

ASSIGNMENT 2 FRONT SHEET

Qualification	BTEC Level 5 HND Diploma in Computing		
Unit number and title	Unit 14: Application Development		
Submission date		Date Received 1st submission	
Re-submission Date		Date Received 2nd submission	
Student Name	Vu Hoang Nam	Student ID	GCD210429
Class	GCD1104	Assessor name	PHAM THANH SON
Student declaration <p>I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.</p>			
		Student's signature	Nam

Grading grid

P4	P5	P6	M3	M4	M5	D2	D3

<p>⚙ Summative Feedback:</p>		<p>⚙ Resubmission Feedback:</p>
Grade:	Assessor Signature:	Date:
<p>IV Signature:</p>		

Table of Contents

I. PEER REVIEW AND FEEDBACK ANALYSIS.....	7
1. Formal questionnaire to review the business application(P4).	7
2. Collect review feedback(P4):.....	10
3. Interpret peer-review feedback(M3):	16
3.1. Interpret Feedbacks:	16
3.2. Identify Opportunities:.....	18
II. APPLICATION DEVELOPMENT	19
1. Folder Structure Of The Application(P5):	19
2. Source code samples of the application with explanation(M3):	24
2.1. Model:	24
2.2. Controller:	34
2.3. View:.....	58
3. Provide Final screenshots of the application:	112
4. Screenshots of using GitHub or GitLab to manage the source code:	123
III. APPLICATION EVALUATION	124
1. Review the performance of the application(P6)	124
2. Conclude whether the application adapts all requirements or it needs to be improved later(P6)	128
3. Develop a functional business application based on a specific Software Design Document with supportive evidence of using the preferred tools, techniques and methodologies(M4).....	129
3.1. C#.....	130
3.2. MVC .net core	130

3.3. Html, Css and Javascript.....	131
3.4. Bootstrap.....	133
3.5. Figma	134
3.6. Project libre	134
3.7. ERD	134
3.8. SQL Server	134
3.9. Waterfall Model.	135
3.10. Visual Studio Code.....	136
3.11. Github.....	137
4. Analyze the factors that influence the performance of the application(M5).....	138
5. Evaluate any new insights, ideas or potential improvements to your system and justify the reasons why you have chosen to include (or not to include) them as part of this business application(D2). ...	140
6. Critically evaluate the strengths and weaknesses of your business application and fully justify opportunities for improvement and further development(D3).	141
REFERENCES	142

Table of Figure:

Figure 1 Project structure.....	20
Figure 2 Area Admin	21
Figure 3 Area Employer	21
Figure 4 Models	22
Figure 5 View Models	22
Figure 6 Admin Controllers.....	22
Figure 7 Employer Controllers.....	23
Figure 8 JobSeeker Controller	23
Figure 9 Shared folder in View	23
Figure 10 Models folder	24
Figure 11 Home page	113
Figure 12 Login page.....	114
Figure 13 Register for Admin and Employer	114
Figure 14 Admin page.....	115
Figure 15 Register for User	116
Figure 16 Post category of Employer	117
Figure 17 Category page.....	119
Figure 18 Job page.....	120
Figure 19 Apply job page.....	121
Figure 20 Source code in Github	123

Figure 21: C#	130
Figure 22: C#	130
Figure 23: MVC .net core	131
Figure 24: HTML	131
Figure 25: CSS	132
Figure 26: Javascript	132
Figure 27: Bootstrap libray link	133
Figure 28: Bootstrap icon	133
Figure 29: Bootstrap for notification	133
Figure 30: Create relationship of model from ERD	134
Figure 31: Create relationship of model from ERD	134
Figure 32: SQL Server	135
Figure 33: Visual Studio Code	137
Figure 34: Github	137
Table 1 Test Case	125

I. PEER REVIEW AND FEEDBACK ANALYSIS

1. Formal questionnaire to review the business application(P4).

To comprehensively evaluate and improve our business application, we've developed a structured questionnaire to gather valuable feedback from users and stakeholders. This questionnaire covers various aspects, including user experience, issue identification, proposed solutions, and strategic development insights. Our goal is to refine the application to better meet the diverse needs and expectations of our user base. By addressing specific questions related to user experiences, potential challenges, proposed solutions, and strategic development plans, we aim to gain a holistic understanding of our application's strengths and areas for improvement.

Job Seeking

* Indicates required question

1. How do you feel about your experience using the Job Seeking app? *

- ☐ 1(Very poor)
- ☐ 2(Poor)
- ☐ 3(Average)
- ☐ 4(Good)
- ☐ 5(Very good)

2. Does the Job Seeking app deliver the features you expect? *

- ☐ 1(Very Poor)
- ☐ 2(Poor)
- ☐ 3(Average)
- ☐ 4(Good)
- ☐ 5(Very good)

3. Do you feel secure about the safety and security of your personal information when using the Job Seeking Application? *

- ☐ 1(Very poor)
- ☐ 2(Poor)
- ☐ 3(Average)
- ☐ 4(Good)
- ☐ 5(Very good)

4. How would you rate Job Seeking app user feedback and technical support process? 

- ☐ 1 (Very poor)
- ☐ 2 (Poor)
- ☐ 3 (Average)
- ☐ 4 (Good)
- ☐ 5 (Very good)

Job Seeking

1. Does our app solve your problem?

Your answer

2. What special features and functionalities does our program have, in your opinion?

Your answer

3. What unique features or functionality do you have that offer compared to existing solutions?

Your answer

4. Do you have any feedback on how we can improve user experience through our interface?

Your answer

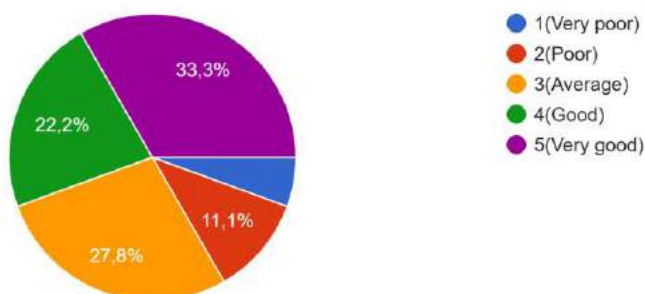
2. Collect review feedback(P4):

The survey results provide valuable insights into users' perceptions of the Job Seeking app across various aspects. Firstly, regarding the overall user experience, it is notable that a significant portion of respondents rated their experience as "Average" (33.3%), with smaller percentages indicating "Good" (22.2%) and "Very good" (27.8%) experiences. However, there were also notable concerns, with a combined 16.7% of users rating their experience as either "Very poor" or "Poor." This suggests a need for

improvement in certain areas to enhance user satisfaction. Moving on to feature delivery, the feedback is somewhat mixed, with a considerable proportion rating it as "Average" (38.9%). While there are positive ratings for security (55.6% rated as either "Good" or "Very good"), a significant minority express concerns, with 16.7% rating it as "Poor" or "Very poor." Moreover, the user feedback and technical support process received generally positive ratings, with 38.9% rating it as "Good" and 38.9% as "Very good." However, there is room for improvement, as 11.1% of respondents rated it as "Average." Overall, these insights highlight both strengths and areas for enhancement within the Job Seeking app, underscoring the importance of ongoing refinement to meet user expectations and ensure a positive user experience.

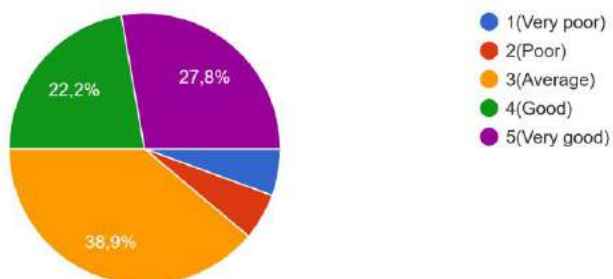
1. How do you feel about your experience using the Job Seeking app?

18 câu trả lời



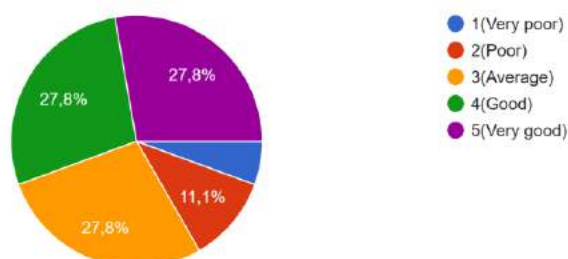
2. Does the Job Seeking app deliver the features you expect?

18 câu trả lời



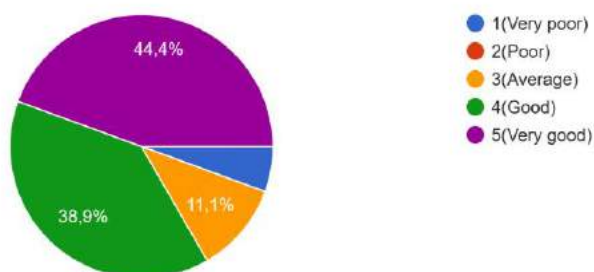
3. Do you feel secure about the safety and security of your personal information when using the Job Seeking Application?

18 câu trả lời



4. How would you rate Job Seeking app user feedback and technical support process?

18 câu trả lời



The survey results offer valuable insights into users' experiences and perceptions of the Job Seeking app, covering various aspects such as problem-solving effectiveness, special features, uniqueness compared to existing solutions, and interface usability. Firstly, concerning the app's problem-solving capabilities, feedback varied, with a notable portion expressing appreciation for its assistance in addressing their needs. However, there were also concerns raised about specific features or functionalities lacking, indicating room for improvement to enhance overall effectiveness. In terms of special features and functionalities, users highlighted innovative aspects such as the job-matching algorithm, real-time collaboration tools, and comprehensive reporting features, which contribute to streamlining the job-seeking process. Moreover, the app's unique offerings, including AI-powered job matching, skills assessment tools, and blockchain-based verification, set it apart from existing solutions, providing users with additional value and differentiation. Additionally, feedback on interface usability highlighted areas

for improvement, such as simplifying navigation, enhancing search functionality, and ensuring responsiveness across devices. Overall, the survey results underscore both strengths and opportunities for enhancement within the Job Seeking app, emphasizing the importance of ongoing refinement to meet user expectations and deliver an optimal user experience.

1. Does our app solve your problem?

7 câu trả lời

Yes

I appreciate the effort put into developing your app, but I'd like to provide some detailed feedback on how it could better solve my problem. Firstly, [mention specific features or functionalities that are lacking]. Additionally, [highlight any usability issues or areas for improvement]. Overall, I believe addressing these points would greatly enhance the effectiveness of your app.

Yes, your app has been incredibly helpful in solving my problem. It streamlines the process and makes it much easier to [describe the specific problem/task your app addresses]. I'm impressed with the functionality and user-friendly interface.

Overall, your app provides some useful features that address parts of my problem. However, there are still some areas where I encounter challenges or limitations. I believe with some improvements or additional features, it could become even more effective.

Unfortunately, your app doesn't fully solve my problem. While it has some promising features, I still encounter significant difficulties in [describe the specific issues or pain points]. I think there's room for improvement to make it more effective in addressing the needs of users like me.

2. What special features and functionalities does our program have, in your opinion?

7 câu trả lời

machine learning techniques, our program analyzes job listings and candidate profiles to suggest the most suitable matches. This ensures that both employers and job seekers find the perfect fit quickly and efficiently.

Our program boasts a user-friendly interface that makes job posting, application management, and candidate selection a breeze. With intuitive navigation and clear prompts, users can easily navigate through the platform, saving time and minimizing frustration.

We've incorporated real-time collaboration tools into our program, allowing employers and candidates to communicate seamlessly throughout the hiring process. From initial application to final interview scheduling, our platform facilitates smooth and efficient communication, leading to faster hiring decisions.

Our program provides comprehensive reporting and analytics features that offer valuable insights into the recruitment process. Employers can track key metrics such as application conversion rates, candidate quality, and time-to-hire, enabling data-driven decision-making and continuous improvement.

To enhance reach and engagement, our program seamlessly integrates with popular social media platforms. Employers can easily share job postings across social networks, reaching a wider audience of potential candidates and increasing visibility for their opportunities.

3. What unique features or functionality do you have that offer compared to existing solutions?

7 câu trả lời

Using ChatBot with AI

Chat with AI

Our project offers an advanced AI-powered job matching system that goes beyond traditional keyword matching. Using sophisticated algorithms, we analyze job listings and candidate profiles to identify the best matches based on skills, experience, and cultural fit. This ensures higher-quality matches and reduces the time spent on manual screening.

Unlike existing solutions, our project integrates skills assessment tools directly into the platform. Employers can create customized assessments to evaluate candidates' skills and competencies, streamlining the hiring process and ensuring a better fit for the job.

One of our standout features is an interactive virtual interviewing platform that offers a more immersive experience than traditional video conferencing tools. With features such as virtual whiteboards, code editors, and collaborative document editing, we provide a dynamic environment for conducting interviews and assessing candidates.

