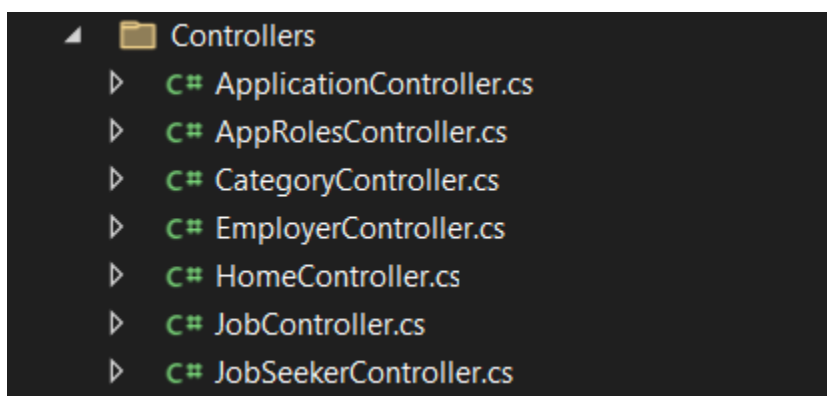


Figure 10: Folder Structure 3

Data Folder: The ApplicationDbContext.cs file in this folder is likely responsible for managing the application's database context, handling operations such as creating, reading, updating, and deleting data within the database.

Migrations Folder: This folder contains files associated with database migrations, which allow for changes to the database schema over time, such as adding or removing tables or columns. The 20240505091948_UpdateProject.cs file is a migration file that represents a specific modification to the database. The ApplicationDbContextModelSnapshot.cs file reflects the state of the database schema after all migrations have been applied.

Models Folder: This folder contains files that define the data models utilized by the application. These models represent the data that the application interacts with. For instance, the ApplicationUser.cs file likely outlines the properties of a user, such as their username, password, and email address. Similarly, the CategoryModel.cs, ErrorViewModel.cs, and JobModel.cs files define other types of data used by the application.

The organization of these files and folders indicates that the project likely follows a design pattern like Model-View-Controller (MVC) or a similar approach. In these patterns, the application is divided into distinct components based on their roles: Models manage the data and business logic, Views manage the user interface, and Controllers handle the interaction between the Models and Views.

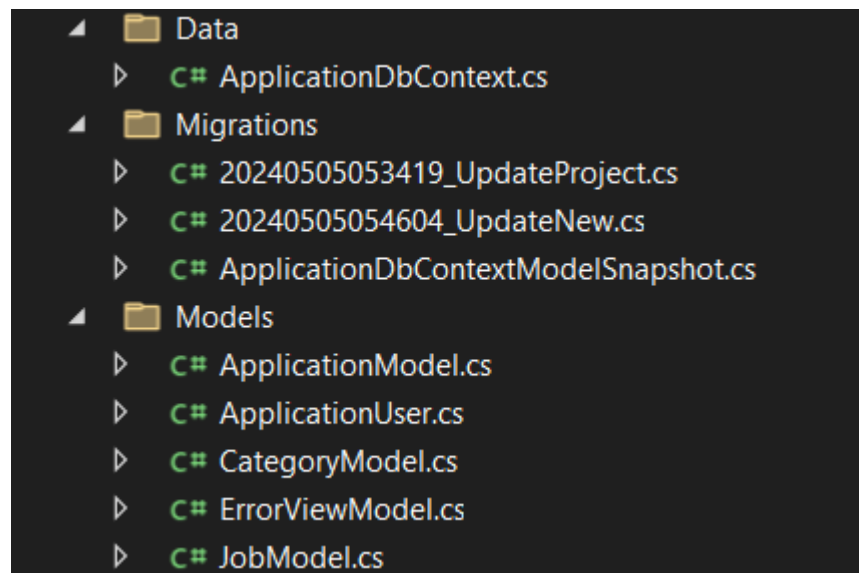


Figure 11: Folder Structure 4

A screenshot from an Integrated Development Environment (IDE) displays a file explorer showing a list of C# code files organized into various folders. These files belong to a software project. Here's a summary:

Repository Folder: This is the primary folder containing all the files and folders related to the repository. In programming, a repository serves as a storage location for your project, encompassing various files and folders.

Subfolders: There are two subfolders also named “Repository,” which might be used to further organize the repository files according to their functionality or the module they belong to.

C# Files: The five visible C# files are likely associated with data handling and user management within the application. Here's a breakdown:

- *ApplicationUserRepository.cs:* This file probably includes methods for database operations related to the application's users.
- *IRepository.cs:* This is likely an interface defining the basic CRUD operations (Create, Read, Update, Delete), which other repository classes implement.
- *UnitOfWork.cs:* This file is likely used to combine multiple repository operations into a single transaction, ensuring that all operations succeed or none do.

This structure indicates a well-organized project that likely adheres to the Repository and Unit of Work patterns. These patterns are commonly employed in applications to separate data access concerns and enhance code reusability and maintainability.

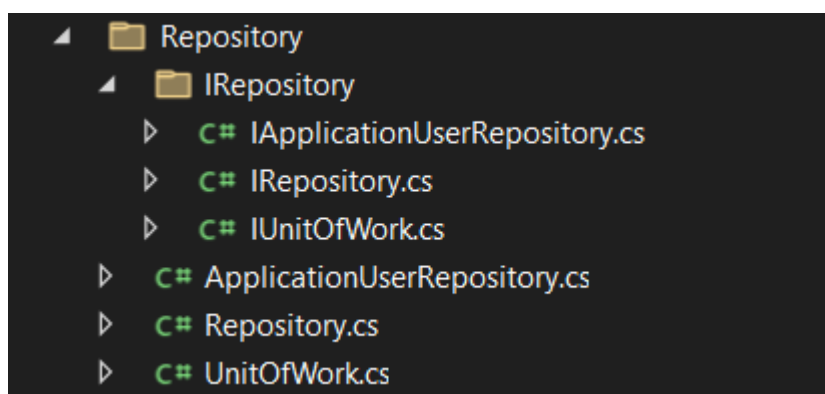


Figure 12: Folder Structure 5

A screenshot of a web development project structure, likely within an ASP.NET MVC application or a similar framework, displays two folders named ‘Application’ and ‘AppRoles’. Each folder contains .cshtml files corresponding to various web pages or views. The files are named according to CRUD

operations (Create, Read/Details, Update/Edit, Delete), indicating their use in managing data related to applications and application roles. This arrangement is typical in web applications, where separate views handle different aspects of data management, facilitating organized and efficient user interaction with the application's data.

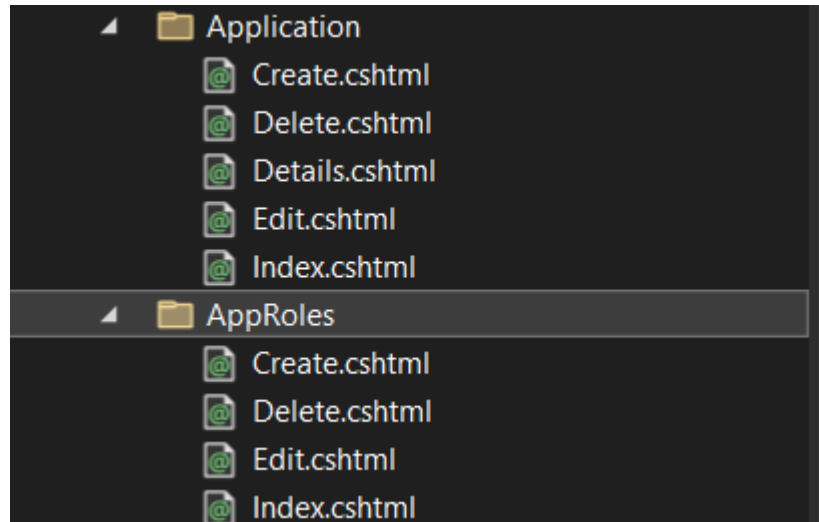


Figure 13: Folder Structure 6

The files, named Create.cshtml, Delete.cshtml, Details.cshtml, Edit.cshtml, Index.cshtml, and Privacy.cshtml, suggest they are views for an ASP.NET MVC web application. These views are likely used for creating, deleting, editing, and displaying categories and home page content. The inclusion of Privacy.cshtml indicates a specific view for privacy policy information. This organization is typical in MVC architecture, where views are arranged according to their function within the application.

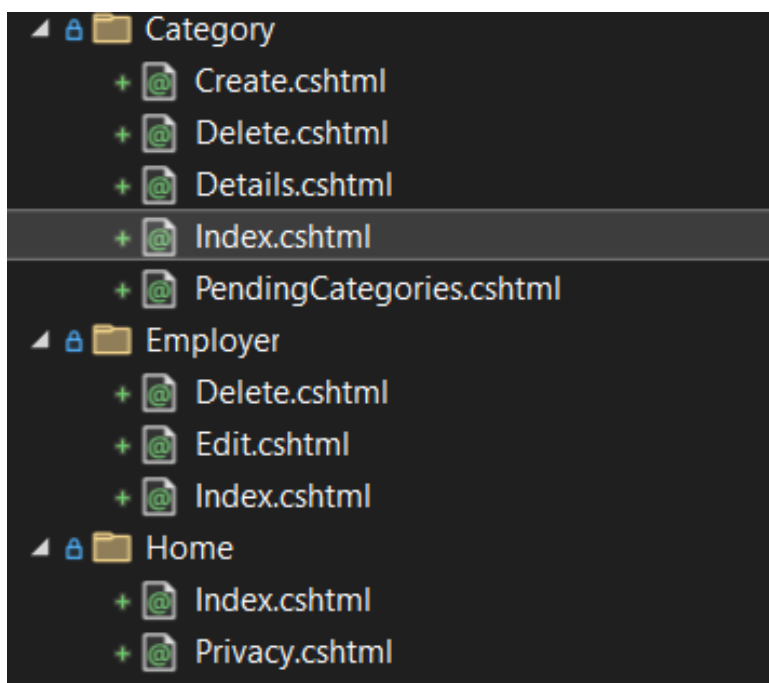


Figure 14: Folder Structure 7

A project structure within an Integrated Development Environment (IDE) or code editor displays the organization of views in an ASP.NET MVC application. The files are organized into “Job” and “Shared” folders. The “Job” folder contains views such as Create.cshtml, Delete.cshtml, Details.cshtml, Edit.cshtml, Index.cshtml, and JobSeeker.cshtml. The “Shared” folder includes _Layout.cshtml, _LoginPartial.cshtml, _ValidationScriptsPartial.cshtml, and Error.cshtml, while _ViewStart.cshtml is located outside these folders. This setup reflects a web development project where .cshtml files are Razor view pages, each serving a distinct role in the application’s user interface.

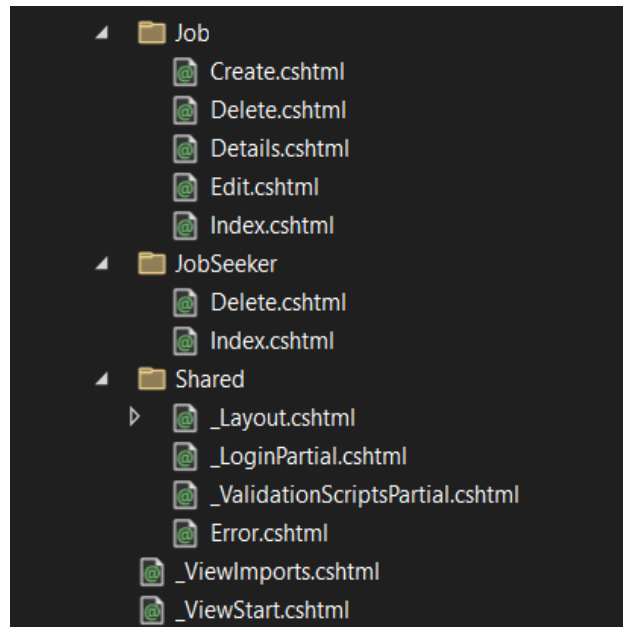


Figure 15: Folder Structure 8

II. Source code samples of the application with explanation:

1. Controllers

a. ApplicationController

This controller is in charge of handling job applications within the application.


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using FPT_JOBPORTAL;
using FPT_JOBPORTAL.Data;
using Microsoft.AspNetCore.Authorization;

namespace FPT_JOBPORTAL.Controllers
{
    [Authorize(Roles = "Job Seeker, Admin, Employer")]

    1 reference
    public class ApplicationController : Controller
    {
        private readonly ApplicationDbContext _context;
        private readonly IWebHostEnvironment _webHostEnvironment;

        0 references
        public ApplicationController(ApplicationDbContext context, IWebHostEnvironment webHostEnvironment)
        {
            _context = context;
            _webHostEnvironment = webHostEnvironment;
        }

        3 references
        public async Task<IActionResult> Index()
        {
            var applicationDbContext = _context.Application.Include(a => a.Job);
            return View(await applicationDbContext.ToListAsync());
        }

        0 references
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
        }
    }
}
```

Figure 1: ApplicationController 1

```
var application = await _context.Application
    .Include(a => a.Job)
    .FirstOrDefaultAsync(m => m.Id == id);
if (application == null)
{
    return NotFound();
}

return View(application);
}

0 references
public IActionResult Create()
{
    ViewData["JobId"] = new SelectList(_context.JobModel, "Id", "NameJob");
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create([Bind("Id,FullName,Email,Phone,Education,JobId,Resume,Status")] Application application)
{
    if (ModelState.IsValid)
    {
        IFormFile Resume = Request.Form.Files["Resume_file"];
        string wwwRootPath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot");
        string resumePath = Path.Combine(wwwRootPath, @"pdf/Resume");
        if (Resume != null)
        {
            string fileName = Guid.NewGuid().ToString() + Path.GetExtension(Resume.FileName);
            using (var fileStream = new FileStream(Path.Combine(resumePath, fileName), FileMode.Create))
            {
                Resume.CopyTo(fileStream);
            }
            application.Resume = fileName;
        }
        application.Status = Status.Pending;
    }
}
```

Figure 2: ApplicationController 2

```

        application.Status = Status.Pending;
        _context.Add(application);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    ViewData["JobId"] = new SelectList(_context.JobModel, "Id", "NameJob", application.JobId);
    return View(application);
}

0 references
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var application = await _context.Application.FindAsync(id);
    if (application == null)
    {
        return NotFound();
    }

    ViewData["JobId"] = new SelectList(_context.JobModel, "Id", "NameJob", application.JobId);
    return View(application);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("Id,FullName,Email,Phone,Education,JobId,Resume,Status")] Application application)
{
    if (id != application.Id)
    {
        return NotFound();
    }
}

```

Figure 3: ApplicationController 3

```

[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Edit(int id, [Bind("Id,FullName,Email,Phone,Education,JobId,Resume,Status")] Application application)
{
    if (id != application.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            var existingApplication = await _context.Application.FindAsync(id);
            if (existingApplication == null)
            {
                return NotFound();
            }

            IFormFile Resume_file = Request.Form.Files["resume_file"];
            string wwwRootPath = _webHostEnvironment.WebRootPath;
            string resumePath = Path.Combine(wwwRootPath, @"pdf/Resume");
            if (Resume_file != null)
            {
                if (!string.IsNullOrEmpty(application.Resume))
                {
                    string oldResumePath = Path.Combine(wwwRootPath, "pdf/Resume", application.Resume);
                    // Console.WriteLine("Old resume path: " + oldResumePath);
                    FileInfo fileInfo = new FileInfo(oldResumePath);
                    if (fileInfo.Exists)
                    {
                        fileInfo.Delete();
                    }
                }
            }
        }
    }
}

```

Figure 4: ApplicationController 4

```

        string fileName = Guid.NewGuid().ToString() + Path.GetExtension(Resume_file.FileName);
        using (var fileStream = new FileStream(Path.Combine(resumePath, fileName), FileMode.Create))
        {
            Resume_file.CopyTo(fileStream);
        }
        application.Resume = fileName;
    }
    else
    {
        application.Resume = existingApplication.Resume;
    }

    // Preserve the existing status if not provided in the form
    if (application.Status == null)
    {
        application.Status = existingApplication.Status;
    }

    _context.Entry(existingApplication).CurrentValues.SetValues(application);
    await _context.SaveChangesAsync();
}
catch (DbUpdateConcurrencyException)
{
    if (!ApplicationExists(application.Id))
    {
        return NotFound();
    }
    else
    {
        throw;
    }
}

```

Figure 5: ApplicationController 5

```

        return RedirectToAction(nameof(Index));
    }
    ViewData["JobId"] = new SelectList(_context.JobModel, "Id", "NameJob", application.JobId);
    return View(application);
}

0 references
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var application = await _context.Application
        .Include(a => a.Job)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (application == null)
    {
        return NotFound();
    }

    return View(application);
}

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var application = await _context.Application.FindAsync(id);
    if (application != null)
    {
        context.Application.Remove(application);
    }
}

```

Figure 6: ApplicationController 6

```

        _context.Application.Remove(application);
    }

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

1 reference
private bool ApplicationExists(int id)
{
    return _context.Application.Any(e => e.Id == id);
}

0 references
public IActionResult ViewResume(string resume)
{
    string filePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot/pdf/Resume", resume);

    if (!System.IO.File.Exists(filePath))
    {
        return NotFound();
    }

    return PhysicalFile(filePath, "application/pdf");
}

```

Figure 7: ApplicationController 7

The `ApplicationController` class is defined and inherits from the `Controller` base class provided by ASP.NET Core. It has two private fields: `_context`, which is of type `ApplicationDbContext`, and `_webHostEnvironment`, which is of type `IWebHostEnvironment`. These fields are used to interact with the application's data context and the web hosting environment, respectively.

The controller's constructor accepts instances of `ApplicationDbContext` and `IWebHostEnvironment` as parameters and assigns them to the respective private fields.

The class is also decorated with the `[Authorize(Roles = "Job Seeker, Admin, Employer")]` attribute, indicating that only authenticated users who belong to one of the specified roles ("Job Seeker," "Admin," or "Employer") can access its actions:

The controller defines several action methods, each corresponding to a different HTTP endpoint.

- **Index:** This action fetches a list of applications from the database and passes it to the associated view.
- **Details:** This action retrieves the details of a specific application using the provided `id` parameter and sends it to the related view.
- **Create:** This action displays a form for creating a new application and processes the form submission to save the new application in the database.
- **Edit:** This action displays a form for editing an existing application and manages the form submission to update the application's information in the database.
- **Delete:** This action shows a confirmation page to delete an application and handles the application's removal from the database upon user confirmation.

TỔ CHỨC GIÁO DỤC FPT

- **ViewResume:** This action retrieves and returns the resume file linked to an application. It accepts a ``resume`` parameter, which specifies the file name, and returns the file as a physical file response.

The Create and Edit actions use the ``[HttpPost]`` attribute, indicating they are triggered only by HTTP POST requests. They also utilize the ``[ValidateAntiForgeryToken]`` attribute to protect against cross-site request forgery (CSRF) attacks. Additionally, both actions use the ``[Bind]`` attribute to specify which properties of the ``Application`` model should be bound from the request data.

For file uploads, the Create action handles the upload of a resume file. It retrieves the uploaded file from the request using ``Request.Form.Files["Resume_file"]``, saves it to the server's file system, and sets the ``Resume`` property of the ``Application`` model to the file name. The ****Edit**** action performs a similar file upload process, allowing for a new resume file to be uploaded and updating the existing file if necessary.

The DeleteConfirmed action is executed when an application deletion is confirmed. It removes the application from the database and saves the changes. The ``ApplicationExists`` method serves as a helper method to check if an application with a specific ``id`` exists in the database.

b. AppRolesController

This controller is responsible for handling user roles within the application.

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
namespace FPT_JOBPORTAL.Controllers
{
    [Authorize ( Roles = "Admin" )]
    1 reference
    public class AppRolesController: Controller
    {
        private readonly RoleManager<IdentityRole> _roleManager;

        0 references
        public AppRolesController(RoleManager<IdentityRole> roleManager)
        {
            _roleManager = roleManager;
        }

        0 references
        public IActionResult Index()
        {
            var roles = _roleManager.Roles.Where(r => r.Name != "Admin");
            return View(roles);
        }

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

        [HttpPost]

        0 references
        public async Task<IActionResult> Create(IdentityRole model)
        {
            // avoid duplicate role
            if (!_roleManager.RoleExistsAsync(model.Name).GetAwaiter().GetResult())
            {
                _roleManager.CreateAsync(new IdentityRole(model.Name)).GetAwaiter().GetResult();
            }
        }
    }
}
```

Figure 8: AppRolesController 1

```

}
[HttpGet]
0 references
public IActionResult Edit(string id)
{
    var role = _roleManager.FindByIdAsync(id).GetAwaiter().GetResult();
    if (role == null)
    {
        return NotFound();
    }

    return View(role);
}

[HttpPost]
0 references
public async Task<IActionResult> Edit(IdentityRole model)
{
    var role = await _roleManager.FindByIdAsync(model.Id);
    if (role == null)
    {
        return NotFound();
    }

    role.Name = model.Name;
    var result = await _roleManager.UpdateAsync(role);

    return RedirectToAction("Index");
}

[HttpPost]

public async Task<IActionResult> Delete(string id)
{
    var role = await _roleManager.FindByIdAsync(id);
    if (role == null)
    {

```

Figure 9: AppRolesController 2


```
[HttpPost]
0 references
public async Task<IActionResult> Delete(string id)
{
    var role = await _roleManager.FindByIdAsync(id);
    if (role == null)
    {
        return NotFound();
    }

    var result = await _roleManager.DeleteAsync(role);

    return RedirectToAction("Index");
}
```

Figure 10: AppRolesController 3

The `AppRolesController` class is defined and inherits from the `Controller` base class provided by ASP.NET Core. It contains a private field, `_roleManager`, of type `RoleManager<IdentityRole>`, which is used to manage user roles and is injected via the constructor.

The controller's constructor accepts an instance of `RoleManager<IdentityRole>` as a parameter and assigns it to the private `_roleManager` field.

The class is decorated with the `[Authorize(Roles = "Admin")]` attribute, meaning that only authenticated users with the "Admin" role can access its actions.

The controller includes several action methods that map to different HTTP endpoints.

- **Index:** This action retrieves a list of roles from `_roleManager` and passes it to the associated view, excluding the "Admin" role from the list.
- **Create:** This action displays a form for creating a new role and handles the form submission to create the role using `_roleManager`. It checks for the existence of the role to prevent duplicates before creating it.
- **Edit:** This action displays a form for editing an existing role. It retrieves the role based on the provided `id` using `_roleManager` and processes the form submission to update the role's name.
- **Delete:** This action is responsible for deleting an existing role. It retrieves the role using the provided `id` through `_roleManager` and then deletes it.

The `[HttpGet]` and `[HttpPost]` attributes are used to define the HTTP verb associated with each action method.

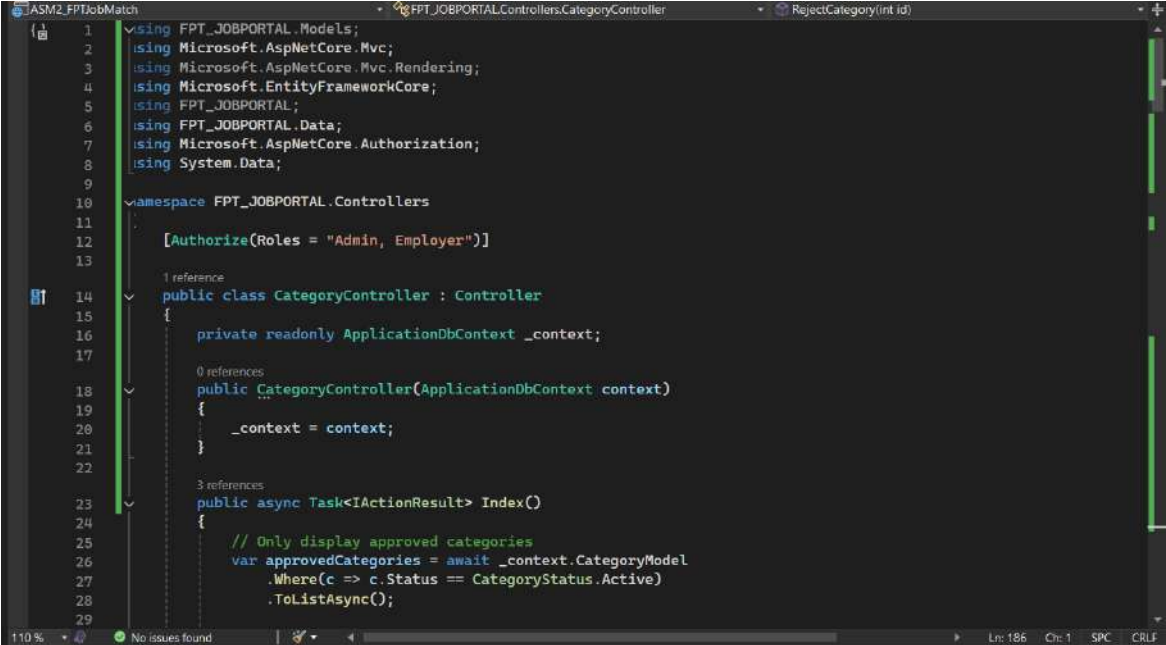
In the Create action, the `RoleExistsAsync` method of `_roleManager` is used to check if a role with the given name already exists. If it does not, a new role is created using `_roleManager`'s `CreateAsync` method.

The Edit action retrieves the existing role from `_roleManager` using the provided `id` parameter, updates the role's name, and saves the changes with the `UpdateAsync` method of `_roleManager`.

Similarly, the Delete action fetches the existing role from `_roleManager` using the given `id` parameter and removes it using the `DeleteAsync` method of `_roleManager`.

c. CategoryController

This controller is in charge of managing categories within a job portal application.



```

1  using FPT_JOBPORTAL.Models;
2  using Microsoft.AspNetCore.Mvc;
3  using Microsoft.AspNetCore.Mvc.Rendering;
4  using Microsoft.EntityFrameworkCore;
5  using FPT_JOBPORTAL;
6  using FPT_JOBPORTAL.Data;
7  using Microsoft.AspNetCore.Authorization;
8  using System.Data;
9
10 namespace FPT_JOBPORTAL.Controllers
11 {
12     [Authorize(Roles = "Admin, Employer")]
13
14     1 reference
15     public class CategoryController : Controller
16     {
17         1 reference
18         private readonly ApplicationDbContext _context;
19
20         0 references
21         public CategoryController(ApplicationDbContext context)
22         {
23             _context = context;
24         }
25
26         3 references
27         public async Task<IActionResult> Index()
28         {
29             // Only display approved categories
30             var approvedCategories = await _context.CategoryModel
31                 .Where(c => c.Status == CategoryStatus.Active)
32                 .ToListAsync();
33         }
34     }
35 }

```

Figure 11: CategoryController 1

```

28      .ToListAsync();
29
30      return View(approvedCategories);
31  }
32
33  0 references
34  public async Task<IActionResult> Details(int? id)
35  {
36      if (id == null)
37      {
38          return NotFound();
39      }
40
41      var categoryModel = await _context.CategoryModel
42          .FirstOrDefaultAsync(m => m.Id == id);
43      if (categoryModel == null)
44      {
45          return NotFound();
46      }
47
48      return View(categoryModel);
49  }
50
51  0 references
52  public IActionResult Create()
53  {
54      return View();
55  }
56  [Authorize(Roles = "Employer")]
57  [HttpPost]

```

Figure 12: CategoryController 2

```

57  [HttpPost]
58  [ValidateAntiForgeryToken]
59  0 references
60  public async Task<IActionResult> Create([Bind("Id,Name,Description,Status")] CategoryModel categoryModel)
61  {
62      if (ModelState.IsValid)
63      {
64          categoryModel.Status = CategoryStatus.Pending; // Đặt trạng thái là đang chờ phê duyệt
65          _context.Add(categoryModel);
66          await _context.SaveChangesAsync();
67          return RedirectToAction(nameof(Index));
68      }
69      return View(categoryModel);
70  }
71
72  [Authorize(Roles = "Admin")]
73  2 references
74  public async Task<IActionResult> PendingCategories()
75  {
76      var pendingCategories = await _context.CategoryModel
77          .Where(c => c.Status == CategoryStatus.Pending)
78          .ToListAsync();
79
80      return View(pendingCategories);
81  }
82  [Authorize(Roles = "Admin")]
83  [HttpPost]
84  [ValidateAntiForgeryToken]
85  0 references
86  public async Task<IActionResult> ApproveCategory(int id)

```

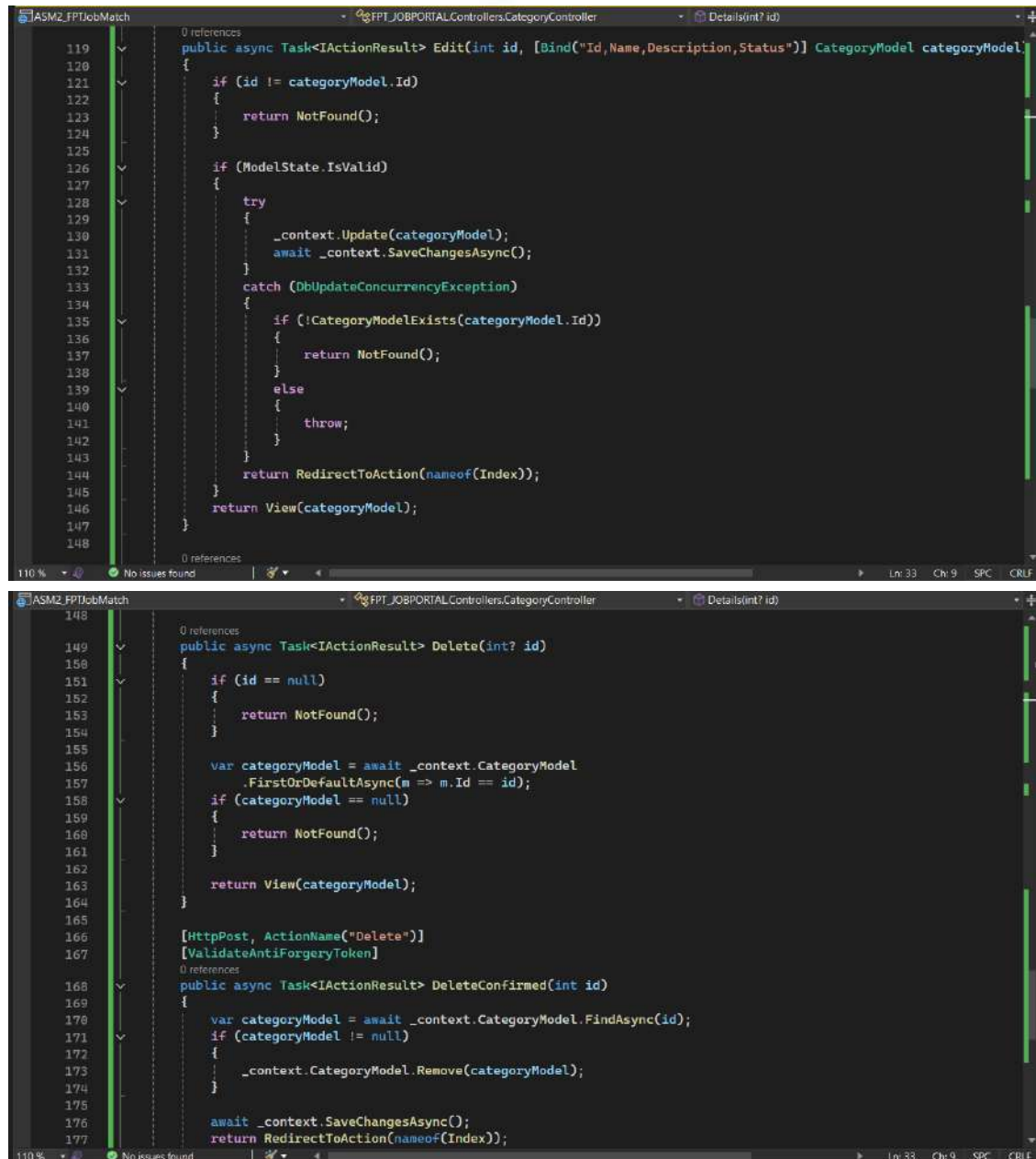
Figure 13: CategoryController 3

```

84  public async Task<IActionResult> ApproveCategory(int id)
85  {
86      var category = await _context.CategoryModel.FindAsync(id);
87      if (category == null)
88      {
89          return NotFound();
90      }
91
92      category.Status = CategoryStatus.Active;
93      _context.Update(category);
94      await _context.SaveChangesAsync();
95
96      return RedirectToAction(nameof(PendingCategories));
97  }
98
99  [Authorize(Roles = "Admin")]
100  [HttpPost]
101  [ValidateAntiForgeryToken]
102  0 references
103  public async Task<IActionResult> RejectCategory(int id)
104  {
105      var category = await _context.CategoryModel.FindAsync(id);
106      if (category == null)
107      {
108          return NotFound();
109      }
110
111      category.Status = CategoryStatus.Denied;
112      _context.Update(category);
113      await _context.SaveChangesAsync();
114
115      return RedirectToAction(nameof(PendingCategories));

```

Figure 14: CategoryController 4



```

119 public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Description,Status")] CategoryModel categoryModel)
120 {
121     if (id != categoryModel.Id)
122     {
123         return NotFound();
124     }
125
126     if (ModelState.IsValid)
127     {
128         try
129         {
130             _context.Update(categoryModel);
131             await _context.SaveChangesAsync();
132         }
133         catch (DbUpdateConcurrencyException)
134         {
135             if (!CategoryModelExists(categoryModel.Id))
136             {
137                 return NotFound();
138             }
139             else
140             {
141                 throw;
142             }
143         }
144         return RedirectToAction(nameof(Index));
145     }
146     return View(categoryModel);
147 }
148
149 public async Task<IActionResult> Delete(int? id)
150 {
151     if (id == null)
152     {
153         return NotFound();
154     }
155
156     var categoryModel = await _context.CategoryModel
157         .FirstOrDefaultAsync(m => m.Id == id);
158     if (categoryModel == null)
159     {
160         return NotFound();
161     }
162     return View(categoryModel);
163 }
164
165 [HttpPost, ActionName("Delete")]
166 [ValidateAntiForgeryToken]
167 public async Task<IActionResult> DeleteConfirmed(int id)
168 {
169     var categoryModel = await _context.CategoryModel.FindAsync(id);
170     if (categoryModel != null)
171     {
172         _context.CategoryModel.Remove(categoryModel);
173     }
174
175     await _context.SaveChangesAsync();
176     return RedirectToAction(nameof(Index));
177 }

```

Figure 15: CategoryController 5

The `CategoryController` class is defined and inherits from the `Controller` base class provided by ASP.NET Core. It has a private field, `_context`, of type `ApplicationDbContext`, which is used to interact with the database and is injected via the constructor.

The constructor of the controller takes an instance of `ApplicationDbContext` as a parameter and assigns it to the private `_context` field.

The class is decorated with the `[Authorize(Roles = "Admin, Employer, Job Seeker")]` attribute, indicating that only authenticated users with the "Admin," "Employer," or "Job Seeker" roles can access its actions.

The controller provides several action methods corresponding to different HTTP endpoints:

- **Index:** Retrieves a list of categories from `_context` and passes them to the associated view.
- **Details:** Retrieves and displays the details of a specific category based on the provided `id` parameter from `_context`.
- **Create:** Displays a form to create a new category and handles form submissions to add the category to the database using `_context`.
- **Edit:** Displays a form to edit an existing category, retrieves the category by `id` from `_context`, and handles form submissions to update the category in the database.
- **Delete:** Displays a confirmation page for deleting an existing category, retrieves the category by `id` from `_context`, and handles form submissions to delete the category from the database.
- **CategoryModelExists:** A private helper method that checks whether a category with the given `id` exists in `_context`.

The `[HttpPost]` and `[HttpGet]` attributes specify the HTTP verb for each action method. The `[ValidateAntiForgeryToken]` attribute is used to prevent Cross-Site Request Forgery (CSRF) attacks by validating the anti-forgery token sent with the form.

In the Create action, the `Status` property of the `CategoryModel` object is set to `CategoryStatus.Pending`. It is then added to `_context` using the `_context.Add` method, and changes are saved to the database with `_context.SaveChangesAsync`. The Edit action updates the category in `_context` using the `_context.Update` method and saves the changes with `_context.SaveChangesAsync`. It also handles concurrency conflicts with `DbUpdateConcurrencyException`. The DeleteConfirmed action removes the category from `_context` using `_context.CategoryModel.Remove` and saves the changes with `_context.SaveChangesAsync`. The `CategoryModelExists` method checks for the existence of a category with the given `id` in `_context` using the LINQ `Any` method.

d. EmployerController

This controller handles the management of employers within a job portal application.

TỔ CHỨC GIÁO DỤC FPT

```
using FPT_JOBPORTAL.Models;
using FPT_JOBPORTAL.Repository.IRepository;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace FPT_JOBPORTAL.Controllers
{
    [Authorize(Roles = "Admin")]
    public class EmployerController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;

        public EmployerController(IUnitOfWork unitOfWork)
        {
            _unitOfWork = unitOfWork;
        }

        public IActionResult Index()
        {
            List<ApplicationUser> employerList = _unitOfWork.ApplicationUserRepository.GetAll(u => u.Role == "Employer").ToList();
            return View(employerList);
        }
    }
}
```

Figure 16: EmployerController 1

```
public IActionResult Delete(string id)
{
    var employer = _unitOfWork.ApplicationUserRepository.Get(u => u.Id == id);
    if (employer == null)
    {
        return NotFound();
    }
    return View(employer);
}

// POST: Employer/Delete
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public IActionResult DeletePost(string id)
{
    var user = _unitOfWork.ApplicationUserRepository.Get(u => u.Id == id);
    if (user == null)
    {
        return NotFound();
    }
    _unitOfWork.ApplicationUserRepository.Delete(user);
    _unitOfWork.Save();
    return RedirectToAction(nameof(Index));
}
}
```

Figure 17: EmployerController 2

The `EmployerController` class, which inherits from ASP.NET Core's `Controller` base class, manages employer-related operations within a job portal application. It includes a private field `_unitOfWork` of type `IUnitOfWork`, injected through the constructor, to interact with data repositories.

The constructor assigns an `IUnitOfWork`` instance to ``_unitOfWork``, allowing access to various repositories.

Decorated with the ``[Authorize(Roles = "Admin")]` attribute, this controller restricts access to users with the "Admin" role.

The class includes several action methods:

- **Index:** Retrieves a list of employers from ``_unitOfWork`` by calling ``GetAll`` on the ``ApplicationUserRepository``, filters the list for the "Employer" role, and passes it to the view.
- **Delete:** Retrieves details of a specific employer by ``id`` from ``_unitOfWork`` using ``Get``, and returns these details to the view.
- **DeletePost:** Handles form submissions for employer deletions. It retrieves the employer by ``id``, deletes the record via ``Delete``, saves the changes with ``Save``, and redirects to the ``Index`` action to refresh the employer list.