4. Do you have any feedback on how we can improve user experience through our interface?

7 câu trả lời

Good

Simplify navigation by organizing menus and links logically. Ensure that users can easily find essential features such as job search, posting, and application management. Consider implementing breadcrumbs or a sticky navigation bar for easy access to different sections of the platform.

Enhance the search functionality to allow users to find relevant job listings quickly. Implement filters such as location, industry, salary range, and job type to narrow down search results. Consider incorporating autocomplete suggestions and advanced search options for improved usability.

Use clear and concise call-to-action buttons to guide users through the desired actions, such as "Apply Now," "Post a Job," or "Sign Up." Ensure that these buttons are prominently displayed and visually distinct to encourage user interaction.

Optimize the presentation of job listings to provide essential information at a glance. Use clear headings, bullet points, and icons to highlight key details such as job title, company name, location, and application deadline. Consider incorporating visual elements such as company logos and job preview images for added context.

3. Interpret peer-review feedback(M3):

3.1. Interpret Feedbacks:

Survey 1 Interpretation:

Q1. Problem-solving Effectiveness:

Feedback: Users provided mixed responses regarding the app's ability to solve their problems. While some users appreciate certain aspects of the app, others feel that it lacks specific features or functionalities.

Interpretation: There is a need to further enhance the app's problem-solving capabilities to better meet user needs and expectations. This could involve implementing additional features or improving existing functionalities to address user concerns more effectively.

Q2. Special Features and Functionalities:

Feedback: Users highlighted several standout features of the app, such as the job-matching algorithm, user-friendly interface, and real-time collaboration tools.

Interpretation: The app's special features are positively received by users and contribute to a positive user experience. These features should be maintained and further optimized to continue providing value to users.

Q3. Uniqueness Compared to Existing Solutions:

Feedback: Users mentioned unique features like AI-powered job matching and integrated skills assessment tools as distinguishing factors of the app.

Interpretation: The app offers unique functionalities that set it apart from existing solutions in the market. Leveraging these unique features can help attract and retain users by providing them with innovative tools for their job search.

Q4. Interface Usability:

Feedback: Users provided recommendations for improving interface usability, such as simplifying navigation and enhancing search functionality.

Interpretation: While the overall feedback on interface usability is positive, there are areas for improvement to make the app more intuitive and user-friendly. Addressing these recommendations can lead to a better overall user experience and higher user satisfaction.

Survey 2 Interpretation:

Q1. Problem-solving Effectiveness:

Feedback: Users expressed mixed opinions on the app's ability to solve their problems, with some users finding it meets their expectations and others encountering issues with certain features.

Interpretation: There is a need to address user concerns and improve the app's problem-solving effectiveness. This could involve refining existing features or adding new functionalities to better address user needs and preferences.

Q2. Special Features and Functionalities:

Feedback: Users highlighted unique features such as AI-powered job matching and personalized career path recommendations as key aspects of the app.

Interpretation: The app's special features offer added value to users and contribute to a more comprehensive job-seeking experience. Continuing to innovate and enhance these features can help differentiate the app from competitors.

Q3. Uniqueness Compared to Existing Solutions:

Feedback: Users mentioned unique aspects like the interactive virtual interviewing platform and blockchain-based verification as distinguishing factors of the app.

Interpretation: The app offers innovative solutions not found in other existing platforms, giving it a competitive edge in the market. Leveraging these unique features can attract more users and increase user engagement.

Q4. Interface Usability:

Feedback: Users provided suggestions for improving interface usability, such as simplifying navigation and enhancing search functionality.

Interpretation: Addressing user feedback on interface usability can lead to a more intuitive and user-friendly app. Implementing these improvements can enhance the overall user experience and drive user satisfaction and retention.

3.2. Identify Opportunities:

Dynamic Recommendation Engine for Job Seekers:

Opportunity: Implement a dynamic recommendation engine that suggests job listings to users based on their past searches, profile information, and job application history. By analyzing user behavior and preferences, the system can recommend relevant job opportunities tailored to each user's skills, experience, and career goals.

Automated Application Tracking System:

Opportunity: Develop an automated application tracking system that allows users to track the status of their job applications in real time. The system can notify users about application updates, interview invitations, and job offer decisions, providing them with timely and relevant information throughout the job application process.

Skill Matching Algorithm:

Opportunity: Develop a skill-matching algorithm that analyzes job requirements and candidates' skills to provide personalized job recommendations. The algorithm can assess candidates' skills based on their profiles, resumes, and assessments, matching them with job listings that best fit their qualifications and expertise.

Interactive Interview Preparation Tools:

Opportunity: Integrate interactive interview preparation tools into the platform to assist job seekers in improving their interview skills and confidence. These tools can include mock interview simulations, personalized feedback, and tips on answering common interview questions effectively.

Machine Learning for Job Matching:

Opportunity: Implement machine learning algorithms to enhance job-matching capabilities. By analyzing historical job application data, candidate profiles, and employer preferences, the system can predict the likelihood of a successful job match for each candidate-job pair. This predictive model can prioritize job listings based on their relevance to candidates and vice versa, improving the overall efficiency and effectiveness of the job search process.

Automated Resume Parsing:

Opportunity: Integrate automated resume parsing technology to extract relevant information from job seekers' resumes and populate their profiles. This technology can automatically parse resumes to extract details such as work experience, education, skills, and certifications, eliminating the need for manual data entry by users.

II. APPLICATION DEVELOPMENT

1. Folder Structure Of The Application(P5):

ASP.NET Core's Models, Views, and Controllers (MVC) architectural style was used to create our website, Job Seeking. Below are all of the source folders containing source code for the project: Here are all folders and files of my project (Folder Name: JobSeeking)

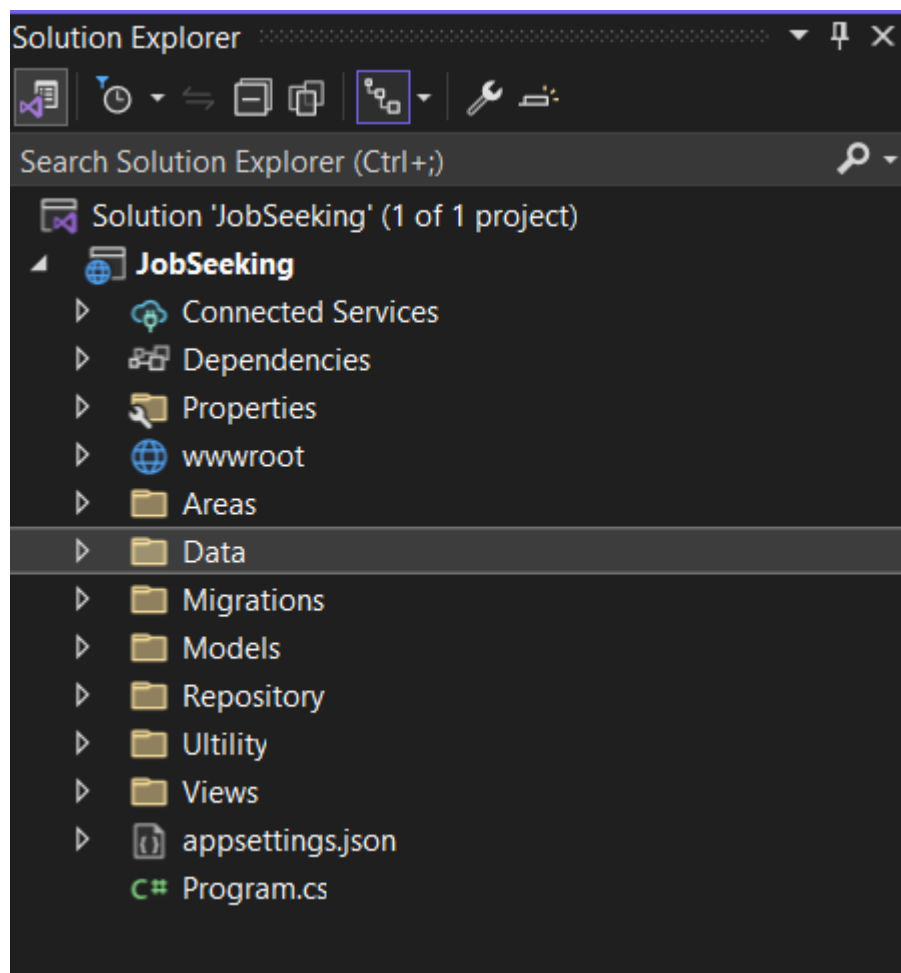


Figure 1 Project structure

The Area folder contains the Admin folder, which has the controller view and the project View

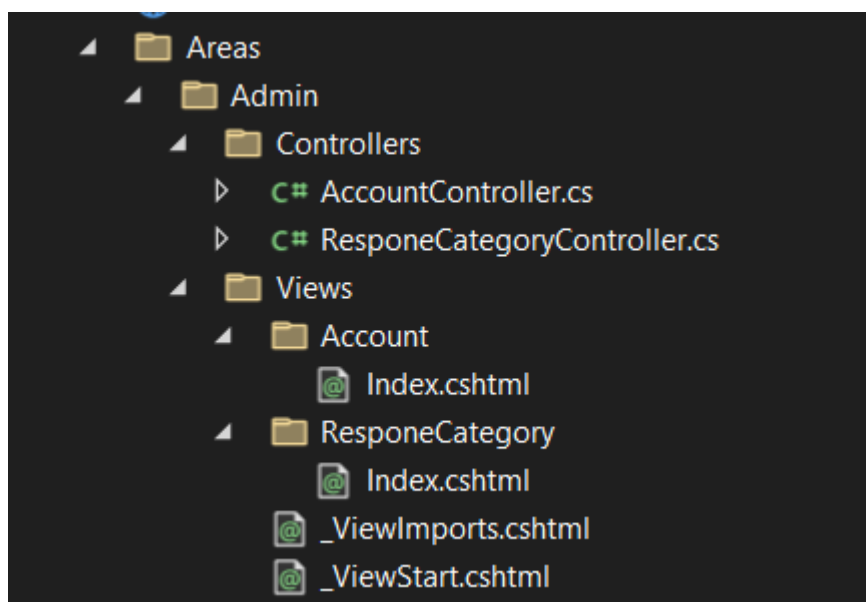


Figure 2 Area Admin

Employer's Areas folder contains controllers and Views that limit customer access.

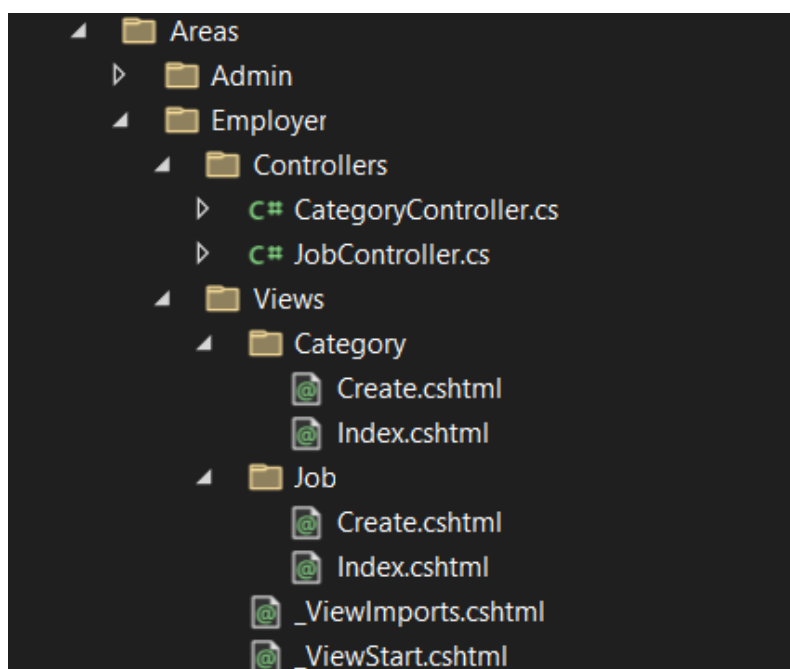


Figure 3 Area Employer

The JobSeeker's Areas folder contains controllers and Views that limit Store access for sales.

(thiếu hình của job seeker)

Models folder: It represents the data and business logic for ApplicationUser, ApplyCV, Category, ErrorViewModel, Job, and News of my application.