The ``[HttpPost]` attribute indicates that ``DeletePost`` responds to POST requests, and the ``[ValidateAntiForgeryToken]` attribute helps prevent CSRF attacks by validating the anti-forgery token.

``IUnitOfWork`` provides access to ``ApplicationUserRepository``, facilitating interactions with employer data.

e. HomeController

This controller manages requests associated with the home page and handles error scenarios.

TỔ CHỨC GIÁO DỤC FPT

```
using System.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using FPT_JOBPORTAL.Models;

namespace FPT_JOBPORTAL.Controllers;

3 references
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;

    0 references
    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }

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

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

    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
    0 references
    public IActionResult Error()
    {
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
    }
}
```

Figure 18: HomeController

The `HomeController` class inherits from ASP.NET Core's `Controller` base class and has a private `_logger` field of type `ILogger<HomeController>`, which is used for logging and injected via the constructor.

The constructor takes an `ILogger<HomeController>` instance and assigns it to the `_logger` field for logging purposes.

The controller includes the following action methods:

- **Index:** Renders the home page view and returns a `ViewResult` for the Index view.
- **Privacy:** Renders the privacy policy view and returns a `ViewResult` for the Privacy view.
- **Error:** Handles error scenarios and renders the error view. This method uses the `[ResponseCache]` attribute to prevent caching by setting `Duration` to 0, `Location` to `ResponseCacheLocation.None`, and `NoStore` to true. It creates an `ErrorViewModel` object with a `RequestId` property, which is set to the current request ID or a unique identifier if the request ID is unavailable. The method then

returns a `ActionResult` for the Error view, passing the `ErrorViewModel` to the view.

The `[ResponseCache]` attribute configures caching settings, and the `Error` method retrieves the request ID or generates a unique identifier using `Activity.Current?.Id ?? HttpContext.TraceIdentifier`. The `ILogger<HomeController>` provides logging functionality to the controller.

f. `HomeController`

This controller manages job listings in a job portal application.

```
[Authorize(Roles = "Admin, Employer, Job Seeker")]

1 reference
public class JobController : Controller
{
    private readonly ApplicationDbContext _context;

    0 references
    public JobController(ApplicationDbContext context)
    {
        _context = context;
    }

    3 references
    public async Task<IActionResult> Index()
    {
        var applicationDbContext = _context.JobModel.Include(j => j.Category);
        return View(await applicationDbContext.ToListAsync());
    }

    0 references
    public async Task<IActionResult> Details(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var jobModel = await _context.JobModel
            .Include(j => j.Category)
            .FirstOrDefaultAsync(m => m.Id == id);
        if (jobModel == null)
        {
            return NotFound();
        }

        return View(jobModel);
    }
}
```

Figure 19: JobController 1

```

0 references
public IActionResult Create()
{
    ViewData["CategoryId"] = new SelectList(_context.CategoryModel, "Id", "Name");
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create([Bind("Id, NameJob, CategoryId, DescriptionJob, Company, DatePost, Salary, Requirement")] JobModel jobModel)
{
    if (ModelState.IsValid)
    {
        var category = await _context.CategoryModel.FindAsync(jobModel.CategoryId);

        if (category.Status == CategoryStatus.Pending)
        {
            ModelState.AddModelError(string.Empty, "Cannot create job for a category with pending status.");
            ViewData["CategoryId"] = new SelectList(_context.CategoryModel, "Id", "Name", jobModel.CategoryId);
            return View(jobModel);
        }

        jobModel.DatePost = DateTime.Now;
        _context.Add(jobModel);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["CategoryId"] = new SelectList(_context.CategoryModel, "Id", "Name", jobModel.CategoryId);
    return View(jobModel);
}

```

Figure 20: JobController 2

```

0 references
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var jobModel = await _context.JobModel.FindAsync(id);
    if (jobModel == null)
    {
        return NotFound();
    }
    ViewData["CategoryId"] = new SelectList(_context.CategoryModel, "Id", "Name", jobModel.CategoryId);
    return View(jobModel);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("Id, NameJob, CategoryId, DescriptionJob, Company, DatePost, Salary, Requirement")] JobModel jobModel)
{
    if (id != jobModel.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            jobModel.DatePost = DateTime.Now;
            _context.Update(jobModel);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!JobModelExists(jobModel.Id))
            {
                return NotFound();
            }
        }
    }
}

```

Figure 21: JobController 3

```

        else
        {
            throw;
        }
    }
    return RedirectToAction(nameof(Index));
}
}
ViewData["CategoryId"] = new SelectList(_context.CategoryModel, "Id", "Name", jobModel.CategoryId);
return View(jobModel);
}

public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var jobModel = await _context.JobModel
        .Include(j => j.Category)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (jobModel == null)
    {
        return NotFound();
    }

    return View(jobModel);
}
}

```

Figure 22: JobController 4


```

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var jobModel = await _context.JobModel.FindAsync(id);
    if (jobModel != null)
    {
        _context.JobModel.Remove(jobModel);
    }

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

1 reference
private bool JobModelExists(int id)
{
    return _context.JobModel.Any(e => e.Id == id);
}
}

```

Figure 23: JobController 5

The `JobController` class, inheriting from ASP.NET Core's `Controller` base class, manages job listings within the application. It is protected by the `[Authorize(Roles = "Admin, Employer, Job Seeker")]` attribute, allowing access only to users with those specific roles.

The controller uses a private field, `_context`, of type `ApplicationDbContext`, which is injected through the constructor. This database context provides access to the application's data.

The controller features several action methods for handling job listings:

- **Index:** Retrieves and displays a list of job listings from the `_context`, including associated categories.
- **Details:** Shows details of a specific job listing, including its associated category.
- **Create (GET):** Renders a form to create a new job listing and populates it with a list of categories.
- **Create (POST):** Handles the form submission to create a new job listing. If the model is valid, it saves the new listing to the database and redirects to the index. If not, it redisplayes the form with validation errors.
- **Edit (GET):** Renders a form to edit an existing job listing, populating it with current details and categories.
- **Edit (POST):** Handles the form submission to update an existing job listing. It updates the listing in the `_context` and saves the changes, or returns an error if the model is

invalid.

- **Delete:** Shows details of a specific job listing for confirmation before deletion.
- **DeleteConfirmed:** Handles the deletion of a job listing. It removes the listing from the `_context` and saves the changes, then redirects to the index.
- **JobModelExists:** A private method that checks if a job listing exists in the `_context` based on its ID.

The `[HttpPost]` attribute specifies that the `Create` and `Edit` actions are for POST requests. The `[ValidateAntiForgeryToken]` attribute helps prevent Cross-Site Request Forgery (CSRF) attacks by validating the anti-forgery token in form submissions.

Overall, `ApplicationDbContext` provides the necessary database access for managing job and category entities.

g. JobSeekerController

This controller manages job seekers within the job portal application.

```
[Authorize(Roles = "Admin")]
1 reference
public class JobSeekerController : Controller
{
    private readonly IUnitOfWork _unitOfWork;
    0 references
    public JobSeekerController(IUnitOfWork unitOfWork)
    {
        _unitOfWork = unitOfWork;
    }

    1 reference
    public IActionResult Index()
    {
        List<ApplicationUser> jobSeekerList = _unitOfWork.ApplicationUserRepository.GetAll(u => u.Role == "Job Seeker").ToList();
        return View(jobSeekerList);
    }
}
```

Figure 24: JobSeekerController 1

```

0 references
public IActionResult Delete(string id)
{
    var jobSeeker = _unitOfWork.ApplicationUserRepository.Get(u => u.Id == id);
    if (jobSeeker == null)
    {
        return NotFound();
    }
    return View(jobSeeker);
}

// POST: Employer/Delete
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references
public IActionResult DeletePost(string id)
{
    var user = _unitOfWork.ApplicationUserRepository.Get(u => u.Id == id);
    if (user == null)
    {
        return NotFound();
    }
    _unitOfWork.ApplicationUserRepository.Delete(user);
    _unitOfWork.Save();
    return RedirectToAction(nameof(Index));
}
}

```

Figure 25: JobSeekerController 2

The `JobSeekerController` class inherits from the `Controller` base class in ASP.NET Core and is decorated with the `[Authorize(Roles = "Admin")]` attribute, meaning that only users with the "Admin" role can access its actions. It has a private field `_unitOfWork` of type `IUnitOfWork`, which manages access to repositories and handles transactional operations.

The controller's constructor takes an `IUnitOfWork` instance and assigns it to `_unitOfWork`.

The `JobSeekerController` includes several action methods:

- **Index:** Retrieves and displays a list of job seekers from the `ApplicationUserRepository` in `_unitOfWork`, filtering users by the "Job Seeker" role.
- **Delete:** Retrieves details of a specific job seeker by ID from `_unitOfWork` and displays them. If the job seeker is not found, it returns a `NotFound` response.
- **DeletePost:** Handles the deletion of a job seeker. It retrieves the user by ID, deletes the user if found, saves the changes, and redirects to the `Index` action to show the updated list of job seekers.

The `[HttpPost]` attribute is used to specify that `DeletePost` should handle POST requests, while the

`[ValidateAntiForgeryToken]` attribute protects against cross-site request forgery (CSRF) attacks by validating the anti-forgery token included in the form submission.

2. Models

a. ApplicationModel

```

{
    3 references
    public enum Status
    {
        Pending,
        Accepted,
        Denied
    }

    18 references
    public class Application
    {
        [Key]
        11 references
        public int Id { get; set; }
        [DisplayName("Full Name")]
        12 references
        public string? FullName { get; set; }
        12 references
        public string? Email { get; set; }
        12 references
        public string? Phone { get; set; }
        12 references
        public string? Education { get; set; }
        8 references
        public int JobId { get; set; }
        18 references
        public string? Resume { get; set; }
        13 references
        public Status Status { get; set; }

        [ForeignKey("JobId")]
        9 references
        public virtual JobModel? Job { get; set; }
    }
}
    
```

Figure 26: ApplicationModel

The `Application` class includes the following properties:

- **Id:** Represents the unique identifier for the application. It is marked with the `[Key]` attribute, indicating it serves as the primary key for the entity.
- **FullName:** Represents the applicant's full name. It is annotated with the `[DisplayName]` attribute, which provides a user-friendly name for display purposes in views or UI.
- **Email:** Represents the applicant's email address.
- **Phone:** Represents the applicant's phone number.
- **Education:** Represents the applicant's educational background.
- **JobId:** Represents the foreign key to the associated job listing. It is annotated with the `[ForeignKey]` attribute, indicating that it is a foreign key linked to the `JobId`

property in the `JobModel` class.

- **Resume:** Represents the applicant's resume or CV.
- **Status:** Represents the application's status. It is an enumeration of type `Status`, which can have one of three values: `Pending`, `Accepted`, or `Denied`.
- **Job:** Represents the navigation property to the related job listing. The `[ForeignKey]` attribute specifies that this property is related to the `JobId` foreign key property in the `JobModel` class. The `virtual` keyword enables lazy loading of the related job entity.

The `Status` enumeration defines the possible values for the `Status` property: `Pending`, `Accepted`, and `Denied`.

The `[ForeignKey("JobId")]` attribute ensures that the `Job` navigation property is associated with the `JobId` foreign key.

b. ApplicationUser

```
namespace FPT_JOBPORTAL.Models
{
    public class ApplicationUser : IdentityUser
    {
        [Required]
        public string Name { get; set; }
        public string? StreetAddress { get; set; }
        public string? City { get; set; }
        public string? Role { get; set; }
    }
}
```

Figure 27: ApplicationUser

The `ApplicationUser` class extends the `IdentityUser` class, which is provided by the ASP.NET Core Identity framework and represents a user in the authentication system.

The `ApplicationUser` class includes the following properties:

- **Name:** Represents the user's name. It is marked with the `[Required]` attribute, meaning it is mandatory and must have a value.
- **StreetAddress:** Represents the user's street address.
- **City:** Represents the user's city.
- **Role:** Represents the user's role as a string. This property can be used to assign roles such as "Admin," "Job Seeker," or "Employer" to the user.

c. CategoryModel

```
namespace FPT_00010711
{
    public enum CategoryStatus
    {
        Pending,
        Active,
        Denied
    }

    public class CategoryModel
    {
        [Key]
        public int Id { get; set; }
        public string? Name { get; set; }
        public string? Description { get; set; }
        public CategoryStatus Status { get; set; }
    }
}
```

The `CategoryModel` class defines several properties related to a job category:

- **Id:** Represents the unique identifier for the category. It is marked with the `[Key]` attribute, indicating it serves as the primary key for the entity.
- **Name:** Represents the name of the category.
- **Description:** Describes the category.
- **Status:** Represents the status of the category. This property is an enumeration of type `CategoryStatus`, which is defined in the same namespace. The `CategoryStatus` enum has three possible values: `Pending`, `Active`, and `Denied`.

The `CategoryStatus` enumeration is defined to provide the possible values for the `Status` property, which includes three members: `Pending`, `Active`, and `Denied`.

The `[Key]` attribute is used to designate the `Id` property as the primary key of the entity.

d. JobModel

```

25 references
public class JobModel
{
    [Key]
    17 references
    public int Id { get; set; }
    [DisplayName("Job Name")]
    19 references
    public string? NameJob { get; set; }
    15 references
    public int CategoryId { get; set; }
    [DisplayName("Job Description")]
    17 references
    public string? DescriptionJob { get; set; }
    17 references
    public string? Company { get; set; }
    [DisplayName("Posted Date")]
    13 references
    public DateTime? DatePost { get; set; }
    17 references
    public double? Salary { get; set; }
    17 references
    public string? Requirement { get; set; }

    [ForeignKey("CategoryId")]
    10 references
    public virtual CategoryModel? Category { get; set; }
}

```

Figure 28: JobModel

The `JobModel` class defines several properties related to a job listing:

- **Id:** The unique identifier for the job listing. It is marked with the `[Key]` attribute, indicating it serves as the primary key of the entity.
- **NameJob:** Represents the job's name.
- **CategoryId:** Refers to the foreign key associated with the job's category. It is annotated with the `[ForeignKey]` attribute, linking it to the `CategoryId` property in the `CategoryModel` class.
- **DescriptionJob:** Describes the job's details.
- **Company:** Represents the company offering the job.
- **DatePost:** Indicates when the job was posted. This property uses the `[DisplayName]` attribute to provide a user-friendly name for the property in views or user interfaces.
- **Salary:** Indicates the salary for the job.
- **Requirement:** Describes the job's requirements.
- **Category:** A navigation property representing the related category. It is marked with the `[ForeignKey]` attribute, establishing a relationship with the `Category` navigation property in the `CategoryModel` class. The `virtual` keyword enables lazy loading for this related category.

The `[Key]` attribute identifies `Id` as the primary key of the entity. The `[DisplayName]` attribute provides a user-friendly display name for `DatePost` in views or UI, while the `[ForeignKey("CategoryId")]` attribute establishes that the `Category` navigation property is related to the `CategoryId` foreign key property.

TỔ CHỨC GIÁO DỤC FPT

3. View

a. Application

* Create Job Application:

```
@model FPT_JOBPORTAL.Application

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

<h4>Application</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create" method="post" enctype="multipart/form-data">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="FullName" class="control-label"></label>
                <input asp-for="FullName" class="form-control" />
                <span asp-validation-for="FullName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Phone" class="control-label"></label>
                <input asp-for="Phone" class="form-control" />
                <span asp-validation-for="Phone" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Education" class="control-label"></label>
                <select asp-for="Education" class="form-control">
                    <option value="">-- Select Education Level --</option>
                    <option value="Level 1 - Elementary I">Level 1 - Elementary I</option>
                    <option value="Level 2 - Elementary II">Level 2 - Elementary II</option>
                    <option value="Level 3 - Elementary III">Level 3 - Elementary III</option>
                    <option value="Level 4 - Intermediate">Level 4 - Intermediate</option>
                    <option value="Level 5 - College">Level 5 - College</option>
                    <option value="Level 6 - University">Level 6 - University</option>
                    <option value="Level 7 - Master">Level 7 - Master</option>
                    <option value="Level 8 - Doctorate">Level 8 - Doctorate</option>
                </select>
                <span asp-validation-for="Education" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="JobId" class="control-label"></label>
                <select asp-for="JobId" class="form-control" asp-items="ViewBag.JobId"></select>
            </div>
        </form>
    </div>
</div>
```

Figure 29: Create 1


```

<span asp-validation-for="Education" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="JobId" class="control-label"></label>
  <select asp-for="JobId" class="form-control" asp-items="ViewBag.JobId"></select>
</div>
<div class="form-group">
  <label asp-for="Resume" class="control-label"></label>
  <input asp-for="Resume" type="file" name="Resume_file" class="form-control" />
  <span asp-validation-for="Resume" class="text-danger"></span>
</div>
@* <div class="form-group">
  <label asp-for="Status" class="control-label"></label>
  <select asp-for="Status" class="form-control"></select>
  <span asp-validation-for="Status" class="text-danger"></span>
</div> *@
<div class="form-group">
  <input type="submit" value="Create" class="btn btn-primary" />
</div>
</form>
</div>
</div>
<div>
  <a asp-action="Index" class="btn btn-danger">Back to List</a>
</div>

@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

Figure 30: Create 2

* Delete Job Application:

```
@model FPT_JOBPORTAL.Application

@{
    ViewData["Title"] = "Delete";
}

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Application</h4>
    <hr />
    <dl class="row">
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.FullName)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.FullName)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Email)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Email)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Phone)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Phone)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Education)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Education)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Resume)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Resume)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Status)
        </dt>
    </dl>
</div>
```

Figure 31: Delete 1


```

7      </dd>
8      <dt class = "col-sm-2">
9          @Html.DisplayNameFor(model => model.Resume)
10     </dt>
11     <dd class = "col-sm-10">
12         @Html.DisplayFor(model => model.Resume)
13     </dd>
14     <dt class = "col-sm-2">
15         @Html.DisplayNameFor(model => model.Status)
16     </dt>
17     <dd class = "col-sm-10">
18         @Html.DisplayFor(model => model.Status)
19     </dd>
20     <dt class = "col-sm-2">
21         @Html.DisplayNameFor(model => model.Job)
22     </dt>
23     <dd class = "col-sm-10">
24         @Html.DisplayFor(model => model.Job.Id)
25     </dd>
26 </dl>
27
28 <form asp-action="Delete">
29     <input type="hidden" asp-for="Id" />
30     <input type="submit" value="Delete" class="btn btn-danger" /> |
31     <a asp-action="Index" class="btn btn-danger">Back to List</a>
32 </form>
33 </div>
34

```

Figure 32: Delete 2

* Job Application Details:

```
@model FPT_JOBPORTAL.Application

@{
    ViewData["Title"] = "Details";
}

<h1>Details</h1>

<div>
    <h4>Application</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.FullName)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.FullName)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Email)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Email)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Phone)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Phone)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Education)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Education)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Resume)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Resume)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Status)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Status)
        </dd>
    </dl>
</div>
```

Figure 33: Details 1

```

35         @Html.DisplayFor(model => model.Education)
36     </dd>
37     <dt class = "col-sm-2">
38         @Html.DisplayNameFor(model => model.Resume)
39     </dt>
40     <dd class = "col-sm-10">
41         @Html.DisplayFor(model => model.Resume)
42     </dd>
43     <dt class = "col-sm-2">
44         @Html.DisplayNameFor(model => model.Status)
45     </dt>
46     <dd class = "col-sm-10">
47         @Html.DisplayFor(model => model.Status)
48     </dd>
49     <dt class = "col-sm-2">
50         @Html.DisplayNameFor(model => model.Job)
51     </dt>
52     <dd class = "col-sm-10">
53         @Html.DisplayFor(model => model.Job.NameJob)
54     </dd>
55 </dl>
56 </div>
57 <div>
58     <a asp-action="Edit" asp-route-id="@Model?.Id">Edit</a>
59     <a asp-action="Index" class="btn btn-danger">Back to List</a>
60 </div>
61

```

Figure 34: Details 2

* Edit Job Application:

```

1  @model FPT_JOBPORTAL.Application
2
3  @{
4      ViewData["Title"] = "Edit";
5  }
6
7  <h1>Edit</h1>
8
9  <h4>Application</h4>
10 <hr />
11 <div class="row">
12     <div class="col-md-4">
13         <form asp-action="Edit" method="post" enctype="multipart/form-data">
14             <div asp-validation-summary="ModelOnly" class="text-danger"></div>
15             <input type="hidden" asp-for="Id" />
16             <div class="form-group">
17                 <label asp-for="FullName" class="control-label"></label>
18                 <input asp-for="FullName" class="form-control" />
19                 <span asp-validation-for="FullName" class="text-danger"></span>
20             </div>
21             <div class="form-group">
22                 <label asp-for="Email" class="control-label"></label>
23                 <input asp-for="Email" class="form-control" />
24                 <span asp-validation-for="Email" class="text-danger"></span>
25             </div>
26             <div class="form-group">
27                 <label asp-for="Phone" class="control-label"></label>
28                 <input asp-for="Phone" class="form-control" />
29                 <span asp-validation-for="Phone" class="text-danger"></span>
30             </div>
31             <div class="form-group">
32                 <label asp-for="Education" class="control-label"></label>
33                 <input asp-for="Education" class="form-control" />
34                 <span asp-validation-for="Education" class="text-danger"></span>
35             </div>
36             <div class="form-group">
37                 <label asp-for="JobId" class="control-label"></label>
38                 <select asp-for="JobId" class="form-control" asp-items="ViewBag.JobId"></select>
39                 <span asp-validation-for="JobId" class="text-danger"></span>
40             </div>
41             <div class="form-group">
42                 <label asp-for="Resume" class="control-label"></label>
43                 <input asp-for="Resume" type="file" name="Resume_file" class="form-control" />
44                 <span asp-validation-for="Resume" class="text-danger"></span>
45             </div>
46             <div class="form-group">
47                 <label asp-for="Status" class="control-label"></label>
48                 <select asp-for="Status" class="form-control">

```

Figure 35: Edit 1


```

38      <select asp-for="JobId" class="form-control" asp-items="ViewBag.JobId"></select>
39      <span asp-validation-for="JobId" class="text-danger"></span>
40    </div>
41    <div class="form-group">
42      <label asp-for="Resume" class="control-label"></label>
43      <input asp-for="Resume" type="file" name="Resume_file" class="form-control" />
44      <span asp-validation-for="Resume" class="text-danger"></span>
45    </div>
46    <div class="form-group">
47      <label asp-for="Status" class="control-label"></label>
48      <select asp-for="Status" class="form-control">
49        @foreach (var status in Enum.GetNames(typeof(FPT_JOBPORTAL.Status)))
50        {
51          <option value="@status">@status</option>
52        }
53      </select>
54      <span asp-validation-for="Status" class="text-danger"></span>
55    </div>
56    <div class="form-group">
57      <input type="submit" value="Save" class="btn btn-primary" />
58    </div>
59  </form>
60 </div>
61 </div>
62
63 <div>

```

Figure 36: Edit 2

* Index of Job Application:

```

1  @model IEnumerable<FPT_JOBPORTAL.Application>
2
3  @{
4      ViewData["Title"] = "Application";
5  }
6
7  <h1>Application</h1>
8  @if (User.IsInRole("Job Seeker"))
9  {
10     <p>
11         <a asp-action="Create" class="btn btn-primary">Create New</a>
12     </p>
13 }
14 <table class="table">
15     <thead>
16     <tr>
17         <th>
18             @Html.DisplayNameFor(model => model.FullName)
19         </th>
20         <th>
21             @Html.DisplayNameFor(model => model.Email)
22         </th>
23         <th>
24             @Html.DisplayNameFor(model => model.Phone)
25         </th>
26         <th>
27             @Html.DisplayNameFor(model => model.Education)
28         </th>
29         <th>
30             @Html.DisplayNameFor(model => model.Resume)
31         </th>
32         <th>
33             @Html.DisplayNameFor(model => model.Status)
34         </th>
35         <th>
36             @Html.DisplayNameFor(model => model.Job)
37         </th>
38     </tr>
39     </thead>
40     <tbody>
41     @foreach (var item in Model) {
42

```

Figure 37: Index 1

```

42 @foreach (var item in Model) {
43     <tr>
44         <td>
45             @Html.DisplayFor(modelItem => item.FullName)
46         </td>
47         <td>
48             @Html.DisplayFor(modelItem => item.Email)
49         </td>
50         <td>
51             @Html.DisplayFor(modelItem => item.Phone)
52         </td>
53         <td>
54             @Html.DisplayFor(modelItem => item.Education)
55         </td>
56         <td>
57             *@ @Html.DisplayFor(modelItem => item.Resume) *@
58             <a asp-action="ViewResume" asp-route-resume="@item.Resume" class="btn btn-primary">View Resume</a>
59         </td>
60         <td>
61             @Html.DisplayFor(modelItem => item.Status)
62         </td>
63         <td>
64             @Html.DisplayFor(modelItem => item.Job.NameJob)
65         </td>
66         <td>
67             @if (User.IsInRole("Employer"))
68             {
69                 <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-primary">Edit</a>
70                 <a asp-action="Delete" asp-route-id="@item.Id" class="btn btn-danger">Delete</a>
71             }
72             @if (User.IsInRole("Admin"))
73             {
74                 <a asp-action="Details" asp-route-id="@item.Id" class="btn btn-primary">Details</a>
75             }
76         </td>
77     </tr>
78 }
79 </tbody>
80 </table>
81

```

Figure 38: Index 2

b. AppRoles

* Create Role:

TỔ CHỨC GIÁO DỤC FPT

Alliance with  Education

```
@using Microsoft.AspNetCore.Identity;
@model IdentityRole
@{
    ViewData["Title"] = "Create Role";
}

<h1>@ViewData["Title"]</h1>

<div class="row">
    <div class="col-md-4">
        <form asp-action="Create" method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-floating mb-3">
                <input asp-for="Name" class="form-control" placeholder="Enter role name." aria-required="true"/>
                <label asp-for="Name" class="form-label"></label>
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <button type="submit" class="btn btn-primary">Create</button>
        </form>
    </div>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

Figure 39: Create

* Delete Role:

```
@model Microsoft.AspNetCore.Identity.IdentityRole

@{
    ViewData["Title"] = "Delete Role";
}

<h2>Delete Role</h2>

<h3>Are you sure you want to delete this role?</h3>
<div>
    <dl class="row">
        <dt class="col-sm-2">
            Role ID:
        </dt>
        <dd class="col-sm-10">
            @Model.Id
        </dd>
        <dt class="col-sm-2">
            Role Name:
        </dt>
        <dd class="col-sm-10">
            @Model.Name
        </dd>
    </dl>

    <form asp-action="Delete" method="post">
        <input type="hidden" asp-for="Id" />
        <button type="submit" class="btn btn-danger">Delete</button>
        <a asp-action="Index" class="btn btn-secondary">Cancel</a>
    </form>
</div>
```

Figure 40: Delete

* Edit Role:

```
1  @using Microsoft.AspNetCore.Identity;
2  @model IdentityRole
3
4  <h2>Edit Role</h2>
5
6  <form method="post">
7      <input type="hidden" asp-for="Id" />
8      <div>
9          <label asp-for="Name"></label>
10         <input asp-for="Name" />
11     </div>
12     <button type="submit">Save</button>
13 </form>
```

Figure 41: Edit

* Index of Role:

TỔ CHỨC GIÁO DỤC FPT

Alliance with  Education

```

1  @using Microsoft.AspNetCore.Identity;
2  @model IEnumerable<IdentityRole>
3
4  @{
5      ViewData["Title"] = "Roles";
6  }
7
8  <h1>Roles</h1>
9  <p>
10     <a asp-action="Create" class="btn btn-primary">Create New</a>
11 </p>
12 <table class="table">
13 <thead>
14 <tr>
15 <th> @Html.DisplayNameFor(model => model.Id)</th>
16 <th> @Html.DisplayNameFor(model => model.Name)</th>
17 <th></th>
18 </tr>
19 <tbody>
20 @foreach (var item in Model)
21 {
22 <tr>
23 <td>@Html.DisplayFor(modelItem => item.Id)</td>
24 <td>@Html.DisplayFor(modelItem => item.Name)</td>
25 <td>
26 <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-primary">Edit</a>
27 <form asp-action="Delete" asp-route-id="@item.Id" method="post" onsubmit="return confirm('Are you sure you want to delete this?')">
28 <input type="submit" class="btn btn-danger" value="Delete" />
29 </form>
30 </td>
31 </tr>
32 }
33 </table>

```

Figure 42: Delete

c. Category

* Create Category:

```

1  @model FPT_JOBPORTAL.CategoryModel
2
3  @{
4      ViewData["Title"] = "Create";
5  }
6
7  <h1>Create</h1>
8
9  <h4>Category</h4>
10 <hr />
11 <div class="row">
12     <div class="col-md-4">
13         <form asp-action="Create">
14             <div asp-validation-summary="ModelOnly" class="text-danger"></div>
15             <div class="form-group">
16                 <label asp-for="Name" class="control-label"></label>
17                 <input asp-for="Name" class="form-control" />
18                 <span asp-validation-for="Name" class="text-danger"></span>
19             </div>
20             <div class="form-group">
21                 <label asp-for="Description" class="control-label"></label>
22                 <input asp-for="Description" class="form-control" />
23                 <span asp-validation-for="Description" class="text-danger"></span>
24             </div>
25             @if (User.IsInRole("Admin"))
26             {
27                 <div class="form-group">
28                     <label asp-for="Status" class="control-label"></label>
29                     <select asp-for="Status" class="form-control">
30                         @foreach (var status in Enum.GetNames(typeof(FPT_JOBPORTAL.CategoryStatus)))
31                         {
32                             <option value="@status">@status</option>
33                         }
34                     </select>
35                     <span asp-validation-for="Status" class="text-danger"></span>
36                 </div>
37             }
38             <div class="form-group">
39                 <input type="submit" value="Create" class="btn btn-primary" />
40             </div>
41         </form>
42     </div>
43 </div>
44
45 <div>
46     <a asp-action="Index" class="btn btn-danger">Back to List</a>
47 </div>

```

Figure 43: Create 1

```

<div class="form-group">
  <input type="submit" value="Create" class="btn btn-primary" />
</div>
</form>
</div>
</div>

<div>
  <a asp-action="Index" class="btn btn-danger">Back to List</a>
</div>

@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

Figure 44: Create 2

* Delete Category:

```

1  @model FPT_JOBPORTAL.CategoryModel
2
3  @{
4    ViewData["Title"] = "Delete";
5  }
6
7  <h1>Delete</h1>
8
9  <h3>Are you sure you want to delete this?</h3>
10
11 <div>
12   <h4>Category</h4>
13   <hr />
14   <dl class="row">
15     <dt class = "col-sm-2">
16       @Html.DisplayNameFor(model => model.Name)
17     </dt>
18     <dd class = "col-sm-10">
19       @Html.DisplayFor(model => model.Name)
20     </dd>
21     <dt class = "col-sm-2">
22       @Html.DisplayNameFor(model => model.Description)
23     </dt>
24     <dd class = "col-sm-10">
25       @Html.DisplayFor(model => model.Description)
26     </dd>
27     <dt class = "col-sm-2">
28       @Html.DisplayNameFor(model => model.Status)
29     </dt>
30     <dd class = "col-sm-10">
31       @Html.DisplayFor(model => model.Status)
32     </dd>
33   </dl>
34
35   <form asp-action="Delete">
36     <input type="hidden" asp-for="Id" />
37     <input type="submit" value="Delete" class="btn btn-danger" /> |
38     <a asp-action="Index" class="btn btn-danger">Back to List</a>
39   </form>
40 </div>

```

Figure 45: Delete

* Category Details:


```

1  @model FPT_JOBPORTAL.CategoryModel
2
3  @{
4      ViewData["Title"] = "Details";
5  }
6
7  <h1>Details</h1>
8
9  <div>
10     <h4>CategoryModel</h4>
11     <hr />
12     <dl class="row">
13         <dt class="col-sm-2">
14             @Html.DisplayNameFor(model => model.Name)
15         </dt>
16         <dd class="col-sm-10">
17             @Html.DisplayFor(model => model.Name)
18         </dd>
19         <dt class="col-sm-2">
20             @Html.DisplayNameFor(model => model.Description)
21         </dt>
22         <dd class="col-sm-10">
23             @Html.DisplayFor(model => model.Description)
24         </dd>
25         <dt class="col-sm-2">
26             @Html.DisplayNameFor(model => model.Status)
27         </dt>
28         <dd class="col-sm-10">
29             @Html.DisplayFor(model => model.Status)
30         </dd>
31     </dl>
32 </div>
33 <div>
34     @* <a asp-action="Edit" asp-route-id="@Model?.Id">Edit</a> |
35     *@ <a asp-action="Index" class="btn btn-danger">Back to List</a>
36 </div>
37

```

Figure 46: Details

* Edit Category:

```

1  @model FPT_JOBPORTAL.CategoryModel
2
3  @{
4      ViewData["Title"] = "Edit";
5  }
6
7  <h1>Edit</h1>
8
9  <h4>Category</h4>
10 <hr />
11 <div class="row">
12     <div class="col-md-4">
13         <form asp-action="Edit">
14             <div asp-validation-summary="ModelOnly" class="text-danger"></div>
15             <input type="hidden" asp-for="Id" />
16             <div class="form-group">
17                 <label asp-for="Name" class="control-label"></label>
18                 <input asp-for="Name" class="form-control" />
19                 <span asp-validation-for="Name" class="text-danger"></span>
20             </div>
21             <div class="form-group">
22                 <label asp-for="Description" class="control-label"></label>
23                 <input asp-for="Description" class="form-control" />
24                 <span asp-validation-for="Description" class="text-danger"></span>
25             </div>
26             @if (User.IsInRole("Admin"))
27             {
28                 <div class="form-group">
29                     <label asp-for="Status" class="control-label"></label>
30                     <select asp-for="Status" class="form-control">
31                         @foreach (var status in Enum.GetNames(typeof(FPT_JOBPORTAL.CategoryStatus)))
32                         {
33                             <option value="@status">@status</option>
34                         }
35                     </select>
36                     <span asp-validation-for="Status" class="text-danger"></span>
37                 </div>
38             }
39             <div class="form-group">
40                 <input type="submit" value="Save" class="btn btn-primary" />
41             </div>
42         </form>
43     </div>
44 </div>
45

```

Figure 47: Edit 1


```

46     <div>
47         <a asp-action="Index" class="btn btn-danger">Back to List</a>
48     </div>
49
50     @section Scripts {
51         @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
52     }
53

```

Figure 48: Edit 2

* Index of Category:

```

1  @model IEnumerable<FPT_JOBPORTAL.CategoryModel>
2
3  @{
4      ViewData["Title"] = "Category";
5  }
6
7  <h1>Category</h1>
8  @if (User.IsInRole("Employer"))
9  {
10     <p>
11         <a asp-action="Create" class="btn btn-primary">Create New</a>
12     </p>
13 }
14 <table class="table">
15     <thead>
16     <tr>
17         <th>
18             @Html.DisplayNameFor(model => model.Name)
19         </th>
20         <th>
21             @Html.DisplayNameFor(model => model.Description)
22         </th>
23         <th>
24             @Html.DisplayNameFor(model => model.Status)
25         </th>
26     </tr>
27     </thead>
28     <tbody>
29     @foreach (var item in Model) {
30     <tr>
31         <td>
32             @Html.DisplayFor(modelItem => item.Name)
33         </td>
34         <td>
35             @Html.DisplayFor(modelItem => item.Description)
36         </td>
37         <td>
38             @Html.DisplayFor(modelItem => item.Status)
39         </td>
40         <td>
41             @if (User.IsInRole("Employer"))
42             {
43                 <a asp-action="Details" asp-route-id="@item.Id" class="btn btn-primary">Details</a>
44             }
45             @if (User.IsInRole("Admin"))
46             {
47

```

Figure 49: Index 1

```

42         if (User.IsInRole("Employee"))
43         {
44             <a asp-action="Details" asp-route-id="@item.Id" class="btn btn-primary">Details</a>
45         }
46         if (User.IsInRole("Admin"))
47         {
48             <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-primary">Edit</a>
49             <form asp-action="Delete" asp-route-id="@item.Id" method="post" onsubmit="return confirm('Are you sure you want to delete this?')">
50                 <input type="submit" class="btn btn-danger" value="Delete" />
51             </form>
52         }
53     </td>
54 </tr>
55 </tbody>
56 </table>
57
58

```

Figure 50: Index 2

d. Employer

* Delete Employer:

```

@model ApplicationUser

<h1>Delete Employer</h1>

<div>
    <h4>Are you sure you want to delete this Employer?</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            Email
        </dt>
        <dd class="col-sm-10">
            @Model.Email
        </dd>
    </dl>
    <form asp-action="Delete" method="post">
        <input type="hidden" asp-for="Id" />
        <div class="form-group">
            <input type="submit" value="Delete" class="btn btn-danger" />
            <a asp-action="Index" class="btn btn-secondary">Cancel</a>
        </div>
    </form>
</div>

```

Figure 51: Delete

* View List of Employer:

```

@model List<ApplicationUser>
<h1>Employer List</h1>
<table class="table table-bordered">
  <thead>
    <tr>
      <th>Full Name</th>
      <th>Email</th>
      <th>Phone Number</th>
      <th>Street Address</th>
      <th>City</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var applicationUser in Model)
    {
      <tr>
        <td>@applicationUser.Name</td>
        <td>@applicationUser.Email</td>
        <td>@applicationUser.PhoneNumber</td>
        <td>@applicationUser.StreetAddress</td>
        <td>@applicationUser.City</td>
        <td>
          <a asp-controller="Employer" asp-action="Suspend" asp-route-id="@applicationUser.Id" class="btn btn-primary">Suspend</a>
          <a asp-controller="Employer" asp-action="Delete" asp-route-id="@applicationUser.Id" class="btn btn-danger">Delete</a>
        </td>
      </tr>
    }
  </tbody>
</table>

```

Figure 52: View

e. Home

*** View Home Page:**