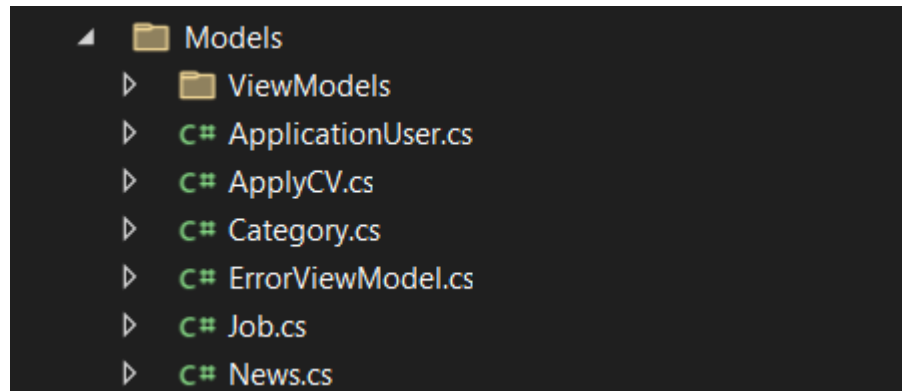


Figure 4 Models

View Models: They represent the data and business logic for JobSeekingVM

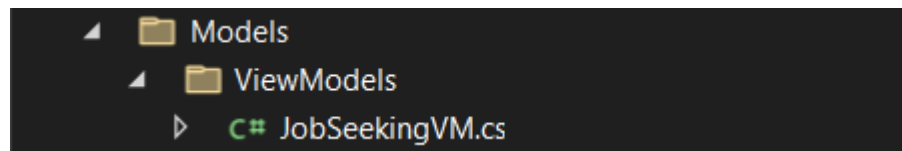


Figure 5 View Models

We domain each role to have different tasks as well as different functions. So below I will provide controllers for 3 roles: Admin, Employer, and JobSeeker.

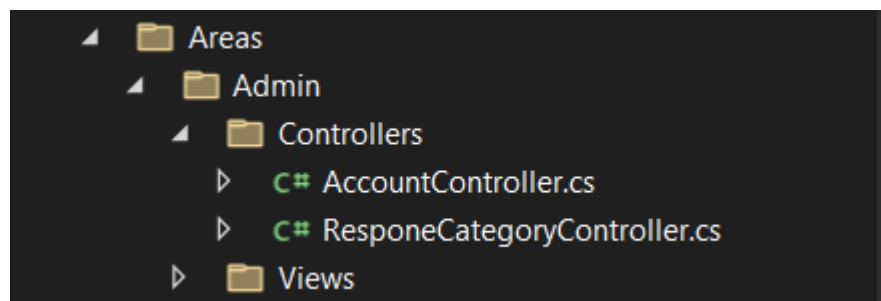


Figure 6 Admin Controllers

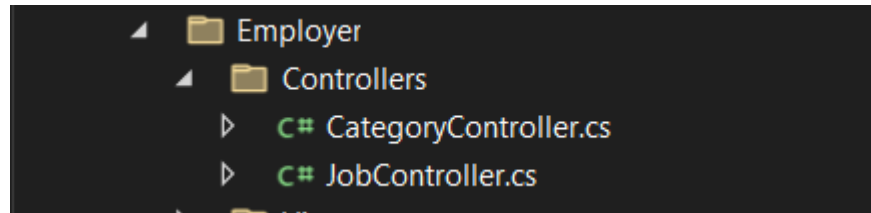


Figure 7 Employer Controllers

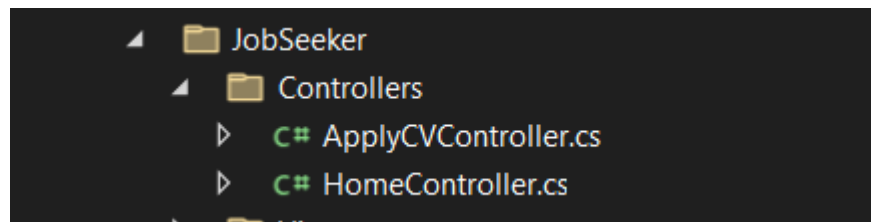


Figure 8 JobSeeker Controller

Shared folder in View: It's used to store view templates and components that are shared across multiple pages or views within my application.

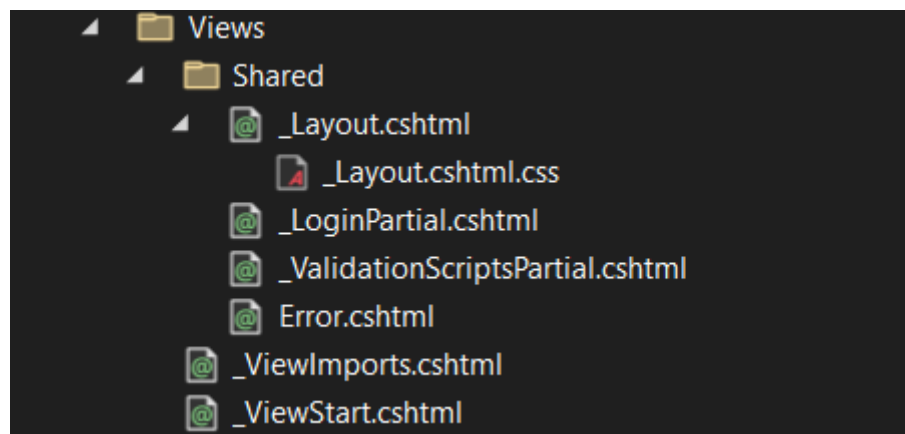
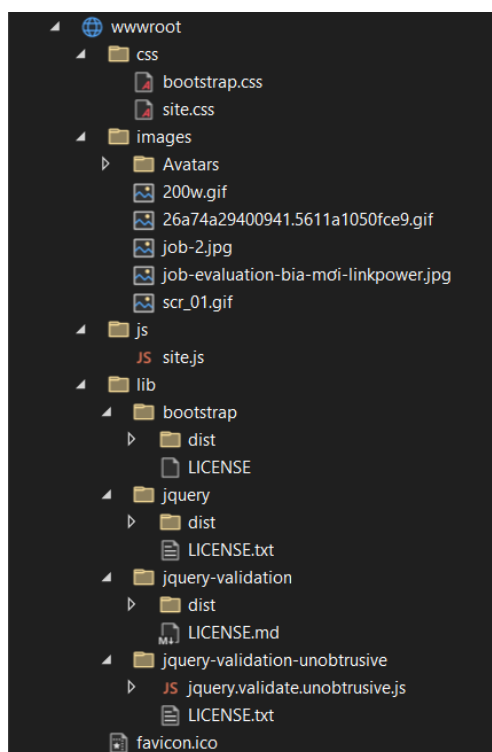


Figure 9 Shared folder in View

wwwroot folder: This is the location where the static, CSS, JS, fonts, images, and lib folders are stored. Directories are served directly to the client by the web server without any processing by the application.



2. Source code samples of the application with explanation(M3):

2.1. Model:

I developed all Models required for my project such as ViewModels folder, ApplicationUser, ApplyCV, Category, ErrorViewModel, Job, and News.

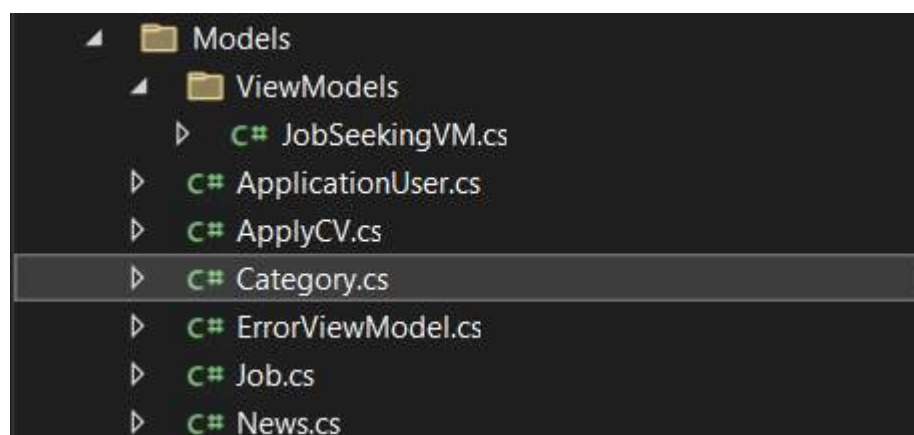


Figure 10 Models folder

JobSeeking.Models.ViewModels.JobSeekingVM

```
using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace JobSeeking.Models.ViewModels
{
    public class JobSeekingVM
    {
        [ValidateNever]
        public Job Job { get; set; }
        [ValidateNever]
        public Category Category { get; set; }
        [ValidateNever]
        public IEnumerable<SelectListItem> Categories { get; set; }
        [ValidateNever]
        public ApplyCV applyCV { get; set; }
    }
}
```

Job Property:

- This property represents a Job object and is decorated with the [ValidateNever] attribute.
- The [ValidateNever] attribute indicates that this property should not be validated during model binding in ASP.NET Core MVC.

Category Property:

- This property represents a Category object and is also decorated with the [ValidateNever] attribute.
- Similar to the Job property, it specifies that the Category property should not undergo validation during model binding.

Categories Property:

- This property is of type IEnumerable<SelectListItem> and is used to hold a collection of SelectListItem objects.
- SelectListItem objects are typically used to populate dropdown lists in HTML forms.
- Like the previous properties, it is decorated with [ValidateNever], indicating that it should not be validated during model binding.

applyCV Property:

- This property represents an ApplyCV object, which is likely used for handling job applications or CV submissions.

- It is also decorated with [ValidateNever], meaning it won't be validated during model binding.

JobSeeking.Models.ApplicationUser

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
using System.ComponentModel.DataAnnotations;

namespace JobSeeking.Models
{
    public class ApplicationUser:IdentityUser
    {
        public string Name { get; set; }
        [ValidateNever]
        public string Address { get; set; }
        [ValidateNever]
        public string City { get; set; }
        [ValidateNever]
        public string? Company { get; set; }
        [ValidateNever]
        public bool isValid { get; set; }
        [ValidateNever]
        public string? Avatar { get; set; }

        [Timestamp]
        public byte[] Version { get; set; }
    }
}
```

ApplicationUser class:

- This class inherits from IdentityUser, indicating that it represents a user in the ASP.NET Identity system.
- Properties:
 - Name: Represents the name of the user.
 - Address: Represents the address of the user. The [ValidateNever] attribute suggests that this property should not be included in model validation.
 - City: Represents the city of the user. Similarly, the [ValidateNever] attribute suggests excluding it from model validation.
 - Company: Represents the company associated with the user. The ? indicates that this property can be null.

- isValid: Represents whether the user is valid or not. This might be used for account activation or validation purposes.
- Avatar: Represents the URL or path to the user's avatar image. The ? indicates that this property can be null.
- Version: This property is decorated with the [Timestamp] attribute, which indicates that it will store the row version value generated by the database.

using directives:

- These directives import namespaces required for the code to compile and run properly. In this case, Microsoft.AspNetCore.Identity and Microsoft.AspNetCore.Mvc.ModelBinding.Validation are imported.

JobSeeking.Models.ApplyCV

```
using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
using System.ComponentModel.DataAnnotations.Schema;

namespace JobSeeking.Models
{
    public class ApplyCV
    {
        public int Id { get; set; }
        [ValidateNever]
        public int JobId { get; set; }
        [ForeignKey("JobId")]
        [ValidateNever]
        public Job Job { get; set; }
        [ValidateNever]
        public string JobSeekerEmail { get; set; }
        public DateTime TimeCreate { get; set; } = DateTime.Now;
        public string Description { get; set; }
        [ValidateNever]
        public string CV { get; set; }
        [ValidateNever]
        public bool? Status { get; set; }
    }
}
```

ApplyCV Class:

- This class represents the application for a job by a job seeker.

Id Property:

- This property represents the unique identifier for the application. It is typically used as the primary key in the database.

JobId Property:

- This property represents the foreign key for the associated job. It establishes a relationship between the ApplyCV and Job entities.

Job Property:

- This property represents the navigation property for the associated job. It allows accessing the related Job entity from an instance of ApplyCV.

JobSeekerEmail Property:

- This property stores the email address of the job seeker who applied for the job.

TimeCreate Property:

- This property stores the timestamp indicating when the application was created. It defaults to the current date and time (DateTime.Now) when not explicitly set.

Description Property:

- This property stores any additional description or notes provided by the job seeker along with the application.

CV Property:

- This property stores the path or reference to the job seeker's CV (resume) file.

Status Property:

- This property indicates the status of the application, typically representing whether it's approved, rejected, or pending. It's nullable (bool?) to allow for a null state indicating an unspecified status.

Attributes:

- The [ValidateNever] attribute indicates that the properties decorated with it should not be included in model validation. This is useful for properties that are meant for internal use or are not intended to be validated.
- The [ForeignKey] attribute specifies the foreign key property for the relationship defined by the Job navigation property.

JobSeeking.Models.Category

```
using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace JobSeeking.Models
{
    public class Category
    {
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        public bool isValid { get; set; }
        [ValidateNever]
        public string EmployerId { get; set; }
        [ForeignKey("EmployerId")]
        [ValidateNever]
        public ApplicationUser User { get; set; }
        public DateTime CreateDay { get; set; } = DateTime.Now;
    }
}
```

Category Class:

- This class represents a category in the job seeking application.

Id Property:

- Represents the unique identifier for the category. It's typically used as the primary key in the database.

Name Property:

- Represents the name of the category. It's decorated with the [Required] attribute, indicating that it must have a value.

isValid Property:

- Represents whether the category is valid or not. This could be used for various purposes, such as marking categories as inactive or deleted.

EmployerId Property:

- Represents the foreign key for the associated employer (user). It establishes a relationship between the Category and ApplicationUser entities.

User Property:

- Represents the navigation property for the associated employer (user). It allows accessing the related ApplicationUser entity from an instance of Category.

CreateDay Property:

- This represents the date and time when the category was created. It defaults to the current date and time (DateTime.Now) when not explicitly set.

categoryValid Property:

- Represents whether the category is valid. This seems redundant with the isValid property and may serve a similar purpose.

Attributes:

- The [ValidateNever] attribute indicates that the properties decorated with it should not be included in model validation. This is useful for properties that are meant for internal use or are not intended to be validated.
- The [ForeignKey] attribute specifies the foreign key property for the relationship defined by the User navigation property.

JobSeeking.Models.ErrorViewModel

```
namespace JobSeeking.Models
{
    public class ErrorViewModel
    {
        public string? RequestId { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

RequestId Property:

- Represents the unique identifier associated with a particular request or error.
- It is of type string?, allowing it to be nullable.
- This property is used to store the ID of the request that resulted in an error.