```

1 ViewData["title"] = "FPT Job Match ";
2
3 }
4
5 <DOCTYPE html>
6 <html lang="en">
7 <head>
8     <meta charset="UTF-8">
9     <meta name="viewport" content="width=device-width, initial-scale=1.0">
10    <title>@ViewData["title"]</title>
11    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" integrity="sha384-JcBbQ3qW3VvRV0RNo1XZIo1K7o0Y2nVtGMj4QA3gNJSE/JLmMBqGLw+Jc1wQ+rg" crossorigin="anonymous">
12 </head>
13 <body>
14
15 <!-- Banner Section -->
16 <div class="jumbotron jumbotron-fluid text-center bg-dark text-white">
17     <div class="container">
18         <h1 class="display-4">Welcome to our FPT Job Match</h1>
19         <p class="lead">Find your dream job today!</p>
20     </div>
21 </div>
22
23 <!-- Main Content Section -->
24 <div class="container">
25     <div class="row justify-content-center">
26         <div class="col-md-8">
27             <a2 class="text-center mb-4">Latest Job Listings</a2>
28             <div class="card">
29                 <div class="card-body">
30                     <h5 class="card-title">Job Title</h5>
31                     <p class="card-text">Job Description</p>
32                     @if (User.Identity.IsAuthenticated)
33                     {
34                         <a href="/job" class="btn btn-primary">Apply Now</a>
35                     }
36                     else
37                     {
38                         <a href="/Identity/Account/Login" class="btn btn-primary">Apply Now</a>
39                     }
40                 </div>
41             </div>
42         </div>
43     </div>
44 </div>

```

Figure 53: View 1

```

46 <!-- Footer Section -->
47 <footer class="bg-dark text-white text-center py-3">
48   <!-- Copy, 2024 Job Portal. All rights reserved. -->
49   </footer>
50
51 <!-- Bootstrap JS -->
52 <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdzHphMSS555CtUpj/zy4c+DKgnaf730W4NE+Ib0VUe+OrxXhF3" crossorigin="anonymous"></script>
53 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js" integrity="sha384-B8gt1j307JH4AgT5D400yU00hCkx5L42/daaqr446R55j1w-3J8B8dLDM" crossorigin="anonymous"></script>
54
55 </body>
56 </html>
57

```

Figure 54: View 2

```

1  @{
2      ViewData["Title"] = "Privacy Policy";
3  }
4  <h1>@ViewData["Title"]</h1>
5
6  <p>Use this page to detail your site's privacy policy.</p>
7

```

Figure 55: View 3

f. Job

* Create Job Listing:

```

@model FPT_JOBPORTAL.JobModel

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="NameJob" class="control-label"></label>
                <input asp-for="NameJob" class="form-control" />
                <span asp-validation-for="NameJob" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="CategoryId" class="control-label"></label>
                <select asp-for="CategoryId" class="form-control" asp-items="ViewBag.CategoryId"> </select>
            </div>
            <div class="form-group">
                <label asp-for="DescriptionJob" class="control-label"></label>
                <input asp-for="DescriptionJob" class="form-control" />
                <span asp-validation-for="DescriptionJob" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Company" class="control-label"></label>
                <input asp-for="Company" class="form-control" />
                <span asp-validation-for="Company" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="DatePost" class="control-label"></label>
                <input asp-for="DatePost" class="form-control" />
                <span asp-validation-for="DatePost" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Salary" class="control-label"></label>
                <input asp-for="Salary" class="form-control" />
                <span asp-validation-for="Salary" class="text-danger"></span>
            </div>
        </form>
    </div>
</div>

```

Figure 56: Create 1

```

<div class="form-group">
  <label asp-for="Salary" class="control-label"></label>
  <input asp-for="Salary" class="form-control" />
  <span asp-validation-for="Salary" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="Requirement" class="control-label"></label>
  <input asp-for="Requirement" class="form-control" />
  <span asp-validation-for="Requirement" class="text-danger"></span>
</div>
<div class="form-group">
  <input type="submit" value="Create" class="btn btn-primary" />
</div>
</form>
</div>
</div>
<div>
  <a asp-action="Index" class="btn btn-danger">Back to List</a>
</div>
@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

Figure 57: Create 2

* Delete Job Listing:


```
@model FPT_JOBPORTAL.JobModel

@{
    ViewData["Title"] = "Delete";
}

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>JobModel</h4>
    <hr />
    <dl class="row">
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.NameJob)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.NameJob)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.DescriptionJob)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.DescriptionJob)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Company)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Company)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.DatePost)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.DatePost)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Salary)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Salary)
        </dd>
        <dt class = "col-sm-2">
```

Figure 58: Delete 1


```

<dd class = "col-sm-10">
    @Html.DisplayFor(model => model.Company)
</dd>
<dt class = "col-sm-2">
    @Html.DisplayNameFor(model => model.DatePost)
</dt>
<dd class = "col-sm-10">
    @Html.DisplayFor(model => model.DatePost)
</dd>
<dt class = "col-sm-2">
    @Html.DisplayNameFor(model => model.Salary)
</dt>
<dd class = "col-sm-10">
    @Html.DisplayFor(model => model.Salary)
</dd>
<dt class = "col-sm-2">
    @Html.DisplayNameFor(model => model.Requirement)
</dt>
<dd class = "col-sm-10">
    @Html.DisplayFor(model => model.Requirement)
</dd>
<dt class = "col-sm-2">
    @Html.DisplayNameFor(model => model.Category)
</dt>
<dd class = "col-sm-10">
    @Html.DisplayFor(model => model.Category.Id)
</dd>
</dl>

<form asp-action="Delete">
    <input type="hidden" asp-for="Id" />
    <input type="submit" value="Delete" class="btn btn-danger" /> |
    <a asp-action="Index" class="btn btn-danger">Back to List</a>
</form>
</div>

```

Figure 59: Delete 2

* Job Listing Details:

```
@{
    ViewData["Title"] = "Details";
}

<h1>Details</h1>

<div>
    <h4>JobModel</h4>
    <hr />
    <dl class="row">
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.NameJob)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.NameJob)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.DescriptionJob)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.DescriptionJob)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Company)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Company)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.DatePost)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.DatePost)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Salary)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Salary)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Requirement)
        </dt>
    </dl>

```

Figure 60: Details 1

```

<dd class = "col-sm-10">
    @Html.DisplayFor(model => model.Company)
</dd>
<dt class = "col-sm-2">
    @Html.DisplayNameFor(model => model.DatePost)
</dt>
<dd class = "col-sm-10">
    @Html.DisplayFor(model => model.DatePost)
</dd>
<dt class = "col-sm-2">
    @Html.DisplayNameFor(model => model.Salary)
</dt>
<dd class = "col-sm-10">
    @Html.DisplayFor(model => model.Salary)
</dd>
<dt class = "col-sm-2">
    @Html.DisplayNameFor(model => model.Requirement)
</dt>
<dd class = "col-sm-10">
    @Html.DisplayFor(model => model.Requirement)
</dd>
<dt class = "col-sm-2">
    @Html.DisplayNameFor(model => model.Category)
</dt>
<dd class = "col-sm-10">
    @Html.DisplayFor(model => model.Category.Name)
</dd>
</dl>
</div>
<div>
*   <a asp-action="Edit" asp-route-id="@Model?.Id">Edit</a> |
    *@   <a asp-action="Index" class="btn btn-danger">Back to List</a>
</div>

```

Figure 61: Details 2

* Edit Job Listing:

```
@{
    ViewData["Title"] = "Edit";
}

<h1>Edit</h1>

<h4>JobModel</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Id" />
            <div class="form-group">
                <label asp-for="NameJob" class="control-label"></label>
                <input asp-for="NameJob" class="form-control" />
                <span asp-validation-for="NameJob" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="CategoryId" class="control-label"></label>
                <select asp-for="CategoryId" class="form-control" asp-items="ViewBag.CategoryId"></select>
                <span asp-validation-for="CategoryId" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="DescriptionJob" class="control-label"></label>
                <input asp-for="DescriptionJob" class="form-control" />
                <span asp-validation-for="DescriptionJob" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Company" class="control-label"></label>
                <input asp-for="Company" class="form-control" />
                <span asp-validation-for="Company" class="text-danger"></span>
            </div>
            @* <div class="form-group">
                <label asp-for="DatePost" class="control-label"></label>
                <input asp-for="DatePost" class="form-control" />
                <span asp-validation-for="DatePost" class="text-danger"></span>
            </div> *@
            <div class="form-group">
                <label asp-for="Salary" class="control-label"></label>
                <input asp-for="Salary" class="form-control" />
                <span asp-validation-for="Salary" class="text-danger"></span>
            </div>
        </form>
    </div>
</div>
```

Figure 62: Edit 1


```

<div class="form-group">
  <label asp-for="Salary" class="control-label"></label>
  <input asp-for="Salary" class="form-control" />
  <span asp-validation-for="Salary" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="Requirement" class="control-label"></label>
  <input asp-for="Requirement" class="form-control" />
  <span asp-validation-for="Requirement" class="text-danger"></span>
</div>
<div class="form-group">
  <input type="submit" value="Save" class="btn btn-primary" />
</div>
</form>
</div>
</div>

<div>
  <a asp-action="Index" class="btn btn-danger">Back to List</a>
</div>

@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

Figure 63: Edit 2

* Index of Job Listing:


```
@model IEnumerable<FPT_JOBPORTAL.JobModel>

@{
    ViewData["Title"] = "Job";
}

<h1>Job</h1>
@if (User.IsInRole("Employer"))
{
    <p>
        <a asp-action="Create" class="btn btn-primary">Create New</a>
    </p>
}
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.NameJob)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.DescriptionJob)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Category)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Company)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.DatePost)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Salary)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Requirement)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
```

Figure 64: Index 1

```
@foreach (var item in Model)
{
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.NameJob)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.DescriptionJob)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Category.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Company)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.DatePost)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Salary)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Requirement)
        </td>
        @if (User.IsInRole("Employer"))
        {
            <td>
                <a asp-action="Details" asp-route-id="@item.Id" class="btn btn-primary">Details</a>
                <a asp-action="Delete" asp-route-id="@item.Id" class="btn btn-danger">Delete</a>
            </td>
        }
        @if (User.IsInRole("Admin"))
        {
            <td>
                <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-primary">Edit</a>
                <a asp-action="Delete" asp-route-id="@item.Id" class="btn btn-danger">Delete</a>
            </td>
        }
    </tr>
}
```

Figure 65: Index 2

g. JobSeeker

* Delete Job Seeker:

```
@model ApplicationUser

<h1>Delete Job Seeker</h1>

<div>
  <h4>Are you sure you want to delete this Job Seeker?</h4>
  <hr />
  <dl class="row">
    <dt class="col-sm-2">
      Email
    </dt>
    <dd class="col-sm-10">
      @Model.Email
    </dd>
  </dl>
  <form asp-action="Delete" method="post">
    <input type="hidden" asp-for="Id" />
    <div class="form-group">
      <input type="submit" value="Delete" class="btn btn-danger" />
      <a asp-action="Index" class="btn btn-secondary">Cancel</a>
    </div>
  </form>
</div>
```

Figure 66: Delete

* Index of Job Seeker:

TỔ CHỨC GIÁO DỤC FPT

```
@model List<ApplicationUser>
<h1>Job Seeker List</h1>
<table class="table table-bordered">
  <thead>
    <tr>
      <th>Full Name</th>
      <th>Email</th>
      <th>Phone Number</th>
      <th>Street Address</th>
      <th>City</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var applicationUser in Model)
    {
      <tr>
        <td>@applicationUser.Name</td>
        <td>@applicationUser.Email</td>
        <td>@applicationUser.PhoneNumber</td>
        <td>@applicationUser.StreetAddress</td>
        <td>@applicationUser.City</td>
        <td>
          <a asp-controller="JobSeeker" asp-action="Suspend" asp-route-id="@applicationUser.Id" class="btn btn-primary">Suspend</a>
          <a asp-controller="JobSeeker" asp-action="Delete" asp-route-id="@applicationUser.Id" class="btn btn-danger">Delete</a>
        </td>
      </tr>
    }
  </tbody>
</table>
```

Figure 67: Index

h. Layout

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] FPTJobMatch</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/ASM2_FPTJobMatch.styles.css" asp-append-version="true" />
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css">
    <link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" />
  </head>
  <body>
    <header>
      <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
        <div class="container-fluid">
          <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">FPTJobMatch</a>
          <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="navbarSupportedCon"
            aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
          </button>
          <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
            <ul class="navbar-nav flex-grow-1">
              <li class="nav-item">
                <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
              </li>
              <li class="nav-item">
                <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
              </li>
              @if (User.IsInRole("Admin"))
              {
                <li class="nav-item">
                  <a class="nav-link text-dark" asp-area="" asp-controller="AppRoles" asp-action="Index">Role</a>
                </li>
                <li class="nav-item dropdown">
                  <a class="nav-link dropdown-toggle text-dark" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">
                    User Managment
                  </a>
                </li>
              }
            </ul>
          </div>
        </div>
      </nav>
    </header>
```

Figure 68: Layout 1


```

<ul class="dropdown-menu">
  <li class="nav-item">
    <a class="dropdown-item" asp-area="" asp-controller="Employer" asp-action="Index">Employer</a>
  </li>
  <li class="nav-item">
    <a class="dropdown-item" asp-area="" asp-controller="JobSeeker" asp-action="Index">Job Seeker</a>
  </li>
</ul>
</li>
}
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="Application" asp-action="Index">Application</a>
</li>
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="Category" asp-action="Index">Category</a>
</li>
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="Job" asp-action="Index">Job</a>
</li>
</ul>
<partial name="_LoginPartial" />
</div>
</nav>
</header>
</div>

```

Figure 69: Layout 2

```

<div class="container">
  <main role="main" class="pb-3">
    @RenderBody()
  </main>
</div>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
</body>
</html>

```

Figure 70: Layout 3

4. Repository
 - a. IRepository


```
namespace FPT_JOBPORTAL.Repository.IRepository
{
    public interface IRepository<T> where T : class
    {
        IEnumerable<T> GetAll(string? includeProperties = null);
        IEnumerable<T> GetAll(Expression<Func<T, bool>> filter, string? includeProperties = null);
        T Get(Expression<Func<T, bool>> filter, string? includeProperties = null);
        void Add(T entity);
        void Delete(T entity);
        void Save();
    }
}
```

Figure 71: IRepository

The `IRepository<T>` interface is a generic interface designed to manage entities of type `T`, where `T` must be a class.

- **GetAll:** This method retrieves all entities of type `T` from the data source. It includes an optional `includeProperties` parameter for specifying related properties to include in the query results. It returns an `IEnumerable<T>`, representing a collection of entities.
- **Overloaded GetAll:** This version of the method includes an additional `filter` parameter, which is an expression for filtering entities based on specific conditions. This allows retrieval of entities that meet certain criteria, with `includeProperties` remaining optional.
- **Get:** This method retrieves a single entity of type `T` based on a specified filter expression. It returns a single instance of `T` or `null` if no entity matches the filter.
- **Add:** This method adds a new entity of type `T` to the data source.
- **Delete:** This method removes an existing entity of type `T` from the data source.
- **Save:** This method commits any changes made to the data source, ensuring that additions or deletions are saved permanently.

b. IUnitOfWork

```
namespace FPT_JOBPORTAL.Repository.IRepository
{
    public interface IUnitOfWork
    {
        IApplicationUserRepository ApplicationUserRepository { get; }
        void Save();
    }
}
```

Figure 72: IUnitOfWork

The `IUnitOfWork` interface includes a property named `ApplicationUserRepository` of type `IApplicationUserRepository`. This property provides access to a specialized repository interface used for managing `ApplicationUser` entities in the data source.

The `Save` method is designed to commit all changes made within the unit of work to the data source in a single transaction. This ensures that either all changes are applied together or none are, thereby maintaining data consistency.

c. IApplicationUserRepository

```
3 references
public interface IApplicationUserRepository : IRepository<ApplicationUser>
{
    1 reference
    void Update(ApplicationUser applicationUser);
}
```

Figure 73: IApplicationUserRepository

The `IApplicationUserRepository` interface extends the `IRepository<ApplicationUser>` interface, inheriting all its methods. This specialization indicates that `IApplicationUserRepository` is specifically designed for managing `ApplicationUser` entities.

`ApplicationUser` is a model class representing a user in the FPT_JOBPORTAL application.

Besides the inherited methods from `IRepository<ApplicationUser>`, the `IApplicationUserRepository` adds an `Update` method. This method is intended to update existing `ApplicationUser` entities within the data source.

d. ApplicationUserRepository

```

namespace FPT_Greenwich.Repository
{
    public class ApplicationUserRepository : Repository<ApplicationUser>, IApplicationUserRepository
    {
        private ApplicationDbContext _db;

        public ApplicationUserRepository(ApplicationDbContext db) : base(db)
        {
            _db = db;
        }

        public void Update(ApplicationUser applicationUser)
        {
            _db.ApplicationUsers.Update(applicationUser);
        }
    }
}

```

Figure 74: ApplicationUserRepository

The `ApplicationUserRepository` class extends the `Repository<ApplicationUser>` class, inheriting its basic data access functionalities. It also implements the `IApplicationUserRepository` interface, which requires the implementation of the `Update` method. The constructor of `ApplicationUserRepository` takes an `ApplicationDbContext` instance, which is used for database operations, and passes it to the base class constructor.

The `Update` method updates an existing `ApplicationUser` entity within the data source. It accesses the `ApplicationUsers` property of the `ApplicationDbContext` and uses the `Update` method to mark the entity as modified.

e. Repository

```

namespace FPT_SUPPORTAL.Repository
{
    3 references
    public class Repository<T> : IRepository<T> where T : class
    {
        private readonly ApplicationDbContext _db;
        internal DbSet<T> dbSet;
        1 reference
        public Repository(ApplicationDbContext db)
        {
            _db = db;
            this.dbSet = _db.Set<T>();
            _db.JobModel.Include(u => u.Category);
        }
        1 reference
        public void Add(T entity)
        {
            dbSet.Add(entity);
        }
        5 references
        public T Get(Expression<Func<T, bool>> predicate, string? includeProperty = null)
        {
            IQueryable<T> query = dbSet;
            query = query.Where(predicate);
            if (!string.IsNullOrEmpty(includeProperty))
            {
                query.Include(includeProperty).ToList();
            }
            return query.FirstOrDefault();
        }
        1 reference
        public IEnumerable<T> GetAll(string? includedProperty = null)
        {
            IQueryable<T> query = dbSet;
            if (!string.IsNullOrEmpty(includedProperty))
            {
            }
        }
    }
}

```

Figure 75: Repository

```

37 public IEnumerable<T> GetAll(string? includedProperty = null)
38 {
39     IQueryable<T> query = dbSet;
40     if (!String.IsNullOrEmpty(includedProperty))
41     {
42         query.Include(includedProperty).ToList();
43     }
44     return query.ToList();
45 }
46
47 public IEnumerable<T> GetAll(Expression<Func<T, bool>> predicate, string? includeProperty = null)
48 {
49     IQueryable<T> query = dbSet;
50     query = query.Where(predicate);
51     if (!String.IsNullOrEmpty(includeProperty))
52     {
53         query.Include(includeProperty).ToList();
54     }
55     return query.ToList();
56 }
57
58 public void Delete(T entity)
59 {
60     dbSet.Remove(entity);
61 }
62
63 public void Save()
64 {
65     _db.SaveChanges();
66 }
67
68 }
69
    
```

The `Repository<T>` class implements the `IRepository<T>` interface, thus providing concrete implementations for its methods. The constructor of the class accepts an `ApplicationDbContext` instance, which is used for accessing the database. It contains an internal `DbSet<T>` field that represents a collection of entities of type `T` in the database.

The `Add` method adds a new entity to the `DbSet`. The `Get` method retrieves a single entity of type `T` based on a specified predicate and an optional include property, using the `Where` method to filter the entities and the `Include` method to eagerly load related entities if the `includeProperty` is supplied. The `GetAll` method retrieves all entities of type `T`, optionally applying an include property and predicate, similar to the `Get` method. The `Delete` method removes an existing entity from the `DbSet`. Finally, the `Save` method persists any changes made to the data source by calling the `SaveChanges` method on the `ApplicationDbContext`.

f. UnitOfWork


```
namespace FPT_WebApp.Repository
{
    public class UnitOfWork : IUnitOfWork
    {
        private ApplicationDbContext _db;

        public IApplicationUserRepository ApplicationUserRepository { get; private set; }

        public UnitOfWork(ApplicationDbContext db)
        {
            _db = db;
            ApplicationUserRepository = new ApplicationUserRepository(_db);
        }

        public void Save()
        {
            _db.SaveChanges();
        }
    }
}
```

Figure 76: UnitOfWork

The `UnitOfWork` class implements the `IUnitOfWork` interface, providing the specific implementation for its members. It has a constructor that takes an `ApplicationDbContext` instance, which is used for database interactions. The class includes a property, `ApplicationUserRepository`, of type `IApplicationUserRepository`, representing the repository interface for the `ApplicationUser` entity. This repository is initialized with the `ApplicationDbContext` provided through the constructor.

The `Save` method commits any changes made within the unit of work by calling the `SaveChanges` method on the `ApplicationDbContext`, thereby updating the data source with the modifications.

5. Data

```

8 public class ApplicationDbContext : IdentityDbContext<IdentityUser>
9 {
10     public DbSet<ApplicationUser> ApplicationUsers { get; set; }
11
12     public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
13     {
14     }
15
16
17     public DbSet<CategoryModel> CategoryModel { get; set; }
18     public DbSet<JobModel> JobModel { get; set; }
19     public DbSet<Application> Application { get; set; }
20     protected override void OnModelCreating(ModelBuilder modelBuilder)
21     {
22         base.OnModelCreating(modelBuilder);
23         modelBuilder.Entity<CategoryModel>().HasData(
24             new CategoryModel { Id = 1, Name = "Back-End Developer", Description = "A Back End Developer is responsible for server-side application logic and integration of the work front-end developers do." },
25             new CategoryModel { Id = 2, Name = "Front-End Developer", Description = "A Front End Developer is focused on the user interface and user experience of a website or application." },
26             new CategoryModel { Id = 3, Name = "Full Stack Developer", Description = "A Full Stack Developer is capable of working on both the front-end and back-end portions of an application." },
27             new CategoryModel { Id = 4, Name = "Mobile Apps Developer", Description = "A Mobile Apps Developer is specialized in creating applications for mobile devices, such as smartphones and tablets." },
28             new CategoryModel { Id = 5, Name = "Product Manager", Description = "A Product Manager is responsible for the strategy, roadmap, and feature definition of a product." },
29             new CategoryModel { Id = 6, Name = "Python Developer", Description = "A Python Developer uses the Python programming language for developing software solutions." },
30             new CategoryModel { Id = 7, Name = "System Administrator", Description = "A System Administrator manages and maintains the operations of computer systems and networks." },
31             new CategoryModel { Id = 8, Name = "Tester", Description = "A Tester tests software to ensure it meets requirements and is free of bugs and defects." },
32             new CategoryModel { Id = 9, Name = "C++ Developer", Description = "A C++ Developer uses the C++ programming language for developing software solutions." },
33             new CategoryModel { Id = 10, Name = "Software Architect", Description = "A Software Architect designs and creates the overall structure of a software system." }
34         );
35         modelBuilder.Entity<JobModel>().HasData(
36             new JobModel
37             {
38                 Id = 1,
39                 NameJob = "Mobile Apps Developer",

```

Figure 77: Data 1

```

35 modelBuilder.Entity<JobModel>().HasData(
36     new JobModel
37     {
38         Id = 1,
39         NameJob = "Mobile Apps Developer",
40         CategoryId = 4,
41         DescriptionJob = "Join our mobile development team to create innovative and user-friendly mobile applications for iOS and Android platforms.",
42         Company = "MegaTechnology",
43         DatePost = DateTime.UtcNow.Date.AddDays(-7),
44         Salary = 95000,
45         Requirement = "Experience in mobile app development using Swift or Kotlin, familiarity with mobile UI/UX principles, ability to adapt to new technologies.",
46     },
47     new JobModel
48     {
49         Id = 2,
50         NameJob = "Full Stack Developer",
51         CategoryId = 3,
52         DescriptionJob = "We're seeking a Full Stack Developer to work on both front-end and back-end components of our applications.",
53         Company = "MegaTechnology",
54         DatePost = DateTime.UtcNow.Date.AddDays(-7),
55         Salary = 95000,
56         Requirement = "Proficiency in both front-end and back-end technologies, experience with frameworks such as React and Node.js, ability to work in a fast-paced environment.",
57     },
58     new JobModel
59     {
60         Id = 3,
61         NameJob = "Front End Developer Intern",
62         CategoryId = 2,
63         DescriptionJob = "Exciting opportunity for a Front End Developer Intern to gain hands-on experience in building user interfaces and enhancing user experiences.",
64         Company = "MegaTechnology",
65         DatePost = DateTime.UtcNow.Date.AddDays(-7),
66         Salary = 25000,
67         Requirement = "Basic understanding of HTML, CSS, and JavaScript, eager to learn and contribute to front-end development projects.",
68     },
69     new JobModel
70     {
71         Id = 4,
72         NameJob = "Senior Back End Developer",
73         CategoryId = 1,
74         DescriptionJob = "Join our team as a Senior Back End Developer to lead server-side application logic and integration efforts.",
75         Company = "MegaTechnology",
76         DatePost = DateTime.UtcNow.Date.AddDays(-7),

```

Figure 78: Data 2

```

58 {
59     Id = 3,
60     NameJob = "Front End Developer Intern",
61     CategoryId = 2,
62     DescriptionJob = "Exciting opportunity For a Front End Developer Intern to gain hands-on experience in building user interfaces and enhancing user experiences.",
63     Company = "MegaTechnology",
64     DatePost = DateTime.UtcNow.Date.AddDays(-7),
65     Salary = 25000,
66     Requirement = "Basic understanding of HTML, CSS, and JavaScript, eager to learn and contribute to front-end development projects.",
67     new JobModel
68 {
69     Id = 4,
70     NameJob = "Senior Back End Developer",
71     CategoryId = 1,
72     DescriptionJob = "Join our team as a Senior Back End Developer to lead server-side application logic and integration efforts.",
73     Company = "MegaTechnology",
74     DatePost = DateTime.UtcNow.Date.AddDays(-7),
75     Salary = 95000,
76     Requirement = "5+ years of experience in back-end development, proficiency in relevant technologies such as Node.js, Python, or Java, strong understanding of databases and API design.",
77     new JobModel
78 {
79     Id = 5,
80     NameJob = "Product Manager",
81     CategoryId = 5,
82     DescriptionJob = "We're looking for a Product Manager to drive the strategy and development of our product offerings.",
83     Company = "MegaTechnology",
84     DatePost = DateTime.UtcNow.Date.AddDays(-7),
85     Salary = 110000,
86     Requirement = "Strong leadership and communication skills, experience in product management, ability to prioritize and manage multiple projects.",
87     };
88 }
89 }
90 }

```

Figure 79: Data 3

5. Areas (Login, Register)

a. Login View

```

4 04
5 }
6 ViewData["Title"] = "Log in";
7
8 <h1>@ViewData["Title"]</h1>
9 <div class="row">
10 <div class="col-md-4">
11 <section>
12 <form id="account" method="post">
13 <h2>Use a local account to log in.</h2>
14 <hr />
15 <div asp-validation-summary="ModelOnly" class="text-danger" role="alert"></div>
16 <div class="form-floating mb-3">
17 <input asp-for="Input.Email" class="form-control" autocomplete="username" aria-required="true" placeholder="name@example.com" />
18 <label asp-for="Input.Email" class="form-label">Email</label>
19 <span asp-validation-for="Input.Email" class="text-danger"></span>
20 </div>
21 <div class="form-floating mb-3">
22 <input asp-for="Input.Password" class="form-control" autocomplete="current-password" aria-required="true" placeholder="password" />
23 <label asp-for="Input.Password" class="form-label">Password</label>
24 <span asp-validation-for="Input.Password" class="text-danger"></span>
25 </div>
26 <div class="checkbox mb-3">
27 <label asp-for="Input.RememberMe" class="form-label">
28 <input class="form-check-input" asp-for="Input.RememberMe" />
29 @Html.DisplayNameFor(m => m.Input.RememberMe)
30 </label>
31 </div>
32 <div>
33 <button id="login-submit" type="submit" class="w-100 btn btn-lg btn-primary">Log in</button>
34 </div>
35 <div>
36 <p>
37 <a id="forgot-password" asp-page="./ForgotPassword">Forgot your password?</a>
38 </p>
39 <p>
40 <a asp-page="./Register" asp-route-returnUrl="@Model.ReturnUrl">Register as a new user</a>
41 </p>
42 <p>
43 <a id="resend-confirmation" asp-page="./ResendEmailConfirmation">Resend email confirmation</a>
44 </p>
45 </div>
46 </form>
47 </div>
48 </div>
49 </div>
50
51 @section Scripts {

```

Figure 80: Login View

b. Login Logic Code

```

4
5 using System;
6 using System.Collections.Generic;
7 using System.ComponentModel.DataAnnotations;
8 using System.Linq;
9 using System.Threading.Tasks;
10 using Microsoft.AspNetCore.Authorization;
11 using Microsoft.AspNetCore.Authentication;
12 using Microsoft.AspNetCore.Identity;
13 using Microsoft.AspNetCore.Identity.UI.Services;
14 using Microsoft.AspNetCore.Mvc;
15 using Microsoft.AspNetCore.Mvc.RazorPages;
16 using Microsoft.Extensions.Logging;
17
18 namespace FPT_JOBPORTAL.Areas.Identity.Pages.Account
19 {
20     public class LoginModel : PageModel
21     {
22         private readonly SignInManager<IdentityUser> _signInManager;
23         private readonly ILogger<LoginModel> _logger;
24
25         public LoginModel(SignInManager<IdentityUser> signInManager, ILogger<LoginModel> logger)
26         {
27             _signInManager = signInManager;
28             _logger = logger;
29         }
30
31         /// <summary>
32         /// This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
33         /// directly from your code. This API may change or be removed in future releases.
34         /// </summary>
35         [BindProperty]
36         public InputModel Input { get; set; }
37
38         /// <summary>
39         /// This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
40         /// directly from your code. This API may change or be removed in future releases.
41         /// </summary>

```

Figure 81: Login logic code 1


```

87 public async Task OnGetAsync(string returnUrl = null)
88 {
89     if (!string.IsNullOrEmpty(ErrorMessage))
90     {
91         ModelState.AddModelError(string.Empty, ErrorMessage);
92     }
93
94     returnUrl ??= Url.Content("~/");
95
96     // Clear the existing external cookie to ensure a clean login process
97     await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);
98
99     ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
100
101     ReturnUrl = returnUrl;
102 }
103
104 0 references
105 public async Task<IActionResult> OnPostAsync(string returnUrl = null)
106 {
107     returnUrl ??= Url.Content("~/");
108
109     ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
110
111     if (ModelState.IsValid)
112     {
113         // This doesn't count login failures towards account lockout
114         // To enable password failures to trigger account lockout, set lockoutOnFailure: true
115         var result = await _signInManager.PasswordSignInAsync(Input.Email, Input.Password, Input.RememberMe, lockoutOnFailure: false);
116         if (result.Succeeded)
117         {
118             _logger.LogInformation("User logged in.");
119             return LocalRedirect(returnUrl);
120         }
121         if (result.RequiresTwoFactor)
122         {
123             return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl, RememberMe = Input.RememberMe });
124         }
125         if (result.IsLockedOut)
126         {
127             _logger.LogWarning("User account locked out.");
128             return RedirectToPage("./Lockout");
129         }
130     }
131
132     // If we got this far, something failed, redisplay form
133     return Page();
134 }
135
136 // If we got this far, something failed, redisplay form
137 return Page();
138 }
139 }
140 }
141

```

Figure 82: Login logic code 2

```

128 }
129 else
130 {
131     ModelState.AddModelError(string.Empty, "Invalid login attempt.");
132     return Page();
133 }
134 }
135
136 // If we got this far, something failed, redisplay form
137 return Page();
138 }
139 }
140 }
141

```

Figure 83: Login logic code

The `LoginModel` class is a Razor Page model responsible for handling user login. It has two private fields injected via the constructor: `SignInManager` and `ILogger`, which are used for authentication and logging purposes.

The `LoginModel` class features a nested `InputModel` class representing the login form's input fields. This includes properties like Email, Password, and RememberMe, along with data annotations for validation.

The `OnGetAsync` method is triggered when the login page is accessed via an HTTP GET request. It checks for any error messages and adds them to the model state, clears existing external cookies for a fresh login session, retrieves available authentication schemes for external logins, and sets the `ReturnUrl` property.

The `OnPostAsync` method handles the form submission via an HTTP POST request. It validates the form data and uses `SignInManager` to attempt user authentication. If successful, it logs the user in and redirects them to the `returnUrl`. If two-factor authentication is needed, it redirects to the `LoginWith2fa` page. If the user's account is locked, it redirects to the `Lockout` page. If the login attempt fails, it adds an error message to the model state and redisplay the login form.

The `returnUrl` parameter determines where to redirect users after a successful login or if they cancel the login process. The `InputModel` class includes a `RememberMe` property, allowing users to choose whether to save their login credentials. The `ExternalLogins` property lists available external authentication options retrieved from `SignInManager`, which are displayed on the login form. The `ErrorMessage` property shows any error messages from previous login attempts.

c. Register View

```

1  @page
2  @model RegisterModel
3  @{
4      ViewData["Title"] = "Register";
5  }
6
7  <h1>@ViewData["Title"]</h1>
8
9  <div class="row">
10     <div class="col-md-4">
11         <form id="registerForm" asp-route-returnUrl="@Model.ReturnUrl" method="post">
12             <h2>Create a new account.</h2>
13             <hr />
14             <div asp-validation-summary="ModelOnly" class="text-danger" role="alert"></div>
15             <div class="form-floating mb-3">
16                 <input asp-for="Input.Email" class="form-control" autocomplete="username" aria-required="true" placeholder="name@example.com" />
17                 <label asp-for="Input.Email" class="ms-2 text-muted">Email</label>
18                 <span asp-validation-for="Input.Email" class="text-danger"></span>
19             </div>
20             <div class="form-floating mb-3">
21                 <input asp-for="Input.Name" class="form-control" placeholder="name" />
22                 <label asp-for="Input.Name" class="ms-2 text-muted">Full Name</label>
23                 <span asp-validation-for="Input.Name" class="text-danger"></span>
24             </div>
25             <div class="form-floating mb-3">
26                 <input asp-for="Input.PhoneNumber" class="form-control" placeholder="phone number" />
27                 <label asp-for="Input.PhoneNumber" class="ms-2 text-muted">Phone Number</label>
28                 <span asp-validation-for="Input.PhoneNumber" class="text-danger"></span>
29             </div>
30             <div class="form-floating mb-3">
31                 <input asp-for="Input.Password" class="form-control" autocomplete="new-password" aria-required="true" placeholder="password" />
32                 <label asp-for="Input.Password" class="ms-2 text-muted">Password</label>
33                 <span asp-validation-for="Input.Password" class="text-danger"></span>
34             </div>
35             <div class="form-floating mb-3">
36                 <input asp-for="Input.ConfirmPassword" class="form-control" autocomplete="new-password" aria-required="true" placeholder="password" />
37                 <label asp-for="Input.ConfirmPassword" class="ms-2 text-muted">Confirm Password</label>
38                 <span asp-validation-for="Input.ConfirmPassword" class="text-danger"></span>
39             </div>
40             <div class="form-floating mb-3">
41                 <input asp-for="Input.StreetAddress" class="form-control" placeholder="address" />
42                 <label asp-for="Input.StreetAddress" class="ms-2 text-muted">Street Address</label>
43                 <span asp-validation-for="Input.StreetAddress" class="text-danger"></span>
44             </div>
45             <div class="form-floating mb-3">
46                 <input asp-for="Input.City" class="form-control" placeholder="city" />
47                 <label asp-for="Input.City" class="ms-2 text-muted">City</label>

```

Figure 84: Register View 1

```

41     <input asp-for="Input.StreetAddress" class="form-control" placeholder="address" />
42     <label asp-for="Input.StreetAddress" class="ms-2 text-muted">Street Address</label>
43     <span asp-validation-for="Input.StreetAddress" class="text-danger"></span>
44 </div>
45 <div class="form-floating mb-3">
46     <input asp-for="Input.City" class="form-control" placeholder="city" />
47     <label asp-for="Input.City" class="ms-2 text-muted">City</label>
48     <span asp-validation-for="Input.City" class="text-danger"></span>
49 </div>
50 <div class="form-floating mb-3">
51     <select asp-for="Input.Role" asp-items="Model.Input.RoleList" class="form-select" aria-required="true">
52         <option value="">Select Role</option>
53         @* @foreach (var role in Model.Input.RoleList)
54         {
55             if (role.Value != "Admin")
56             {
57                 <option value="@role.Value">@role.Text</option>
58             }
59         } *@
60     </select>
61     <span asp-validation-for="Input.Role" class="text-danger"></span>
62 </div>
63 <button id="registerSubmit" type="submit" class="w-100 btn btn-lg btn-primary">Register</button>
64 </form>
65 </div>
66 </div>
67
68 @section Scripts {
69     <partial name="_ValidationScriptsPartial" />
70 }
71

```

Figure 85: Register View 2

d. Register Logic Code

```

4
5 using System;
6 using System.Collections.Generic;
7 using System.ComponentModel.DataAnnotations;
8 using System.Linq;
9 using System.Text;
10 using System.Text.Encodings.Web;
11 using System.Threading;
12 using System.Threading.Tasks;
13 using FPT_JOBPORTAL.Models;
14 using Microsoft.AspNetCore.Authentication;
15 using Microsoft.AspNetCore.Authorization;
16 using Microsoft.AspNetCore.Identity;
17 using Microsoft.AspNetCore.Identity.UI.Services;
18 using Microsoft.AspNetCore.Mvc;
19 using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
20 using Microsoft.AspNetCore.Mvc.RazorPages;
21 using Microsoft.AspNetCore.Mvc.Rendering;
22 using Microsoft.AspNetCore.WebUtilities;
23 using Microsoft.Extensions.Logging;
24
25 namespace FPT_JOBPORTAL.Areas.Identity.Pages.Account
26 {
27     8 references
28     public class RegisterModel : PageModel
29     {
30         private readonly SignInManager<IdentityUser> _signInManager;
31         private readonly UserManager<IdentityUser> _userManager;
32         private readonly IUserStore<IdentityUser> _userStore;
33         private readonly IUserEmailStore<IdentityUser> _emailStore;
34         private readonly ILogger<RegisterModel> _logger;
35         private readonly IEmailSender _emailSender;
36         private readonly RoleManager<IdentityRole> _roleManager;
37
38         0 references
39         public RegisterModel(
40             UserManager<IdentityUser> userManager,
41             IUserStore<IdentityUser> userStore,
42             SignInManager<IdentityUser> signInManager,
43             ILogger<RegisterModel> logger,
44             IEmailSender emailSender,
45             RoleManager<IdentityRole> roleManager)
46         {
47             _userManager = userManager;
48             _userStore = userStore;