ShowRequestId Property (with Expression-bodied Member):

- Represents a boolean value indicating whether the RequestId is not null or empty.
- It is a read-only property (get accessor only).
- This property returns true if the RequestId is not null or empty, indicating that a valid request ID is available to display.

JobSeeking.Models.Job

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace JobSeeking.Models
{
    public class Job
    {
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public string Description { get; set; }
        [Required]
        public double Salary { get; set; }
        [ValidateNever]
        public string? CompanyName { get; set; }
        public string? Logo { get; set; }
        [Required]
        [BindProperty]
        public string[] Category { get; set; }
        [Required]
        [ValidateNever]
        public string EmployerId { get; set; }
        [ForeignKey("EmployerId")]
        [ValidateNever]
```

```
public ApplicationUser User { get; set; }  
    [ValidateNever]  
    public int amountOfCV { get; set; }  
}  
}
```

Job Class:

- This class represents a job in the job-seeking application.

Id Property:

- Represents the unique identifier for the job. It's typically used as the primary key in the database.

Name Property:

- Represents the name of the job. It's decorated with the [Required] attribute, indicating that it must have a value.

Description Property:

- Represents the description of the job. It's also decorated with the [Required] attribute.

Salary Property:

- Represents the salary offered for the job. It's decorated with the [Required] attribute, implying that a salary value must be provided.

CompanyName Property:

- Represents the name of the company offering the job. This property is marked with the [ValidateNever] attribute, suggesting that it should not be included in model validation.

Logo Property:

- Represents the logo of the company offering the job. It's nullable (string?) and may contain the path to the logo image.

Category Property:

- Represents the categories associated with the job. It's an array of strings and is decorated with the [Required] attribute, indicating that at least one category must be specified.

EmployerId Property:

- Represents the foreign key for the associated employer (user). It establishes a relationship between the Job and ApplicationUser entities.

User Property:

- Represents the navigation property for the associated employer (user). It allows accessing the related ApplicationUser entity from an instance of Job.

amountOfCV Property:

- Represents the number of CVs associated with the job. This property is marked with the [ValidateNever] attribute, implying that it should not be included in model validation.

JobSeeking.Models.News

```
namespace JobSeeking.Models
{
    public class News
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string? image { get; set; }
        public DateTime DateCreated { get; set; } = DateTime.Now;
    }
}
```

Id Property:

- Represents the unique identifier of the news article.
- It is of type int.

Name Property:

- Represents the name or title of the news article.
- It is of type string.

Description Property:

- Represents the description or content of the news article.
- It is of type string.

image Property:

- Represents the image associated with the news article.
- It is of type string?, allowing it to be nullable.

DateCreated Property:

- Represents the date and time when the news article was created.
- It is of type DateTime.
- It is initialized with the current date and time using DateTime.Now.

2.2. Controller:

2.2.1. Admin:

JobSeeking.Areas.Admin.Controllers.AccountController

```
using JobSeeking.Models;
using JobSeeking.Repository.IRepository;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace JobSeeking.Areas.Admin.Controllers
{
    [Area("Admin")]
    [Authorize(Roles = "Admin")]
    public class AccountController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly UserManager<ApplicationUser> _userManager;

        public AccountController(IUnitOfWork unitOfWork, UserManager<ApplicationUser>
userManager)
        {
            _unitOfWork = unitOfWork;
            _userManager = userManager;
        }

        // View all users
        public IActionResult Index()
        {
            List<ApplicationUser> myList =
            _unitOfWork.ApplicationUserRepository.GetAll().ToList();
            return View(myList);
        }
    }
}
```

```

    }
    public IActionResult Details(string? id)
    {
        if (id==null)
        {
            return NotFound();
        }
        ApplicationUser? user = _unitOfWork.ApplicationUserRepository.Get(c=>c.Id ==
id);
        if (user==null)
        {
            return NotFound();
        }
        return View(user);
    }
    // Delete Account
    public async Task<IActionResult> Delete(string id)
    {
        try
        {
            var user = await _userManager.FindByIdAsync(id);
            if (user == null)
            {
                return NotFound();
            }
            else
            {
                await _userManager.DeleteAsync(user);
                return RedirectToAction("Index");
            }
        }
        catch (Exception ex)
        {
            // Log the exception or handle it as per your requirement
            return StatusCode(500, "Internal server error");
        }
    }

    // Lock Account
    public async Task<IActionResult> LockAccount(string Id)
    {
        try
        {
            var user = await _userManager.FindByIdAsync(Id);
            if (user == null)
            {
                return NotFound();
            }
            else
            {
                user.isValid = !user.isValid;
            }
        }
    }

```

```
        await _userManager.UpdateAsync(user);
        return RedirectToAction("Index");
    }
}
catch (Exception ex)
{
    // Log the exception or handle it as per your requirement
    return StatusCode(500, "Internal server error");
}
}
}
```

Namespace and Using Statements:

- The controller file begins with necessary namespace declarations and uses statements to import required classes and namespaces.

Controller Declaration:

- AccountController is declared, inheriting from Controller, which is the base class for MVC controllers in ASP.NET Core.

Constructor:

- The constructor initializes the AccountController with instances of IUnitOfWork and UserManager<ApplicationUser>, injected through dependency injection.

Index Action:

- The Index action retrieves a list of all users from the database using the IUnitOfWork instance and passes it to the view.

Details Action:

- The Details action retrieves details of a specific user based on the provided ID. If the ID is null or the user is not found, it returns a 404 status code. Otherwise, it returns the user details view.

Delete Action:

- The Delete action deletes a user account based on the provided ID. It first finds the user using UserManager, deletes it asynchronously, and redirects to the Index action.

Lock Account Action:

- The LockAccount action toggles the isValid property of a user, effectively locking or unlocking the account. It finds the user based on the provided ID, updates the isValid property, and redirects to the Index action.

Error Handling:

- Both Delete and LockAccount actions include error handling to catch exceptions and return appropriate HTTP status codes in case of errors.

JobSeeking.Areas.Admin.Controllers.NewsController

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

namespace JobSeeking.Areas.Admin.Controllers
{
    [Area("Admin")]
    [Authorize(Roles = "Admin")]
    public class NewsController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IWebHostEnvironment _webHostEnvironment;
        public NewsController(IUnitOfWork unitOfWork, IWebHostEnvironment webHostEnvironment)
        {
            _unitOfWork = unitOfWork;
            _webHostEnvironment = webHostEnvironment;
        }
        public IActionResult Index()
        {
            List<News> list = _unitOfWork.NewsRepository.GetAll().ToList();
            return View(list);
        }
        public IActionResult Create()
        {
            return View();
        }
        [HttpPost]
        public IActionResult Create(IFormFile? file, News news)
        {
            string wwwrootPath = _webHostEnvironment.WebRootPath;
```

```

        if (ModelState.IsValid)
        {
            string fileName = Guid.NewGuid().ToString() +
Path.GetExtension(file.FileName);
            string newsPath = Path.Combine(wwwrootPath, @"images\ImagesOfNews");
            using (var fileStream = new FileStream(Path.Combine(newsPath, fileName),
FileMode.Create))
            {
                file.CopyTo(fileStream);
            }
            news.image = @"\images\ImagesOfNews\" + fileName;
            _unitOfWork.NewsRepository.Add(news);
            _unitOfWork.NewsRepository.Save();
            return RedirectToAction("Index");
        }
        return View(news);
    }

    public IActionResult Edit(int? id)
    {
        if (id==null||id==0)
        {
            return NotFound();
        }
        News ? news = _unitOfWork.NewsRepository.Get(c => c.Id == id);
        if (news == null)
        {
            return NotFound();
        }
        return View(news);
    }

    [HttpPost]
    public IActionResult Edit(IFormFile? file, News news)
    {
        if (ModelState.IsValid)
        {
            string wwwrootPath = _webHostEnvironment.WebRootPath;
            if (file != null)
            {
                string fileName = Guid.NewGuid().ToString() +
Path.GetExtension(file.FileName);
                string newsPath = Path.Combine(wwwrootPath, @"images\ImagesOfNews");
                //Delete Old Images
                if (!string.IsNullOrEmpty(news.image))
                {
                    var oldImagePath = Path.Combine(wwwrootPath,
news.image.TrimStart('\'));
                    if (System.IO.File.Exists(oldImagePath))
                    {
                        System.IO.File.Delete(oldImagePath);
                    }
                }
                //Copy File to \img\Books
                using (var fileStream = new FileStream(Path.Combine(newsPath, fileName),

```

```

FileMode.Create))
    {
        file.CopyTo(fileStream);
    }
    //Update ImageUrl in DB
    news.image = @"images\ImagesOfNews\" + fileName;
}
_unitOfWork.NewsRepository.Update(news);
_unitOfWork.NewsRepository.Save();
return RedirectToAction("Index");
}
return View(news);
}

public IActionResult Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    News? news = _unitOfWork.NewsRepository.Get(c => c.Id == id);
    if (news == null)
    {
        return NotFound();
    }
    else
    {
        _unitOfWork.NewsRepository.Delete(news);
        _unitOfWork.NewsRepository.Save();
        return RedirectToAction("Index");
    }
}
}
}

```

Namespace Import:

- The controller imports necessary namespaces such as JobSeeking.Models, JobSeeking.Repository.IRepository, Microsoft.AspNetCore.Authorization, Microsoft.AspNetCore.Identity, and Microsoft.AspNetCore.Mvc.

Controller Attributes:

- [Area("Admin")]: Specifies that this controller belongs to the "Admin" area of the application.
- [Authorize(Roles = "Admin")]: Restricts access to the controller actions to users who are in the "Admin" role.

Constructor:

- The controller has a constructor that receives instances of `IUnitOfWork` and `userManager<ApplicationUser>` through dependency injection.

Index Action:

- `[HttpGet]` action method named `Index` retrieves a list of all users from the database using the `IUnitOfWork` repository.
- The list of users is passed to the corresponding view.

Delete Action:

- `[HttpPost]` action method named `Delete` deletes a user account based on the provided user ID.
- It retrieves the user from the `userManager<ApplicationUser>` using the provided user ID.
- If the user exists, it deletes the user using `userManager.DeleteAsync(user)` and redirects to the `Index` action.
- If the user does not exist, it returns a 404 Not Found response.

Lock Account Action:

- `[HttpPost]` action method named `LockAccount` toggles the `isValid` property of a user account between true and false.
- It retrieves the user from the `userManager<ApplicationUser>` using the provided user ID.
- If the user exists and their `isValid` property is false, it sets it to true (unlocking the account) and updates the user using `userManager.UpdateAsync(user)`.
- If the user exists and their `isValid` property is true, it sets it to false (locking the account) and updates the user.
- It then redirects to the `Index` action.
- If the user does not exist, it returns a 404 Not Found response.

JobSeeking.Areas.Admin.Controllers. ResponseCategoryController

```
using JobSeeking.Models;  
using JobSeeking.Repository.IRepository;  
using Microsoft.AspNetCore.Authorization;
```



```
using Microsoft.AspNetCore.Mvc;

namespace JobSeeking.Areas.Admin.Controllers
{
    [Area("Admin")]
    [Authorize(Roles = "Admin")]
    public class ResponseCategoryController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;
        public ResponseCategoryController(IUnitOfWork unitOfWork)
        {
            _unitOfWork = unitOfWork;
        }
        //Index category
        public IActionResult Index()
        {
            var categories = _unitOfWork.CategoryRepository.GetAllWithUser(c => c.isValid == false).ToList();
            return View(categories);
        }
        //Method accept category
        public IActionResult Accept( int? id)
        {
            if (id == null || id == 0)
            {
                return NotFound();
            }
            Category? category = _unitOfWork.CategoryRepository.Get(c => c.Id == id);
            if (category == null)
            {
                return NotFound();
            }
            if (ModelState.IsValid)
            {
                category.isValid = true;
                _unitOfWork.CategoryRepository.Update(category);
                _unitOfWork.CategoryRepository.Save();

                return RedirectToAction("Index");
            }
            return View(category);
        }
        //Method httpGet edit
        public IActionResult Edit(int? id)
        {
            if (id==null || id == 0)
            {
                return NotFound();
            }
            Category category = _unitOfWork.CategoryRepository.Get(c=>c.Id == id);
            if (category == null)
            {
                return NotFound();
            }
        }
    }
}
```

```

    }
    return View(category);
}
//Method http post edit
[HttpPost]
public IActionResult Edit(Category category)
{
    if (ModelState.IsValid)
    {
        _unitOfWork.CategoryRepository.Update(category);
        _unitOfWork.CategoryRepository.Save();
        return RedirectToAction("Index");
    }
    return View(category);
}
//Delete Category
public IActionResult Delete(int? id)
{
    if(id==null || id == 0)
    {
        return NotFound() ;
    }
    Category? category = _unitOfWork.CategoryRepository.Get(c=>c.Id==id);
    if (category == null)
    {
        return NotFound();
    }
    else
    {
        _unitOfWork.CategoryRepository.Delete(category);
        _unitOfWork.CategoryRepository.Save();
        return RedirectToAction("Index");
    }
}
}
}
}

```

Namespace Import:

- The code imports necessary namespaces such as JobSeeking.Models, JobSeeking.Repository.IRepository, Microsoft.AspNetCore.Authorization, and Microsoft.AspNetCore.Mvc.