```

Figure 86: Register logic code 1


```

84      /// </summary>
85      [Required]
86      [EmailAddress]
87      [Display(Name = "Email")]
88      7 references
89      public string Email { get; set; }
90
91      /// <summary>
92      ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
93      ///     directly from your code. This API may change or be removed in future releases.
94      /// </summary>
95      [Required]
96      [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]
97      [DataType(DataType.Password)]
98      [Display(Name = "Password")]
99      4 references
100     public string Password { get; set; }
101
102     /// <summary>
103     ///     This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used
104     ///     directly from your code. This API may change or be removed in future releases.
105     /// </summary>
106     [DataType(DataType.Password)]
107     [Display(Name = "Confirm password")]
108     [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
109     3 references
110     public string ConfirmPassword { get; set; }
111
112     [Required]
113     6 references
114     public string? Role { get; set; }
115
116     [ValidateNever]
117     2 references
118     public IEnumerable<SelectListItem> RoleList { get; set; }
119     [Required]
120     4 references
121     public string Name { get; set; }
122     - references
123     public string? StreetAddress { get; set; }
124     4 references
125     public string? City { get; set; }
126     - references
127     public string? PhoneNumber { get; set; }
128 }

```

Figure 87: Register logic code 2

```

122 public async Task OnGetAsync(string returnUrl = null)
123 {
124     Input = new()
125     {
126         RoleList = _roleManager.Roles.Where(r => r.Name != "Admin").Select(x => x.Name).Select(i => new SelectListItem
127         {
128             Text = i,
129             Value = i
130         });
131     };
132
133     returnUrl = returnUrl;
134     ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
135 }
136
137
138 public async Task<IActionResult> OnPostAsync(string returnUrl = null)
139 {
140     returnUrl ??= Url.Content("~/");
141     ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
142     if (ModelState.IsValid)
143     {
144         var user = CreateUser();
145
146         await _userStore.SetUserNameAsync(user, Input.Email, CancellationToken.None);
147         await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);
148         user.StreetAddress = Input.StreetAddress;
149         user.City = Input.City;
150         user.Name = Input.Name;
151         user.PhoneNumber = Input.PhoneNumber;
152         if (Input.Role != null)
153         {
154             user.Role = Input.Role;
155         }
156         else
157         {
158             user.Role = "Job Seeker";
159         }
160         var result = await _userManager.CreateAsync(user, Input.Password);
161         if (result.Succeeded)
162     }

```

Figure 88: Register logic code 3


```

160 var result = await _userManager.CreateAsync(user, Input.Password);
161
162 if (result.Succeeded)
163 {
164     _logger.LogInformation("User created a new account with password.");
165
166     if (!String.IsNullOrEmpty(Input.Role))
167     {
168         await _userManager.AddToRoleAsync(user, Input.Role);
169     }
170     else
171     {
172         await _userManager.AddToRoleAsync(user, "Job Seeker");
173     }
174     var userId = await _userManager.GetUserIdAsync(user);
175     var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
176     code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
177     var callbackUrl = Url.Page(
178         "/Account/ConfirmEmail",
179         pageHandler: null,
180         values: new { area = "Identity", userId = userId, code = code, returnUrl = returnUrl },
181         protocol: Request.Scheme);
182
183     await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
184         $"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");
185
186     if (_userManager.Options.SignIn.RequireConfirmedAccount)
187     {
188         return RedirectToPage("RegisterConfirmation", new { email = Input.Email, returnUrl = returnUrl });
189     }
190     else
191     {
192         await _signInManager.SignInAsync(user, isPersistent: false);
193         return LocalRedirect(returnUrl);
194     }
195 }
196 foreach (var error in result.Errors)
197 {
198     ModelState.AddModelError(string.Empty, error.Description);
199 }
200
201

```

Figure 89: Register logic code 4

```

199
200
201
202 // If we got this far, something failed, redisplay form
203 return Page();
204
205
206 private ApplicationUser CreateUser()
207 {
208     try
209     {
210         return Activator.CreateInstance<ApplicationUser>();
211     }
212     catch
213     {
214         throw new InvalidOperationException($"Can't create an instance of '{nameof(IdentityUser)}'. " +
215             $"Ensure that '{nameof(IdentityUser)}' is not an abstract class and has a parameterless constructor, or alternatively " +
216             $"override the register page in /Areas/Identity/Pages/Account/Register.cshtml");
217     }
218 }
219
220 private IUserEmailStore<IdentityUser> GetEmailStore()
221 {
222     if (!_userManager.SupportsUserEmail)
223     {
224         throw new NotSupportedException("The default UI requires a user store with email support.");
225     }
226     return (IUserEmailStore<IdentityUser>)_userStore;
227 }
228
229
230

```

Figure 90: Register logic code 5

The RegisterModel class is a Razor Page model responsible for handling user registration. It includes several private fields injected through the constructor, such as instances of `SignInManager`, `userManager`, `IUserStore`, `IUserEmailStore`, `ILogger`, `IEmailSender`, and `RoleManager`. These dependencies support user management, authentication, logging, email communication, and role management.

Inside RegisterModel, there is a nested class called InputModel, which represents the input fields on the registration form. This class includes properties like Email, Password, ConfirmPassword, Role, Name, StreetAddress, City, and PhoneNumber, along with data annotations for validation purposes.

The `OnGetAsync` method is triggered when the registration page is accessed via an HTTP GET request. It initializes the `Input` property with an `InputModel` instance, retrieves available authentication schemes for external logins, and populates the `RoleList` property with selectable roles for the registration form.

The OnPostAsync method handles the submission of the registration form via an HTTP POST request. It validates the form data, creates a new user with the specified properties using UserManager, and assigns any provided roles to the user. It also generates an email confirmation token and sends a confirmation email. If user creation is successful, it signs in the user and redirects them to the specified returnUrl.

The CreateUser method is a helper that creates an ApplicationUser instance for a new user during registration. The GetEmailStore method retrieves the email store associated with UserManager, providing the necessary email functionality for the registration process.

III. Final screenshots of the application:

1. Home

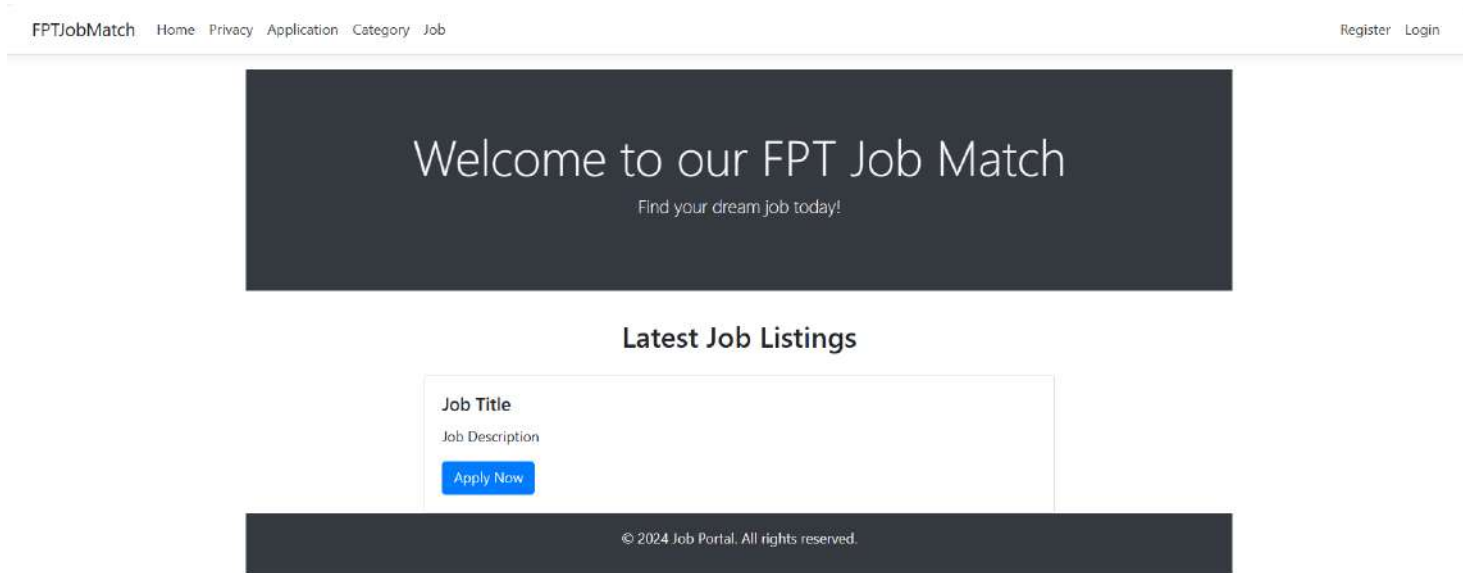


Figure 91: Home

The webpage titled “FPTJobMatch” serves as a job portal where users can search for and apply to job opportunities. The main navigation bar includes options such as Home, Privacy, Application, Category, and Job. Users can register or log in from the top right corner of the page. A welcoming banner greets visitors with the message, “Welcome to our FPT Job Match” and encourages them to “Find your dream job today!” Below this banner, there is a section labeled “Latest Job Listings” with placeholders for job titles and descriptions, along with an “Apply Now” button for users to submit their applications. The design of the webpage aims to offer a user-friendly experience for job seekers.

2. Log in and Register

a. Log in

Log in

Use a local account to log in.

Email
admin@gmail.com

Password
.....

☐ Remember me?

Log in

[Forgot your password?](#)

[Register as a new user](#)

[Resend email confirmation](#)

Figure 92: Login

The page prompts users to log in with a local account, providing fields for entering an email and password. There is also a “Remember me?” checkbox for users who wish to save their login details for future visits. Below the “Log in” button, there are links for users to recover a forgotten password, register as new users, or resend email confirmation. This layout is common in many web applications, offering a simple and user-friendly interface for account access.

Create a new account.

Email

employer3@gmail.com

Full Name

Ngô Trí Tài

Phone Number

0899325547

Password

.....

Confirm Password

.....

Street Address

K01/06 Đỗ Quang

City

Da Nang

Select Role ▼

Register

Figure 93: Register

The form is intended for users to create a new account. It requests personal details such as email, full name, phone number, password, street address, city, and role. The form is designed to be user-friendly, featuring a clean layout with plenty of white space. A “Register” button at the bottom allows users to submit their information.

3. Admin

Roles

Create New

Id	Name	
1b5553e6-42af-49fb-a74d-64be97af8a00	Employer	Edit Delete
6416f141-937e-47b1-90c9-99f54d536082	Job Seeker	Edit Delete

Figure 94: Roles

This administrative panel features a navigation bar with options such as “Home,” “User Role Management,” and “Application Category.” The user greeting “Hello admin@admin.com | Logout” indicates that an admin user is logged in. In the “Roles” section, there is functionality for creating new roles, and the table lists two roles: ‘Job Seeker’ and ‘Employer,’ each with distinct IDs and options to edit or delete. This interface is designed for administrators to manage user roles within a software system, facilitating the assignment and maintenance of access permissions for various user types.

Job Seeker List

Full Name	Email	Phone Number	Street Address	City	
Job Seeker	JobSeeker1@gmail.com	0905614566	Da Nang	Viet Nam	Edit Delete

Figure 95: User Manage 1

This “Job Seeker List” table is part of a job-seeking platform or database management system. It features entries with details including Full Name, Email, Phone Number, Street Address, and City. The table provides administrative options like “Suspend” or “Delete” for each entry, reflecting the system’s ability to manage user statuses within the job-seeking service. This interface is essential for administrators to oversee and maintain the integrity of the user database.

Employer List

Full Name	Email	Phone Number	Street Address	City	
Ngô Trí Tài	employer3@gmail.com	0899325547	K01/06 Đỗ Quang	Da Nang	Edit Delete
Employer	Employer2@gmail.com	123456789	do quang	dn	Edit Delete
Employer	Employer@gmail.com	0899325547	do quang	dn	Edit Delete

Figure 96: User Manage 2

This “Employer List” table is part of a user interface, likely within a job management or HR system. It displays employers with columns for Full Name, Email, Phone Number, Street Address, and City. The table includes options to ‘Suspend’ or ‘Delete’ each employer, indicating administrative control over the accounts. This setup is common in systems designed to organize and manage employer information efficiently, facilitating quick actions like suspending or deleting accounts as needed.

Application

Full Name	Email	Phone	Education	Resume	Status	Job
Ngô Trí Tài	taintgcd210330@fpt.edu.vn	0899325547	Level 6 - University	Accepted	Full Stack Developer	Details
Nguyen Van Tuyen	tuyen3103@gmail.com	0905614566	Level 6 - University	Pending	Senior Back End Developer	Details

Figure 97: Application

The form shows the applicant’s contact information and education level as Bachelor’s, with the application status marked as ‘Accepted’. Features like “View Resume” and “Details” provide functionality for recruiters to review the applicant’s details.

TỔ CHỨC GIÁO DỤC FPT

Danh Sách Danh Mục

[View Pending Approval List](#)
[Create a new category](#)

Name	Description	Status	
Back-End Developer	A Back End Developer is responsible for server-side application logic and integration of the work front-end developers do.	Active	Delete
Front-End Developer	A Front End Developer is focused on the user interface and user experience of a website or application.	Active	Delete
Full Stack Developer	A Full Stack Developer is capable of working on both the front-end and back-end portions of an application.	Active	Delete
Mobile Apps Developer	A Mobile Apps Developer is specialized in creating applications for mobile devices, such as smartphones and tablets.	Active	Delete
Product Manager	A Product Manager is responsible for the strategy, roadmap, and feature definition of a product.	Active	Delete
Python Developer	A Python Developer uses the Python programming language for developing software solutions.	Active	Delete
System Administrator	A System Administrator manages and maintains the operations of computer systems and networks.	Active	Delete
C++ Developer	A C++ Developer uses the C++ programming language for developing software solutions.	Active	Delete
Software Architect	A Software Architect designs and creates the overall structure of a software system.	Active	Delete
Tester	A Tester tests software to ensure it meets requirements and is free of bugs and defects.	Active	Delete

Figure 98: Category in admin

This interface displays a table for managing job roles within a tech organization, listing categories such as Back-End Developer, Front-End Developer, and Mobile Apps Developer, along with their respective responsibilities. For example, the Back-End Developer handles server-side logic and integrates front-end work. The ‘Edit’ and ‘Delete’ buttons for each role allow for modifications or removal of roles as needed.

Jobs

Job Name	Name	Company	Posted Date	Salary	Requirement	
Mobile Apps Developer	Mobile Apps Developer	MegaTechnology	8/23/2024 12:00:00 AM	95000	Experience in mobile app development using Swift or Kotlin, familiarity with mobile UI/UX principles, ability to adapt to new technologies.	Edit Details Delete
Full Stack Developer	Full Stack Developer	MegaTechnology	8/23/2024 12:00:00 AM	95000	Proficiency in both front-end and back-end technologies, experience with frameworks such as React and Node.js, ability to work in a fast-paced environment.	Edit Details Delete
Front End Developer Intern	Front-End Developer	MegaTechnology	8/23/2024 12:00:00 AM	25000	Basic understanding of HTML, CSS, and JavaScript, eager to learn and contribute to front-end development projects.	Edit Details Delete
Senior Back End Developer	Back-End Developer	MegaTechnology	8/23/2024 12:00:00 AM	95000	5+ years of experience in back-end development, proficiency in relevant technologies such as Node.js, Python, or Java, strong understanding of databases and API design.	Edit Details Delete
Product Manager	Product Manager	MegaTechnology	8/23/2024 12:00:00 AM	110000	Strong leadership and communication skills, experience in product management, ability to prioritize and manage multiple projects.	Edit Details Delete
Tester	Tester	Mega Technology	8/31/2024 1:42:45 AM	20000	1	Edit Details Delete

[Create New](#)

Figure 99: Job in admin

The interface presents a table titled “Job” that lists various tech-related positions available at ‘MegaTech’ as of 12/04/2000. The table provides detailed information including job name, description, category, company name, date added, salary, and requirement description. Positions such as Mobile Apps Innovator, Full Stack Developer, and Senior Back End Developer are included, each with specific requirements and salary details. The lack of data in the ‘Est’ column suggests it is either not applicable or yet to be completed. The ‘Edit’ and ‘Delete’ options indicate administrative functions for managing these job listings.

4. Employer

Application

Full Name	Email	Phone	Education	Resume	Status	Job
Ngô Trí Tài	taintgcd210330@fpt.edu.vn	0899325547	Level 6 - University	Accepted	Full Stack Developer	Edit Delete
Nguyen Van Tuyen	tuyen3103@gmail.com	0905614566	Level 6 - University	Pending	Senior Back End Developer	Edit Delete

Figure 100: Application in employer

The interface displays a web page for an application portal, featuring a table with columns for “FullName,” “Email,” “Phone,” “Education,” “Resume,” “Status,” and “Job.” It includes functionalities for viewing the candidate's resume and deleting the application, suggesting it is used by employers or recruiters to track and manage job applications.

Danh Sách Danh Mục

[View Pending Approval List](#)
[Create a new category](#)

Name	Description	Status
Back-End Developer	A Back End Developer is responsible for server-side application logic and integration of the work front-end developers do.	Active Delete
Front-End Developer	A Front End Developer is focused on the user interface and user experience of a website or application.	Active Delete
Full Stack Developer	A Full Stack Developer is capable of working on both the front-end and back-end portions of an application.	Active Delete
Mobile Apps Developer	A Mobile Apps Developer is specialized in creating applications for mobile devices, such as smartphones and tablets.	Active Delete
Product Manager	A Product Manager is responsible for the strategy, roadmap, and feature definition of a product.	Active Delete
Python Developer	A Python Developer uses the Python programming language for developing software solutions.	Active Delete
System Administrator	A System Administrator manages and maintains the operations of computer systems and networks.	Active Delete
C++ Developer	A C++ Developer uses the C++ programming language for developing software solutions.	Active Delete
Software Architect	A Software Architect designs and creates the overall structure of a software system.	Active Delete
Tester	A Tester tests software to ensure it meets requirements and is free of bugs and defects.	Active Delete

Figure 101: Category in employer

The interface titled “Category” shows a table used for managing job roles within an organization. It includes a “Create New” button, allowing users to add new job categories.

Jobs

Job Name	Name	Company	Posted Date	Salary	Requirement	
Mobile Apps Developer	Mobile Apps Developer	MegaTechnology	8/23/2024 12:00:00 AM	95000	Experience in mobile app development using Swift or Kotlin, familiarity with mobile UI/UX principles, ability to adapt to new technologies.	Edit Details Delete
Full Stack Developer	Full Stack Developer	MegaTechnology	8/23/2024 12:00:00 AM	95000	Proficiency in both front-end and back-end technologies, experience with frameworks such as React and Node.js, ability to work in a fast-paced environment.	Edit Details Delete
Front End Developer Intern	Front-End Developer	MegaTechnology	8/23/2024 12:00:00 AM	25000	Basic understanding of HTML, CSS, and JavaScript, eager to learn and contribute to front-end development projects.	Edit Details Delete
Senior Back End Developer	Back-End Developer	MegaTechnology	8/23/2024 12:00:00 AM	95000	5+ years of experience in back-end development, proficiency in relevant technologies such as Node.js, Python, or Java, strong understanding of databases and API design.	Edit Details Delete
Product Manager	Product Manager	MegaTechnology	8/23/2024 12:00:00 AM	110000	Strong leadership and communication skills, experience in product management, ability to prioritize and manage multiple projects.	Edit Details Delete
Tester	Tester	Mega Technology	8/31/2024 1:42:45 AM	20000	1	Edit Details Delete

[Create New](#)

Figure 102: Job in employer

The interface titled “Category” displays a table used for managing job roles within an organization. It includes a “Create New” button, which allows users to add new job categories. Additionally, the ‘Details’ link suggests that more information is available for each role.