Controller Attributes:

- [Area("Admin")]: Specifies that this controller belongs to the "Admin" area of the application.
- [Authorize(Roles = "Admin")]: Restricts access to the controller actions to users who are in the "Admin" role.

Constructor:

- The controller has a constructor that receives an instance of IUnitOfWork through dependency injection.

Index Action:

- [HttpGet] action method named Index retrieves a list of categories from the database where isValid is false using the IUnitOfWork repository.
- The list of categories is passed to the corresponding view.

Accept Action:

- [HttpPost] action method named Accept accepts a category ID as a parameter.
- It checks if the provided ID is valid (not null and not zero).
- It retrieves the category object from the database using the provided ID.
- If the category exists, it updates its isValid property to true and saves the changes using the IUnitOfWork repository.
- Finally, it redirects to the Index action to display the updated list of categories.

2.2.2. Employer:

JobSeeking.Areas.Employer.Controllers.CategoryController

```
using JobSeeking.Models;
using JobSeeking.Repository.IRepository;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
namespace JobSeeking.Areas.Employer.Controllers
{
    [Area("Employer")]
    [Authorize(Roles = "Employer")]
    public class CategoryController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly UserManager<ApplicationUser> _userManager;
        public CategoryController(IUnitOfWork unitOfWork, UserManager<ApplicationUser>
userManager)
        {
            _unitOfWork = unitOfWork;
        }
    }
}
```

```

        _userManager = userManager;
    }
    public IActionResult Index()
    {
        List<Category> myList = _unitOfWork.CategoryRepository.GetAll().ToList();
        return View(myList);
    }
    public IActionResult Create()
    {
        return View();
    }
    [HttpPost]
    public async Task<IActionResult> Create(Category category)
    {
        if (ModelState.IsValid)
        {
            var currentUser = await _userManager.GetUserAsync(User);
            if (currentUser != null) {
                category.EmployerId = currentUser.Id;
                category.IsValid = false;
                _unitOfWork.CategoryRepository.Add(category);
                _unitOfWork.Save();
                return RedirectToAction("Index");
            }
            else
            {
                return RedirectToAction("Index");
            }
        }
        return View(category);
    }
    public async Task<IActionResult> ToggleNotification(int id)
    {
        if (id == null)
        {
            return NotFound();
        }
        Category? category = _unitOfWork.CategoryRepository.Get(c => c.Id == id);
        if (category == null)
        {
            return NotFound();
        }

        category.categoryValid = !category.categoryValid;
        _unitOfWork.Save();
    }
    /*
    TempData["Success"] = "Notification status delete successfully!";*/
    return RedirectToAction("Index");
}
}
}

```

Namespace Import:

- The code imports necessary namespaces such as JobSeeking.Models, JobSeeking.Repository.IRepository, Microsoft.AspNetCore.Authorization, Microsoft.AspNetCore.Identity, and Microsoft.AspNetCore.Mvc.

Controller Attributes:

- [Area("Employer")]: Specifies that this controller belongs to the "Employer" area of the application.
- [Authorize(Roles = "Employer")]: Restricts access to the controller actions to users who are in the "Employer" role.

Constructor:

- The controller has a constructor that receives instances of IUnitOfWork and UserManager<ApplicationUser> through dependency injection.

Index Action:

- [HttpGet] action method named Index retrieves a list of categories from the database using the IUnitOfWork repository and passes it to the corresponding view.

Create Action:

- [HttpGet] action method named Create returns a view for creating a new category.
- [HttpPost] action method named Create accepts a Category object as a parameter.
- It checks if the model state is valid.
- If the model state is valid, it gets the current user using UserManager<ApplicationUser> and sets the EmployerId property of the category to the current user's ID.
- It sets the isValid property of the category to false, adds the category to the database using the IUnitOfWork repository, and saves the changes.
- Finally, it redirects to the Index action to display the updated list of categories.

JobSeeking.Areas.Employer.Controllers.JobController

```
using JobSeeking.Models;
using JobSeeking.Models.ViewModels;
using JobSeeking.Repository.IRepository;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Linq.Expressions;

namespace JobSeeking.Areas.Employer.Controllers
{
    [Area("Employer")]
    [Authorize(Roles = "Employer")]
    public class JobController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly IWebHostEnvironment _webHostEnvironment;

        public JobController(IUnitOfWork unitOfWork, UserManager<ApplicationUser>
userManager, IWebHostEnvironment webHostEnvironment)
        {
            _unitOfWork = unitOfWork;
            _userManager = userManager;
            _webHostEnvironment = webHostEnvironment;
        }
        public async Task<ActionResult> Index()
        {
            var currentUser = await _userManager.GetUserAsync(User);
            var jobs = _unitOfWork.JobRepository.GetAll().Where(j => j.EmployerId ==
currentUser.Id).ToList();
            foreach (var job in jobs)
            {
                var categories = new List<string>();
                foreach (var categoryIdString in job.Category)
                {
                    int categoryId = int.Parse(categoryIdString);
                    var categoryName =
_unitOfWork.CategoryRepository.GetCategoryNameById(categoryId);
                    if (categoryName != null)
                    {
                        categories.Add(categoryName);
                    }
                }
                job.Category = categories.ToArray();
            }
            return View(jobs);
        }
        public IActionResult Create()
        {
            JobSeekingVM jobSeekingVM = new JobSeekingVM
```

```

        {
            Categories = _unitOfWork.CategoryRepository.GetAll().Where(c =>
c.isValid).Select(c => new SelectListItem()
            {
                Text = c.Name,
                Value = c.Id.ToString(),
            })
        };

        return View(jobSeekingVM);
    }

    [HttpPost]
    public async Task<IActionResult> Create(JobSeekingVM jobSeeking)
    {
        if (ModelState.IsValid)
        {
            var currentUser = await _userManager.GetUserAsync(User);
            if (currentUser != null)
            {
                jobSeeking.Job.EmployerId = currentUser.Id;
                jobSeeking.Job.Logo = currentUser.Avatar;
                jobSeeking.Job.CompanyName = currentUser.Company;
                _unitOfWork.JobRepository.Add(jobSeeking.Job);
                _unitOfWork.Save();
                return RedirectToAction("Index");
            }
        }
        return View(jobSeeking);
    }

    public IActionResult Edit(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        JobSeekingVM jobSeekingVM = new JobSeekingVM()
        {
            Categories = _unitOfWork.CategoryRepository.GetAll().Where(c =>
c.isValid).Select(c => new SelectListItem()
            {
                Text = c.Name,
                Value = c.Id.ToString(),
            }
        ),
            Job = _unitOfWork.JobRepository.Get(c => c.Id == id)
        };
        jobSeekingVM.Categories = _unitOfWork.CategoryRepository.GetAll().Where(c =>
c.isValid).Select(c => new SelectListItem()
        {
            Text = c.Name,
            Value = c.Id.ToString(),
        }
    );
    }
    
```

```

    });
    return View(jobSeekingVM);
}

[HttpPost]
public async Task<IActionResult> Edit(JobSeekingVM jobSeeking)
{
    if (ModelState.IsValid)
    {
        _unitOfWork.JobRepository.Update(jobSeeking.Job);
        _unitOfWork.JobRepository.Save();
        return RedirectToAction("Index");
    }
    return View(jobSeeking);
}
public IActionResult Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    Job? job = _unitOfWork.JobRepository.Get(c => c.Id == id);
    if (job == null)
    {
        return NotFound();
    }
    else
    {
        _unitOfWork.JobRepository.Delete(job);
        _unitOfWork.JobRepository.Save();
        return RedirectToAction("Index");
    }
}
public IActionResult ViewAllCV(int? id, bool? status)
{
    if (id == null || id == 0)
    {
        return NotFound();
    }
    Job job = _unitOfWork.JobRepository.Get(j => j.Id == id);
    job.amountOfCV = 0;
    _unitOfWork.JobRepository.Update(job);
    _unitOfWork.JobRepository.Save();
    List<ApplyCV> mylist = _unitOfWork.ApplyCVRepository.GetAll().ToList();
    List<ApplyCV> applyCVs = mylist.Where(cv=>cv.JobId==job.Id).ToList();
    if (status.HasValue)
    {
        if (status == true)
        {
            applyCVs = mylist.Where(cv => cv.JobId == job.Id && cv.Status ==
true).ToList();
        }
    }
}

```



```

        else
        {
            applyCVs = mylist.Where(cv => cv.JobId == job.Id && cv.Status ==
false).ToList();
        }
    }
    else
    {
        applyCVs = mylist.Where(cv => cv.JobId == job.Id && cv.Status ==
null).ToList();
    }

    return View(applyCVs);
}
public IActionResult Detail(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    ApplyCV? cv = _unitOfWork.ApplyCVRepository.Get(c=>c.Id == id);
    if (cv==null)
    {
        return NotFound();
    }
    return View(cv);
}
public IActionResult DownloadCV(string cvPath)
{
    var filePath = Path.Combine(_webHostEnvironment.WebRootPath,
cvPath.TrimStart('\\', '/'));

    if (!System.IO.File.Exists(filePath))
    {
        return NotFound();
    }
    var fileContent = System.IO.File.ReadAllBytes(filePath);
    return File(fileContent, "application/pdf", Path.GetFileName(filePath));
}
public IActionResult AcceptCV(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    ApplyCV? applyCV = _unitOfWork.ApplyCVRepository.Get(c=>c.Id==id);
    if (applyCV == null)
    {
        return NotFound();
    }
    applyCV.Status = true;
    _unitOfWork.ApplyCVRepository.Update(applyCV);
    _unitOfWork.ApplyCVRepository.Save();
}

```

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

    public async Task< IActionResult> DetailOfJobSeeker(int? id)
    {
        if (id==null)
        {
            return NotFound();
        }
        var emailUser = _unitOfWork.ApplyCVRepository.Get(c => c.Id == id);
        var detailUser = await _userManager.FindByEmailAsync(emailUser.JobSeekerEmail);
        if (detailUser==null)
        {
            return NotFound();
        }
        return View(detailUser);
    }
}
```

Constructor

- The JobController class has a constructor that injects instances of IUnitOfWork, UserManager<ApplicationUser>, and IWebHostEnvironment.

Index Action

- The Index action retrieves a list of jobs belonging to the currently logged-in employer and populates their associated categories for display in the view.

Create Action

- The Create action returns a view to create a new job and accepts a POST request to create the job in the database if the model state is valid.

Edit Action

- The Edit action retrieves a job by ID and returns a view to edit its details. It accepts a POST request to update the job in the database.

Delete Action

- The Delete action deletes a job by ID from the database.

ViewAllCV Action

- The ViewAllCV action retrieves all CVs associated with a specific job for display in the view. It allows filtering CVs based on their status.

Detail Action

- The Detail action retrieves details of a specific CV by ID for display in the view.

DownloadCV Action

- The DownloadCV action allows users to download CV files by providing the file path.

AcceptCV Action

- The AcceptCV action updates the status of a CV to "accepted."

DetailOfJobSeeker Action

- The DetailOfJobSeeker action retrieves details of the job seeker who applied for a specific job by their email address.

Namespace Import

- The code imports necessary namespaces such as JobSeeking.Models, JobSeeking.Models.ViewModels, JobSeeking.Repository.IRepository, Microsoft.AspNetCore.Authorization, Microsoft.AspNetCore.Hosting, Microsoft.AspNetCore.Identity, and Microsoft.AspNetCore.Mvc.

Controller Attributes

- [Area("Employer")]: Specifies that this controller belongs to the "Employer" area of the application.
- [Authorize(Roles = "Employer")]: Restricts access to the controller actions to users who are in the "Employer" role.

2.2.3. JobSeeker:

JobSeeking.Areas.JobSeeker.Controllers.ApplyCVController

```
using JobSeeking.Models;  
using JobSeeking.Models.ViewModels;  
using JobSeeking.Repository.IRepository;  
using Microsoft.AspNetCore.Authorization;
```

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace JobSeeking.Areas.JobSeeker.Controllers
{
    [Area("JobSeeker")]
    [Authorize(Roles = "JobSeeker")]
    public class ApplyCVController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly IWebHostEnvironment _webHostEnvironment;
        public ApplyCVController(IUnitOfWork unitOfWork, UserManager<ApplicationUser>
userManager, IWebHostEnvironment webHostEnvironment)
        {
            _unitOfWork = unitOfWork;
            _userManager = userManager;
            _webHostEnvironment = webHostEnvironment;
        }

        public IActionResult Index()
        {
            var jobs = _unitOfWork.JobRepository.GetAll().ToList();
            foreach (var job in jobs)
            {
                var categories = new List<string>();
                foreach (var categoryIdString in job.Category)
                {
                    int categoryId = int.Parse(categoryIdString);
                    var categoryName =
_unitOfWork.CategoryRepository.GetCategoryNameById(categoryId);
                    if (categoryName != null)
                    {
                        categories.Add(categoryName);
                    }
                }
                job.Category = categories.ToArray();
            }
            return View(jobs);
        }

        public IActionResult Create(int? id)
        {
            if (id == null || id == 0)
            {
                return NotFound();
            }

            var job = _unitOfWork.JobRepository.Get(c => c.Id == id);
            if (job == null)
            {
                return NotFound();
            }
        }
    }
}
```

```

        return NotFound();
    }
    var jobSeekingVM = new JobSeekingVM();
    jobSeekingVM.applyCV = new ApplyCV();
    jobSeekingVM.applyCV.JobId = job.Id;
    return View(jobSeekingVM);
}

[HttpPost]
public async Task<IActionResult> Create(JobSeekingVM jobSeekingVM, IFormFile file)
{
    if (ModelState.IsValid)
    {
        var currentUser = await _userManager.GetUserAsync(User);
        if (currentUser != null)
        {
            jobSeekingVM.applyCV.JobSeekerEmail = currentUser.Email;

            if (file != null && file.Length > 0)
            {
                string wwwrootPath = _webHostEnvironment.WebRootPath;
                string fileName = Guid.NewGuid().ToString() +
Path.GetExtension(file.FileName);
                string cvPath = Path.Combine(wwwrootPath, @"images\CV");

                using (var fileStream = new FileStream(Path.Combine(cvPath,
fileName), FileMode.Create))
                {
                    await file.CopyToAsync(fileStream);
                }

                jobSeekingVM.applyCV.CV = @"images\CV\" + fileName;
            }
            _unitOfWork.ApplyCVRepository.Add(jobSeekingVM.applyCV);
            _unitOfWork.ApplyCVRepository.Save();
            return RedirectToAction("Index");
        }
    }

    return View(jobSeekingVM.applyCV);
}

public async Task<IActionResult> ListCV()
{
    var currentUser = await _userManager.GetUserAsync(User);
    List<ApplyCV> myList = _unitOfWork.ApplyCVRepository.GetAll("Job").Where(c =>
c.JobSeekerEmail == currentUser.Email).ToList();
    return View(myList);
}

public IActionResult ViewDetailOfJob(int? id)
{
    if (id == null)
    {

```

```

        return NotFound();
    }

    JobSeekingVM jobSeekingVM = new JobSeekingVM()
    {
        Categories = _unitOfWork.CategoryRepository.GetAll().Where(c =>
c.isValid).Select(c => new SelectListItem()
        {
            Text = c.Name,
            Value = c.Id.ToString(),
        }),
        Job = _unitOfWork.JobRepository.Get(c => c.Id == id)
    };
    jobSeekingVM.Categories = _unitOfWork.CategoryRepository.GetAll().Where(c =>
c.isValid).Select(c => new SelectListItem()
    {
        Text = c.Name,
        Value = c.Id.ToString(),
    });
    return View(jobSeekingVM);
}

public IActionResult Delete(int? id)
{
    if (id == null || id == 0)
    {
        return NotFound();
    }
    ApplyCV? applyCV = _unitOfWork.ApplyCVRepository.Get(c => c.Id == id);
    if (applyCV == null)
    {
        return NotFound();
    }
    else
    {
        _unitOfWork.ApplyCVRepository.Delete(applyCV);
        _unitOfWork.ApplyCVRepository.Save();
        return RedirectToAction("Index");
    }
}
}
}

```

Namespace Import:

- The code imports necessary namespaces such as JobSeeking.Models, JobSeeking.Models.ViewModels, JobSeeking.Repository.IRepository, Microsoft.AspNetCore.Authorization, Microsoft.AspNetCore.Hosting, Microsoft.AspNetCore.Identity, and Microsoft.AspNetCore.Mvc.

Controller Attributes:

- [Area("JobSeeker")]: Specifies that this controller belongs to the "JobSeeker" area of the application.
- [Authorize(Roles = "JobSeeker")]: Restricts access to the controller actions to users who are in the "JobSeeker" role.

Constructor:

- The controller has a constructor that receives instances of IUnitOfWork, UserManager<ApplicationUser>, and IWebHostEnvironment through dependency injection.

Index Action:

- [HttpGet] action method named Index retrieves a list of all jobs from the database using the IUnitOfWork repository and passes it to the corresponding view.
- It also retrieves and populates the categories associated with each job.

Create Action:

- [HttpGet] action method named Create returns a view for creating a new CV application for a specific job.
- [HttpPost] action method named Create accepts a JobSeekingVM object and an IFormFile object as parameters.
- It checks if the model state is valid.
- If the model state is valid, it gets the current user using UserManager<ApplicationUser> and sets the JobSeekerEmail property of the CV application to the current user's email.
- It uploads the CV file to the server and saves its path to the CV property of the CV application.
- It adds the CV application to the database using the IUnitOfWork repository and saves the changes.

ListCV Action:

- [HttpGet] action method named ListCV retrieves all CV applications submitted by the current user from the database using the IUnitOfWork repository and passes them to the corresponding view.

ViewDetailOfJob Action:

- [HttpGet] action method named ViewDetailOfJob retrieves details of a specific job by its ID from the database using the IUnitOfWork repository and passes them to the corresponding view.
- It also retrieves and populates the categories associated with the job.

Delete Action:

- [HttpGet] action method named Delete deletes a CV application by its ID from the database using the IUnitOfWork repository.

JobSeeking.Areas.JobSeeker.Controllers.HomeController

```
using JobSeeking.Models;
using JobSeeking.Repository.IRepository;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;

namespace JobSeeking.Areas.JobSeeker.Controllers
{
    [Area("JobSeeker")]
    [Authorize(Roles = "Admin,JobSeeker,Employer")]

    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        private readonly IUnitOfWork _unitOfWork;

        public HomeController(ILogger<HomeController> logger, IUnitOfWork unitOfWork)
        {
            _logger = logger;
            _unitOfWork = unitOfWork;
        }

        public IActionResult Index()
        {
            List<News> news = _unitOfWork.NewsRepository.GetAll().ToList();
            return View(news);
        }

        public IActionResult ShowNews(int? id)
```



```
{
    if (id==null)
    {
        return NotFound();
    }
    var news = _unitOfWork.NewsRepository.Get(c=>c.Id==id);
    return View(news);
}
public IActionResult Privacy()
{
    return View();
}

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

HomeController

- This controller handles requests related to the home page and news display for job seekers.
- It belongs to the "JobSeeker" area of the application.
- Authorized users with roles "Admin," "JobSeeker," or "Employer" can access its actions.

Constructor:

- Injects ILogger<HomeController> and IUnitOfWork dependencies into the controller.

Index Action:

- Retrieves a list of news articles from the repository using `_unitOfWork.NewsRepository.GetAll().ToList()` and passes it to the corresponding view.
- Displays the list of news articles on the home page.

ShowNews Action:

- Accepts an optional id parameter representing the ID of a specific news article.

- Retrieves the news article corresponding to the provided ID from the repository using `_unitOfWork.NewsRepository.Get(c => c.Id == id)`.
- Displays the details of the news article on a dedicated view.

Privacy Action:

- Renders the privacy policy view.

Error Action:

- Handles errors in the application and displays an error view, including a unique request identifier for tracking purposes.

AboutUs Action:

- Renders the "About Us" view.

2.3. View:

2.3.1. Admin:

JobSeeking.Areas.Admin.Views.Account.Details

```
@using Microsoft.AspNetCore.Identity
@model ApplicationUser
@inject UserManager<ApplicationUser> UserManager
<style>
    body {
        height: 200px;
        background-color: #E1D5E7; /* Light pastel purple */
        background-image: linear-gradient(to right, #E1D5E7, #F8BBD0, #BBDEFB); /* Pastel
purple to pastel pink to pastel blue */
    }
</style>
<br />
<br />
<div class="row">
    @{
        var userRoles = await UserManager.GetRolesAsync(@Model);
    }
    <if (userRoles.Contains("Employer") || userRoles.Contains("JobSeeker"))
    {
        <div class="col-lg-4 col-md-6 mb-4">
            <form>
                <div class="form-group">
                    
                </div>
            </form>
        </div>
    }
</if>
</div>
```

```

        </div>
        <div class="form-group">
            <label>Email:</label>
            <input type="text" class="form-control" value="@Model.Email" readonly>
        </div>
        <div class="form-group">
            <label>Name:</label>
            <input type="text" class="form-control" value="@Model.Name" readonly>
        </div>
        @if (!string.IsNullOrEmpty(@Model.Company))
        {
            <div class="form-group">
                <label>Company:</label>
                <input type="text" class="form-control" value="@Model.Company" readonly>
            </div>
        }
        @if (userRoles.Contains("Employer"))
        {
            <p class="text-success">Employer</p>
        }
        else if (userRoles.Contains("JobSeeker"))
        {
            <p class="text-primary">Job Seeker</p>
        }
        @if (@Model.isValid)
        {
            <p class="text-success">Valid</p>
        }
        else
        {
            <p class="text-danger">Invalid</p>
        }
        <div class="form-group">
            <a asp-action="Delete" asp-route-Id="@Model.Id" class="btn btn-danger"
onclick="return confirm('Are you sure you want to delete user:@Model.Email');"><i class="bi
bi-trash3"></i> </a>
            <a asp-action="LockAccount" asp-route-Id="@Model.Id" class="btn btn-
warning" onclick="return confirm('Are you sure you want to Lock & Unlock
user:@Model.Email');"><i class="bi bi-lock"></i> </a>
        </div>
    </form>
</div>
}
</div>

```

Imports and Model Declaration:

- The @using Microsoft.AspNetCore.Identity directive imports the Microsoft.AspNetCore.Identity namespace.
- The @model ApplicationUser directive specifies the model type for this view.

- The @inject UserManager<ApplicationUser> UserManager directive injects the UserManager<ApplicationUser> dependency into the view.

Styling:

- The <style> section contains CSS styles for the background color and gradient.

Body:

- The
 tags create line breaks for spacing.
- The <div class="row"> element starts a row in a grid layout.

User Roles and Form:

- The userRoles variable retrieves the roles of the current user using the injected UserManager.
- The @if statement checks if the user has roles "Employer" or "JobSeeker" and proceeds to display user information if true.

Inside the form:

- User avatar, email, name, and company (if available) are displayed in read-only input fields.
- The user's role ("Employer" or "Job Seeker") is displayed in text with appropriate styling.
- The user's validity status is displayed in text with appropriate styling.
- Two buttons are provided for deleting and locking/unlocking the user account.
- The delete button triggers a confirmation prompt.
- The lock/unlock button also triggers a confirmation prompt.

JobSeeking.Areas.Admin.Views.Account.Index

```
@using Microsoft.AspNetCore.Identity
@inject UserManager<ApplicationUser> UserManager
@model List<ApplicationUser>
<h1>User</h1>
<a asp-area="Identity" asp-page="/Account/EmployerRegister" class="btn btn-primary">Create
Employer Account</a>
<link href="~/css/indexaccount.css" rel="stylesheet" />
<style>
    body {
```