5. Job Seeker

Application

Full Name	Email	Phone	Education	Status	Job
Ngô Trí Tài	taintgcd210330@fpt.edu.vn	0899325547	Level 6 - University	Accepted	Full Stack Developer
Nguyen Van Tuyen	tuyen3103@gmail.com	0905614566	Level 6 - University	Pending	Senior Back End Developer

Figure 103: Application in JobSeeker

Each application is linked to a specific job seeker and includes their full name, email, phone number, education level, a link to their resume, and the status of their application. The applications are organized in a table format, allowing users to easily compare and review different submissions.

Create Application

Full Name

Email

Phone

Education

-- Select Education Level --

JobId

Mobile Apps Developer

Resume

Choose File No file chosen

Create

Back to List

Figure 104: Create in Job Seeker

The form titled “Create Application” is intended for users to apply for jobs. It includes fields for entering the applicant's full name, email, phone number, and education level. The field labeled “JobId” is pre-filled with the position “Mobile Apps Developer.” Below these fields, there is an option to upload a resume. At the bottom

TỔ CHỨC GIÁO DỤC FPT

of the form, there are two buttons: “Create” to submit the application and “Back to List” to presumably return to the job listings page.

Access denied

You do not have access to this resource.

Figure 105: Category in Job Seeker

The webpage from "FPTJobMatch" shows a list of job categories within the software development and IT sector. Each category, including roles like Back-End Developer, Front-End Developer, Full Stack Developer, and others, is accompanied by a brief description of the responsibilities for each role. All the listed positions are marked as "Active."

Jobs

Job Name	Name	Company	Posted Date	Salary	Requirement	
Mobile Apps Developer	Mobile Apps Developer	MegaTechnology	8/23/2024 12:00:00 AM	95000	Experience in mobile app development using Swift or Kotlin, familiarity with mobile UI/UX principles, ability to adapt to new technologies.	Edit Details Delete
Full Stack Developer	Full Stack Developer	MegaTechnology	8/23/2024 12:00:00 AM	95000	Proficiency in both front-end and back-end technologies, experience with frameworks such as React and Node.js, ability to work in a fast-paced environment.	Edit Details Delete
Front End Developer Intern	Front-End Developer	MegaTechnology	8/23/2024 12:00:00 AM	25000	Basic understanding of HTML, CSS, and JavaScript, eager to learn and contribute to front-end development projects.	Edit Details Delete
Senior Back End Developer	Back-End Developer	MegaTechnology	8/23/2024 12:00:00 AM	95000	5+ years of experience in back-end development, proficiency in relevant technologies such as Node.js, Python, or Java, strong understanding of databases and API design.	Edit Details Delete
Product Manager	Product Manager	MegaTechnology	8/23/2024 12:00:00 AM	110000	Strong leadership and communication skills, experience in product management, ability to prioritize and manage multiple projects.	Edit Details Delete
Tester	Tester	Mega Technology	8/31/2024 1:42:45 AM	20000	1	Edit Details Delete

Create New

Figure 106: Job in Job Seeker

The webpage from "FPTJobMatch" presents a list of job openings in the technology sector. Each job listing includes the job title, role description, job category, company name, posting date and time, salary offer, and specific applicant requirements. The jobs are organized in a table format, which facilitates easy comparison of different positions. At the top right corner, an email address shows that a user named "Seeker" is currently logged in. This design aims to offer a user-friendly experience for job seekers, enabling them to easily browse and apply for positions in their field of interest.

6. Manage your Account

Manage your account

Change your account settings

Profile
Email
Password
Two-factor authentication
Personal data

Profile

Username
JobSeeker1@gmail.com

Phone number
0905614566

Save

Figure 107: Manage Your Account

The screenshot shows an account management page titled "Manage your account," offering users options to update their account settings. These options include Profile, Email, Password, Two-Factor Authentication, and Personal Data. On the right side of the page, there is a completed profile form displaying a username and phone number. This design allows users to easily navigate between different sections of their account settings and make necessary changes. It is a typical layout found in many web applications, providing a clear and user-friendly interface for managing accounts.

IV. Screenshots of using GitHub or GitLab to manage the source code:

Step 1: This process involves creating separate branches for each group member to work on their individual tasks independently. Once finished, they can upload their code to their specific branch for others to access. Team members can also pull code from each other's branches if needed. At the end of the project, the leader will merge all the code into the main branch.

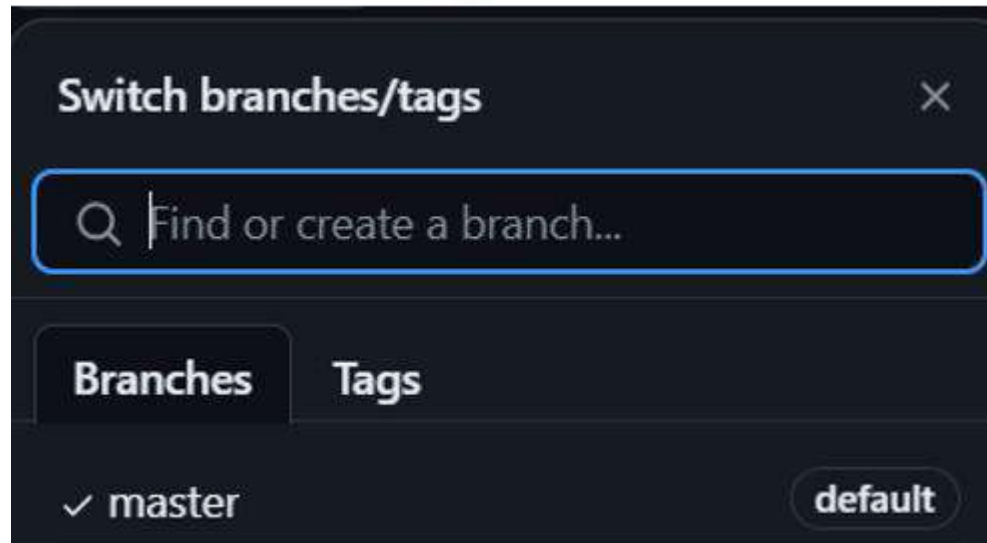


Figure 108: Branch

Step 2: This involves using Visual Studio, where members can view branches created by users on GitHub. They just need to select their branch and begin coding in that branch.

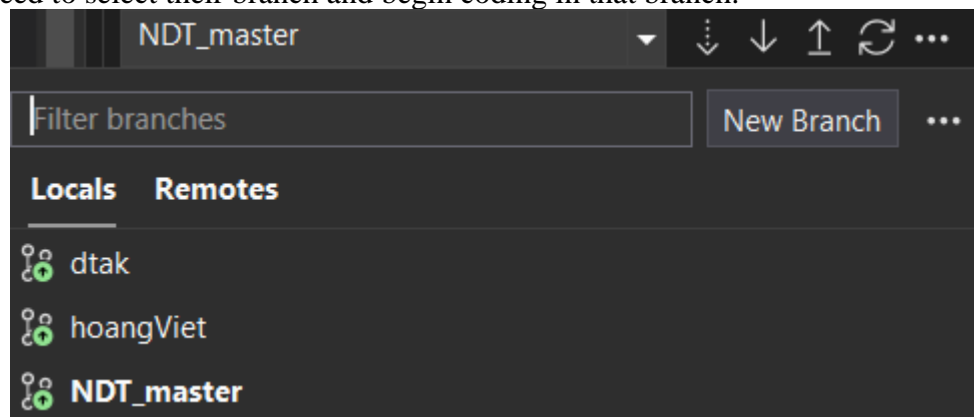


Figure 109: Choose branch

Step 3: After completing the coding of a specific feature, users can view their code changes in this section. If needed, they can compare their code with previous versions to identify any potential issues. Once they are satisfied with their updates, they can commit and push their code to the respective branch.

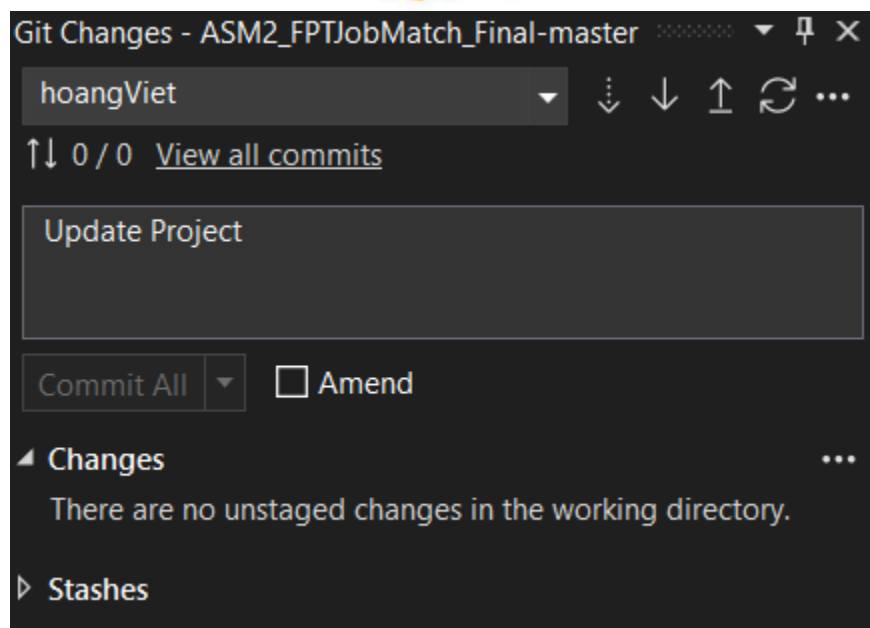


Figure 110: Commit and Push

Step 4: This page acts as the central hub for all GitHub files, allowing members to easily track new file additions. They can also monitor the number of commits and see the timestamp of the last edit made on GitHub.



















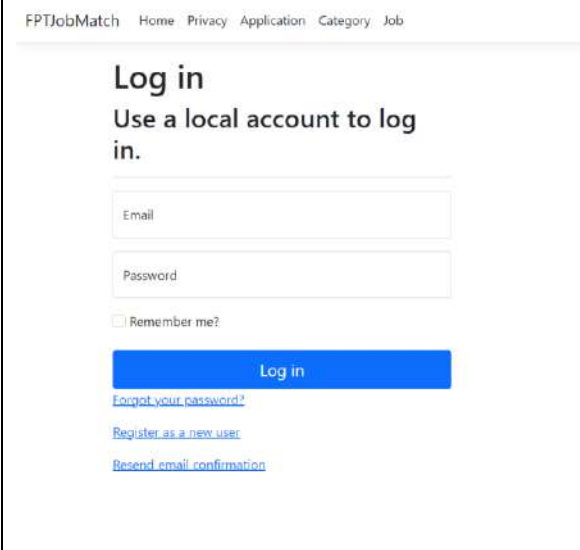
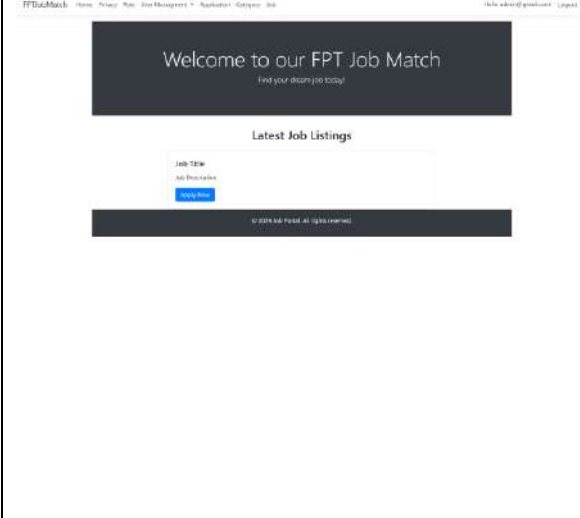
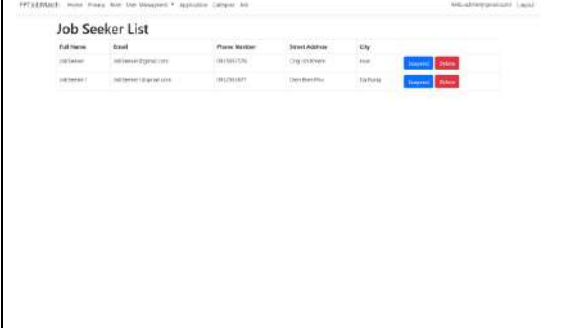
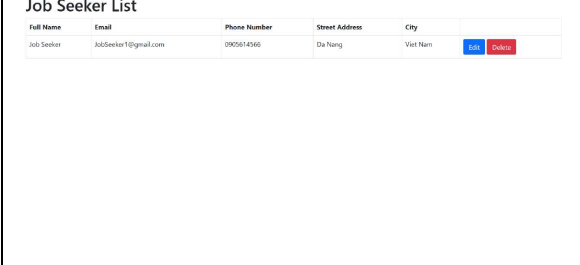
 ndt0403 Add project files.	d544c89 · 31 minutes ago	 1 Commits
 Areas/Identity/Pages	Add project files.	31 minutes ago
 Controllers	Add project files.	31 minutes ago
 Data	Add project files.	31 minutes ago
 Migrations	Add project files.	31 minutes ago
 Models	Add project files.	31 minutes ago
 Properties	Add project files.	31 minutes ago
 Repository	Add project files.	31 minutes ago
 Views	Add project files.	31 minutes ago
 wwwroot	Add project files.	31 minutes ago
 .gitattributes	Add project files.	31 minutes ago
 .gitignore	Add project files.	31 minutes ago
 ASM2_FPTJobMatch.csproj	Add project files.	31 minutes ago
 ASM2_FPTJobMatch.sln	Add project files.	31 minutes ago
 Program.cs	Add project files.	31 minutes ago
 app.db	Add project files.	31 minutes ago
 appsettings.Development.json	Add project files.	31 minutes ago

Figure 111: Github code



C. Review the performance of your business application against the Problem Definition Statement and initial requirements:

I. Review the performance of the application:

Test Case	Test Content	Testing	Expected	Actual result	Pass/Fail

1	Check accessibility for people who are not logged in	Click on the items	Lead people to the login page		Pass
2	Check the effectiveness of logging into the admin account	Fill in the admin information created previously (Email: Admin@gmail.com , Password: Admin12345@)	Successfully leads to Admin page		Pass
3	Check the user management functionality of the admin role	Click on the User Management section, select any role to manage (Job Seeker)	Leads to the management page of Job Seeker		Pass
4	Check for user deletion feature	Click on the Delete button (JobSeeker2 is the deleted user)	JobSeeker2 has been completely removed from the Job Seeker		Pass

			management page		
5	Check for the ability to manage Categories	Click on Category	The category management page is displayed, information about the category will be displayed on the page		Pass
6	Check the Employer role's ability to manage recruitment applications	Click on Application	Employer can view information about job seeker, apply job in job list		Pass
7	Check the Employer role's ability to manage category information	Click on Category to view the category information	Employer can view category and create new category		Pass
8	Check the Employer role's ability to create categories	Click on Create new to add new category to list	When choose button Create then Employer can input name and Description and click Save. New Category auto set status Pending		Pass

9	Check the Job Seeker role's ability to access the Application page	Click on Application	Job Seeker can apply job when choose button Application	Application 	Pass
10	Check the Suspend account user feature from the Admin role	Click on Suspend button	When admin suspend account seeker or employer they cannot login.	Employer List 	Fail

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

*** Conditions satisfied**

The application features a thoughtfully designed user interface that offers a smooth and intuitive experience. Each function and feature is clearly defined, allowing users to navigate and use the application effortlessly while quickly understanding its functionalities.

The search functionality is flexibly integrated throughout the application, making job postings and recruitment processes more accessible. This enhancement greatly improves user experience by helping customers easily find relevant job opportunities.

Application owners can efficiently manage and monitor customer activity with tools that allow them to review and update job postings, thereby enhancing overall system management.

Administrators have comprehensive control over both customer and business accounts, including the ability to reset passwords and resolve access issues swiftly. This strong account management system ensures user information is secure and provides exceptional customer support.

*** Conditions need to be improved**

To enhance the user experience and boost system efficiency, several key improvements are essential. The foremost priority is to strengthen the security of customer data and transactions. Implementing two-factor authentication is a crucial measure to add an extra layer of protection for sensitive information.

In addition, introducing an advanced notification system would greatly benefit users by providing timely updates on job postings, employer information, and other important details. This would keep users well-informed and engaged. It is also important to refine the application's interface for both mobile and desktop platforms to

improve accessibility and usability.

Improving job portfolio management is another critical upgrade. By implementing role-specific permissions, the application can ensure safer and more efficient operations. Additionally, incorporating advanced statistical and reporting tools will help monitor business performance and identify key trends, which can guide strategic decisions.

Looking ahead, comprehensive testing across various browsers and devices is planned to ensure the application's functionality and adaptability on different platforms. This step is crucial for maintaining a robust and versatile application, ensuring a seamless experience for all users. These enhancements will support the application's growth and enhance its ability to serve users effectively, making it a more valuable resource for both professional and personal use.

III. Analyse the factors that influence the performance of the application:

The application's performance is shaped by several key factors, each vital for maintaining its overall efficiency and reliability. One important factor is the user interface, which is carefully designed to be both detailed and clear. This thoughtful design significantly improves the user experience by making navigation straightforward and facilitating easy interaction with the application.

Another crucial aspect is the management of job postings, which impacts the application's performance significantly. Efficient management ensures accurate tracking of candidate statuses, which is essential for operational efficiency. Any shortcomings in this area could lead to problems such as delays or repeated spamming, negatively affecting the user experience and the application's credibility.

Effective admin account management is also crucial, particularly for resolving access issues. Proper management ensures that user problems are addressed quickly, contributing to a smoother user experience and enhanced security.

Security is especially critical in a job registration application, where protecting personal information and transaction details is paramount. A strong security framework is necessary to prevent data breaches, which could otherwise undermine user trust and damage the application's reputation.

Finally, secure and reliable payment integration is essential for smooth transactions. Poor payment integration can cause issues like payment errors or lost transactions, which could erode user trust and discourage continued use of the application.

In summary, the application's performance depends on a comprehensive approach to improvement and optimization. Enhancing the user interface, strengthening security measures, and ensuring reliable payment processes all require continuous attention and refinement. Only with such thorough oversight and ongoing enhancement can the application maintain and expand its user base while ensuring user satisfaction and trust.

IV. Evaluate the strengths and weaknesses of the application:

*** Strengths:**

User-Friendly Interface: The application offers a well-organized and intuitive user interface, enhancing

the user experience by making navigation simple and helping users easily understand how to utilize its features.

Effective Job Post Management: It provides efficient tools for tracking and managing job postings, which is essential for operational efficiency and for users to accurately monitor candidate statuses.

Admin Account Management: Strong management of admin accounts enables quick resolution of user access issues, bolstering security and user trust.

Secure Payment Integration: The application incorporates secure payment methods, ensuring a smooth and reliable transaction process, which boosts user confidence in managing financial transactions within the platform.

*** Weaknesses:**

Potential Management Inefficiencies: Inefficient management of job postings could result in delays and issues such as continuous spamming, negatively affecting user experience and the overall performance of the application.

Security Concerns: Although the application implements security features like two-factor authentication, there is a continuous need to strengthen the security framework to prevent data breaches. Protecting personal information and transaction data is crucial in a job registration application.

Dependency on Continuous Optimization: The application requires regular optimization and testing across different platforms to ensure compatibility and smooth operation. This ongoing need for updates and improvements can be resource-intensive.

Stability of Payment Integration: Any instability or security issues in the payment integration could lead to transaction errors or loss of funds, potentially damaging user trust and discouraging further use of the application.

=> These strengths and weaknesses underscore the application's areas of excellence as well as opportunities for improvement to maintain its competitiveness and reliability for users.