```

        height: 200px;
        background-color: #E1D5E7; /* Light pastel purple */
        background-image: linear-gradient(to right, #E1D5E7, #F8BBD0, #BBDEFB); /* Pastel
purple to pastel pink to pastel blue */
    }
</style>
<br />
<br />
<div class="row">
    @foreach (var obj in Model)
    {
        var userRoles = await UserManager.GetRolesAsync(obj);
        if (userRoles.Contains("Employer") || userRoles.Contains("JobSeeker"))
        {
            <div class="card">
                <div class="card__img">
                    <svg width="100%" xmlns="http://www.w3.org/2000/svg">
                        <rect height="450" width="540" fill="#ffffff"></rect>
                        <defs>
                            <linearGradient gradientTransform="rotate(222,648,379)"
                                y2="100%"
                                y1="0"
                                x2="0"
                                x1="0"
                                gradientUnits="userSpaceOnUse"
                                id="a">
                                <stop stop-color="#ffffff" offset="0"></stop>
                                <stop stop-color="#FC726E" offset="1"></stop>
                            </linearGradient>
                            <pattern viewBox="0 0 1080 900"
                                y="0"
                                x="0"
                                height="250"
                                width="300"
                                id="b"
                                patternUnits="userSpaceOnUse">
                                <g fill-opacity="0.5">
                                    <polygon points="90 150 0 300 180 300"
                                        fill="#444"></polygon>
                                    <polygon points="90 150 180 0 0 0"></polygon>
                                    <polygon points="270 150 360 0 180 0"
                                        fill="#AAA"></polygon>
                                    <polygon points="450 150 360 300 540 300"
                                        fill="#DDD"></polygon>
                                    <polygon points="450 150 540 0 360 0"
                                        fill="#999"></polygon>
                                    <polygon points="630 150 540 300 720 300"></polygon>
                                    <polygon points="630 150 720 0 540 0"
                                        fill="#DDD"></polygon>
                                    <polygon points="810 150 720 300 900 300"
                                        fill="#444"></polygon>
                                    <polygon points="810 150 900 0 720 0"
                                        fill="#FFF"></polygon>
                                </g>
                            </pattern>
                        </defs>
                    </svg>
                </div>
            </div>
        }
    }

```

```

fill="#DDD"></polygon>
fill="#444"></polygon>
fill="#DDD"></polygon>
fill="#666"></polygon>
fill="#AAA"></polygon>
fill="#DDD"></polygon>
fill="#999"></polygon>
fill="#999"></polygon>
fill="#FFF"></polygon>
fill="#DDD"></polygon>
fill="#AAA"></polygon>
fill="#444"></polygon>
fill="#222"></polygon>
fill="#DDD"></polygon>
fill="#444"></polygon>
fill="#AAA"></polygon>
fill="#666"></polygon>
fill="#999"></polygon>
fill="#999"></polygon>
fill="#444"></polygon>
fill="#FFF"></polygon>
fill="#222"></polygon>
fill="#FFF"></polygon>

<polygon points="990 150 900 300 1080 300">
<polygon points="990 150 1080 0 900 0">
<polygon points="90 450 0 600 180 600">
<polygon points="90 450 180 300 0 300"></polygon>
<polygon points="270 450 180 600 360 600">
<polygon points="270 450 360 300 180 300">
<polygon points="450 450 360 600 540 600">
<polygon points="450 450 540 300 360 300">
<polygon points="630 450 540 600 720 600">
<polygon points="630 450 720 300 540 300">
<polygon points="810 450 720 600 900 600"></polygon>
<polygon points="810 450 900 300 720 300">
<polygon points="990 450 900 600 1080 600">
<polygon points="990 450 1080 300 900 300">
<polygon points="90 750 0 900 180 900">
<polygon points="270 750 180 900 360 900"></polygon>
<polygon points="270 750 360 600 180 600">
<polygon points="450 750 540 600 360 600"></polygon>
<polygon points="630 750 540 900 720 900"></polygon>
<polygon points="630 750 720 600 540 600">
<polygon points="810 750 720 900 900 900">
<polygon points="810 750 900 600 720 600">
<polygon points="990 750 900 900 1080 900">
<polygon points="180 0 90 150 270 150">
<polygon points="360 0 270 150 450 150">
<polygon points="540 0 450 150 630 150">
<polygon points="900 0 810 150 990 150"></polygon>
<polygon points="0 300 -90 450 90 450">
<polygon points="0 300 90 150 -90 150">
<polygon points="180 300 90 450 270 450">

```

```

fill="#FFF"></polygon>
fill="#666"></polygon>
fill="#222"></polygon>
fill="#FFF"></polygon>
fill="#444"></polygon>
fill="#222"></polygon>
fill="#AAA"></polygon>
fill="#666"></polygon>
fill="#FFF"></polygon>
fill="#999"></polygon>

fill="#666"></polygon>
fill="#AAA"></polygon>
fill="#444"></polygon>
fill="#444"></polygon>
fill="#999"></polygon>
fill="#666"></polygon>
fill="#222"></polygon>
fill="#FFF"></polygon>
fill="#222"></polygon>
fill="#DDD"></polygon>
fill="#444"></polygon>
fill="#FFF"></polygon>
fill="#AAA"></polygon>
fill="#FFF"></polygon>
fill="#222"></polygon>
fill="#222"></polygon>

<polygon points="180 300 270 150 90 150"
<polygon points="360 300 270 450 450 450"
<polygon points="360 300 450 150 270 150"
<polygon points="540 300 450 450 630 450"
<polygon points="540 300 630 150 450 150"
<polygon points="720 300 630 450 810 450"
<polygon points="720 300 810 150 630 150"
<polygon points="900 300 810 450 990 450"
<polygon points="900 300 990 150 810 150"

<polygon points="0 600 -90 750 90 750"></polygon>
<polygon points="0 600 90 450 -90 450"

<polygon points="180 600 90 750 270 750"
<polygon points="180 600 270 450 90 450"
<polygon points="360 600 270 750 450 750"
<polygon points="360 600 450 450 270 450"
<polygon points="540 600 630 450 450 450"
<polygon points="720 600 630 750 810 750"
<polygon points="900 600 810 750 990 750"
<polygon points="900 600 990 450 810 450"

<polygon points="0 900 90 750 -90 750"
<polygon points="180 900 270 750 90 750"
<polygon points="360 900 450 750 270 750"
<polygon points="540 900 630 750 450 750"
<polygon points="720 900 810 750 630 750"
<polygon points="900 900 990 750 810 750"

<polygon points="1080 300 990 450 1170 450"
<polygon points="1080 300 1170 150 990 150"

```

```
fill="#FFF"></polygon>
                                <polygon points="1080 600 990 750 1170 750"></polygon>
                                <polygon points="1080 600 1170 450 990 450"
fill="#666"></polygon>
                                <polygon points="1080 900 1170 750 990 750"
fill="#DDD"></polygon>
                                </g>
                                </pattern>
                                </defs>
                                <rect height="100%" width="100%" fill="url(#a)" y="0" x="0"></rect>
                                <rect height="100%" width="100%" fill="url(#b)" y="0" x="0"></rect>
                                </svg>
</div>
<div class="card__avatar">
    
</div>
<div class="card__title">
    Email: @obj.Email
    Name: @obj.Name
</div>
@if (userRoles.Contains("Employer"))
{
    <div class="card__subtitle">Employer</div>
}
else if (userRoles.Contains("JobSeeker"))
{
    <div class="card__subtitle">Job Seeker</div>
}

<div class="card__wrapper">
    <a asp-action="Details" asp-route-Id="@obj.Id"
class="card__btn">Detail</a>
</div>
</div>
    }
}
</div>
```

Imports and Model Declaration:

- The @using Microsoft.AspNetCore.Identity directive imports the Microsoft.AspNetCore.Identity namespace.
- The @inject UserManager<ApplicationUser> UserManager directive injects the UserManager<ApplicationUser> dependency into the view.
- The @model List<ApplicationUser> directive specifies that the model for this view is a list of ApplicationUser objects.

Styling:

- The <style> section contains CSS styles for the background color and gradient.

Body:

- The
 tags create line breaks for spacing.
- The <div class="row"> element starts a row in a grid layout.

User Iteration and Display:

- The @foreach loop iterates over each ApplicationUser object in the model.
- Within the loop:
 - o The userRoles variable retrieves the roles of the current user.
 - o The @if statement checks if the user has roles "Employer" or "JobSeeker".
 - o If the user has relevant roles, a card is created to display user information:
 - The card includes a background gradient pattern and an avatar image.
 - User email and name are displayed.
 - The user's role ("Employer" or "Job Seeker") is displayed in the subtitle.
 - A "Detail" button is provided to navigate to the user's details page.

JobSeeking.Areas.Admin.Views.News.Create

```
@model News
<h1>Create News</h1>
<form method="post" enctype="multipart/form-data">
    <div class="form-group" >
        <label asp-for="@Model.Name"></label>
        <input asp-for="@Model.Name">
        <span asp-validation-for="@Model.Name" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="@Model.Description"></label>
        <input asp-for="@Model.Description">
        <span asp-validation-for="@Model.Description" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="@Model.image"></label>
        <input type="file" name="file" class="form-control">
    </div>
```

```
<button type="submit" class="btn btn-primary">Create</button>
<a asp-action="Index" class="btn btn-primary">Back to list</a>
</form>
@section Scripts {
    @{
        <partial name="_ValidationScriptsPartial" />
    }
}
```

Model Declaration:

- @model News: Specifies that the model for this view is of type News.

HTML Structure:

- The view starts with an <h1> tag displaying "Create News" as the heading.
- It includes a <form> element with the method set to "post" and the enctype set to "multipart/form-data" for uploading files.

Form Fields:

- Three form fields are included:
 - o Name: Displays a label and an input field for the news name.
 - o Description: Displays a label and an input field for the news description.
 - o Image: Displays a label and a file input field for uploading an image.

Validation:

- Validation messages are displayed below each input field using asp-validation-for to show any validation errors.
- If there are validation errors, they are displayed in red text.

Buttons:

- There are two buttons:
- "Create": Submits the form to create the news.
- "Back to list": Redirects the user to the index page without submitting the form.

Script Section:

- The @section Scripts directive includes validation scripts to handle client-side validation.

JobSeeking.Areas.Admin.Views.News.Edit

```
@model News
<h1>Edit News</h1>
<form method="post" enctype="multipart/form-data">
  <input hidden asp-for="@Model.Id" />
  <input hidden asp-for="@Model.image" />
  <div class="form-group">
    <label asp-for="@Model.Name"></label>
    <input asp-for="@Model.Name">
    <span asp-validation-for="@Model.Name" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="@Model.Description"></label>
    <input asp-for="@Model.Description">
    <span asp-validation-for="@Model.Description" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="@Model.image"></label>
    <input type="file" name="file" class="form-control">
  </div>
  <div class="col-2">
    
  </div>
  <button type="submit" class="btn btn-primary">Edit</button>
  <a asp-action="Index" class="btn btn-primary">Back to list</a>
</form>
```

Model Declaration:

- @model News: Specifies that the model for this view is of type News.

HTML Structure:

- The view starts with an <h1> tag displaying "Edit News" as the heading.
- It includes a <form> element with the method set to "post" and the enctype set to "multipart/form-data" for uploading files.

Hidden Inputs:

- Two hidden input fields are included:
 - o Id: Hidden input field to store the news item's ID.

- image: Hidden input field to store the news item's image URL.

Form Fields:

- Two form fields are included for editing:
 - Name: Displays a label and an input field for editing the news name.
 - Description: Displays a label and an input field for editing the news description.
- Additionally, there is a file input field for uploading a new image.

Current Image Display:

- An tag is included to display the current image associated with the news item.
- The src attribute is bound to the image property of the News model to display the current image.

Validation:

- Validation messages are displayed below each input field using asp-validation-for to show any validation errors.
- If there are validation errors, they are displayed in red text.

Buttons:

- There are two buttons:
 - "Edit": Submits the form to edit the news item.
 - "Back to list": Redirects the user to the index page without submitting the form.

JobSeeking.Areas.Admin.Views.News.Index

```
@model List<News>
<h1>List of News</h1>
<a asp-action="Create" class="btn btn-primary">Create</a>
<div class="card-container">
    @foreach (var newItem in Model)
    {
        <div class="card">
            @if (!string.IsNullOrEmpty(newItem.image))
            {
                
            }
        }
    }
</div>
```

```

    }
    <div class="card-body">
        <h2>@newsItem.Name</h2>
        <p>@newsItem.Description</p>
        <p>@newsItem.DateCreated.ToString("MMMM dd, yyyy")</p>
    </div>
    <a asp-area="Admin" asp-controller="News" asp-action="Edit" asp-route-
Id="@newsItem.Id" class="btn btn-primary">Edit</a>
    <a asp-area="Admin" asp-controller="News" asp-action="Delete" asp-route-
Id="@newsItem.Id" class="btn btn-danger"
        onclick="return confirm('Are you want to delete
user:@newsItem.Name')">Delete</a>
    </div>
}
</div>

```

Model Declaration:

- @model List<News>: Indicates that this view is strongly typed to a list of News objects.

HTML Structure:

- Starts with an <h1> tag displaying "List of News" as the heading.
- Includes a link to create a new news item using the Create action.

Card Container:

- Each news item is displayed within a card container (<div class="card-container">).

Looping Through News Items:

- Iterates over each News item in the model using a foreach loop.

For each news item:

- Displays an image if image property is not empty.
- Displays the news item's name (Name), description (Description), and creation date (DateCreated).
- Provides buttons to edit and delete the news item.

Edit Button:

- The "Edit" button links to the Edit action of the News controller in the Admin area, passing the news item's ID as a route parameter.

Delete Button:

- The "Delete" button links to the Delete action of the News controller in the Admin area, passing the news item's ID as a route parameter.
- It includes a JavaScript confirmation dialog to confirm the deletion action.

JobSeeking.Areas.Admin.Views. ResponseCategory.Edit

```
@model Category
<h1>Edit category @Model.Name</h1>
<form method="post">
    <input hidden asp-for="@Model.Id" />
    <input hidden asp-for="@Model.EmployerId" />
    <div class="form-group">
        <label asp-for="Name"></label>
        <input asp-for="Name">
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <button type="submit" class="btn btn-primary">Save</button>
</form>
```

Model and Heading:

- The @model Category directive specifies that the model for this view is a single Category object.
- The <h1> tag displays the heading "Edit category [Category Name]".

Form for Editing Category:

- The <form> tag sets up a form for editing the category details. The method="post" attribute indicates that the form will submit a POST request.
- Two hidden input fields are included for the Id and EmployerId properties of the Category model. These fields are hidden from the user interface but are necessary for identifying the category and its associated employer when submitting the form.
- Inside the form:
 - o A <div> with class form-group is used to group form elements together.
 - o A <label> element with asp-for="Name" attribute is displayed as a label for the category name input field.

- An <input> field with asp-for="Name" attribute allows the user to input the category name.
- A element with asp-validation-for="Name" attribute is used to display validation error messages for the category name input field, if any.

Save Button:

- The <button> element with type="submit" attribute is a submit button for saving the changes made to the category.
- The button has a class btn btn-primary for styling, indicating it as the primary action button.

JobSeeking.Areas.Admin.Views. ResponseCategory.Index

```
@model List<Category>
<h1>All Categories are waiting to respond </h1>
<div class="row">
  @foreach (var obj in Model)
  {
    @if (obj.isValid == false)
    {
      <div class="col-lg-4 col-md-6 mb-4">
        <div class="card">
          <div class="card-header">
            <h5> Name: @obj.Name</h5>
          </div>
          <div class="card-body">
            <h5>
              Valid:@obj.isValid
            </h5>
            <h5>
              Time Create: @obj.CreateDay
            </h5>
            <h5>
              Email:@obj.User.Email
            </h5>
          </div>
          <div class="card-footer">
            <a asp-area="Admin" asp-controller="ResponseCategory" asp-
action="Accept" asp-route-Id="@obj.Id" class="btn btn-primary"
onclick="return confirm('Are you want to accept
category:@obj.Name')">Accept</a>
            <a asp-area="Admin" asp-controller="ResponseCategory" asp-
action="Edit" asp-route-Id="@obj.Id" class="btn btn-primary">Edit</a>
            <a asp-area="Admin" asp-controller="ResponseCategory" asp-
```

```

action="Delete" asp-route-Id="@obj.Id" class="btn btn-primary"
                                onclick="return confirm('Are you want to delete
category:@obj.Name')">Delete</a>
                                </div>
                                </div>
                                </div>
                                }
                                }
                                </div>

```

Model and Heading:

- The @model List<Category> directive specifies that the model for this view is a list of Category objects.
- The <h1> tag displays the heading "All Categories are waiting to respond".

Iterating Over Categories:

- The <div class="row"> element starts a row in a grid layout to display categories.
- The @foreach loop iterates over each Category object in the model.
- Within the loop:
 - o The @if statement checks if the category's isValid property is false.
 - o If the category is waiting for a response:
 - A <div> with class col-lg-4 col-md-6 mb-4 sets up a column layout for the category card.
 - Inside the card:
 - The category's name (Name property) is displayed in the card header.
 - Details such as validity (isValid property), creation time (CreateDay property), and associated user's email (User.Email property) are displayed in the card body.
 - Action buttons (Accept, Edit, Delete) are provided in the card footer, each linking to corresponding actions in the ResponseCategory controller.

- JavaScript confirm dialogs are included for the "Accept" and "Delete" actions to confirm user intent.

JobSeeking.Areas.Employer.Views. Category.Create

```
@model Category
@using JobSeeking.Models
<h1>Create Category</h1>
<form method="post">
    <div class="form-group">
        <label asp-for="Name"></label>
        <input asp-for="Name">
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <button type="submit" class="btn btn-primary">Create</button>
</form>
@section Scripts {
    @{
        <partial name="_ValidationScriptsPartial" />
    }
}
```

Model and Heading:

- The @model Category directive specifies that the model for this view is a single Category object.
- The <h1> tag displays the heading "Create Category".

Form for Creating a New Category:

- The <form> tag sets up a form for creating a new category. The method="post" attribute indicates that the form will submit a POST request.
- Inside the form:
 - A <div> with class form-group is used to group form elements together.
 - A <label> element with asp-for="Name" attribute is displayed as a label for the category name input field.
 - An <input> field with asp-for="Name" attribute allows the user to input the category name.
 - A element with asp-validation-for="Name" attribute is used to display validation error messages for the category name input field, if any.

Create Button:

- The <button> element with type="submit" attribute is a submit button for creating the new category.
- The button has a class btn btn-primary for styling, indicating it as the primary action button.

Scripts Section:

- The @section Scripts { ... } block includes scripts or partial views related to client-side validation.
- In this case, the _ValidationScriptsPartial partial view is rendered, which contains JavaScript functions for client-side validation.

JobSeeking.Areas.Employer.Views. Category.Index

```
@model List<Category>
@using JobSeeking.Models
<h1>All Categories</h1>
<style>
    body {
        height: 200px;
        background-color: #E1D5E7; /* Light pastel purple */
        background-image: linear-gradient(to right, #E1D5E7, #F8BBD0, #BBDEFB); /*
Pastel purple to pastel pink to pastel blue */
    }
</style>
<a asp-action="Create" class="btn btn-primary">Create Category</a>
<button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-
target="#exampleModal">
    Notification
</button>
<div class="row">
    @foreach (var obj in Model)
    {
        @if (obj.IsValid == true)
        {
            <div class="col-lg-4 col-md-6 mb-4">
                <div class="card">
                    <div class="card-header">
                        <h5> Name: @obj.Name</h5>
                    </div>
                    <div class="card-body">
                        <h5>
                            Valid:@obj.IsValid
                        </h5>
                    </div>
                </div>
            </div>
        }
    }
</div>
```

```

        Time Create: @obj.CreateDay
    </h5>
</div>
<div class="card-footer">
</div>
</div>
</div>
}
}
</div>

<!-- Modal -->
<div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModalLabel"
aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h1 class="modal-title fs-5" id="exampleModalLabel">Modal title</h1>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
            </div>
            <div class="modal-body">
                @foreach (var category in Model.Where(c => c.isValid))
                {
                    @if (category.isValid && !category.categoryValid)
                    {
                        <div>
                            @if (!category.categoryValid)
                            {
                                <div class="notification-status">
                                    <form asp-controller="Category" asp-
action="ToggleNotification" method="post">
                                        Category <strong>@category.Name</strong> has been
confirmed.
                                        <input type="hidden" name="id" value="@category.Id"
/>
                                        <button type="submit" class="btn-close"></button>
                                    </form>
                                </div>
                            }
                        </div>
                    }
                }
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Close</button>
            </div>
        </div>
    </div>
</div>

<script>

```

```
function confirmSeen(categoryId) {  
    document.getElementById('notification-' + categoryId).style.display = 'none';  
}  
</script>
```

Model and Heading:

- The @model List<Category> directive specifies that the model for this view is a list of Category objects.
- The <h1> tag displays the heading "All Categories".

Styling:

- CSS styles are applied to the body to set the background color and image gradient.

Buttons:

- An anchor tag <a> with asp-action="Create" is a button to create a new category. It has the class btn btn-primary for styling.
- A button with type="button" is used to trigger a modal for notifications. It has the class btn btn-primary.

Category Display:

- Inside the <div class="row"> tag, a loop iterates through each category in the model.
- For each category:
 - o If the category is valid (isValid == true), it is displayed as a card with its name, validity status, and creation date.

Modal for Notifications:

- A modal is defined with an id exampleModal, which can be triggered by the notification button.
- Inside the modal:
- A loop iterates through each category in the model where isValid is true.
- If a category is valid but its notification status is not confirmed (categoryValid == false):

- A notification is displayed for the category, allowing the user to confirm it.
- The notification includes the category name and a button to close it.

JavaScript Function:

- A JavaScript function `confirmSeen(categoryId)` is defined to handle the confirmation of notifications. It hides the notification element when invoked.

JobSeeking.Areas.Employer.Views. Job.Create

```
@using JobSeeking.Models.ViewModels
@model JobSeekingVM

<h1>Create Job</h1>
<form asp-action="Create" method="post">
    <div class="form-group">
        <label asp-for="Job.Name" class="control-label"></label>
        <input asp-for="Job.Name" class="form-control" />
        <span asp-validation-for="Job.Name" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Job.Description" class="control-label"></label>
        <textarea asp-for="Job.Description" class="form-control"></textarea>
        <span asp-validation-for="Job.Description" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Job.Salary" class="control-label"></label>
        <input asp-for="Job.Salary" class="form-control" />
        <span asp-validation-for="Job.Salary" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Job.Category" class="col-sm-4 col-form-label">Category</label>
        <select asp-for="Job.Category" class="form-select" asp-items="@Model.Categories"
multiple></select>
        <span asp-validation-for="Job.Category" class="text-danger"></span>
    </div>
    <button type="submit" class="btn btn-primary">Create</button>
</form>
@section Scripts {
    @{
        <partial name="_ValidationScriptsPartial" />
    }
}
```

Model and Heading:

- The `@model JobSeekingVM` directive specifies that the model for this view is of type `JobSeekingVM`.

- The <h1> tag displays the heading "Create Job".

Form:

- The <form> tag specifies the action to be taken when the form is submitted, which is asp-action="Create" and the HTTP method is post.
- Inside the form, there are several <div class="form-group"> elements, each containing a form field:
 - o For the job name, a text input field is provided with the label "Name".
 - o For the job description, a textarea field is provided with the label "Description".
 - o For the job salary, a text input field is provided with the label "Salary".
 - o For the job category, a select dropdown field is provided with the label "Category". It allows selecting multiple categories and is populated with options from the Model.Categories.
- Each form field has an associated element to display validation error messages if any.

Submit Button:

- The submit button is a <button> element with the type "submit" and the class "btn btn-primary". It displays "Create" as the button text.

Script Section:

- Inside the @section Scripts { }, a partial view _ValidationScriptsPartial is included. This partial view likely contains scripts for client-side validation.

JobSeeking.Areas.Employer.Views. Job.Details

```
<link href="~/css/detailcv.css" rel="stylesheet" />
@model ApplyCV
<h1>Detail of CV</h1>
<form enctype="multipart/form-data">
    <div class="form-group">
        <label asp-for="@Model.JobSeekerEmail" class="control-label"></label>
        <input disabled asp-for="@Model.Description" class="form-control"></input>
    </div>
    <div class="form-group">
        <label asp-for="@Model.Description" class="control-label"></label>
```

```

        <textarea disabled asp-for="@Model.Description" class="form-control"></textarea>
    </div>
    <div class="form-group">
        <label asp-for="@Model.TimeCreate" class="control-label"></label>
        <input disabled asp-for="@Model.TimeCreate" class="form-control"></input>
    </div>
    <div class="form-group">
        <a class="Btn" asp-action="DownloadCV" asp-route-cvPath="@Model.CV">
            <svg class="svgIcon" viewBox="0 0 384 512" height="1em"
xmlns="http://www.w3.org/2000/svg"><path d="M169.4 470.6c12.5 12.5 32.8 12.5 45.3 0l160-
160c12.5-12.5 12.5-32.8 0-45.3s-32.8-12.5-45.3 0L224 370.8 224 64c0-17.7-14.3-32-32-32s-32
14.3-32 32l0 306.7L54.6 265.4c-12.5-12.5-32.8-12.5-45.3 0s-12.5 32.8 0 45.3l160
160z"></path></svg>
            <span class="icon2"></span>
            <span class="tooltip">Download CV</span>
        </a>

    </div>

    @if (Model.Status == false)
    {
        <a asp-area="Employer" asp-controller="Job" asp-action="AcceptCV" asp-route-
Id="@Model.Id">Accept</a>
    }

    <button class="learn-more" asp-area="Employer" asp-controller="Job" asp-
action="DetailOfJobSeeker" asp-route-Id="@Model.Id">
        <span class="circle" aria-hidden="true">
            <span class="icon arrow"></span>
        </span>
        <span class="button-text">Detail of Job Seeker</span>
    </button>

    <button class="button" asp-action="Index">
        <div class="button-box">
            <span class="button-elem">
                <svg viewBox="0 0 46 40" xmlns="http://www.w3.org/2000/svg">
                    <path d="M46 20.038c0-.7-.3-1.5-.8-2.1l-16-17c-1.1-1-3.2-1.4-4.4-.3-1.2
1.1-1.2 3.3 0 4.4l11.3 11.9H3c-1.7 0-3 1.3 3s1.3 3 3h33.1l-11.3 11.9c-1 1-1.2 3.3 0 4.4
1.2 1.1 3.3.8 4.4-.3l16-17c.5-.5.8-1.1.8-1.9z"></path>
                </svg>
            </span>
            <span class="button-elem">
                <svg viewBox="0 0 46 40">
                    <path d="M46 20.038c0-.7-.3-1.5-.8-2.1l-16-17c-1.1-1-3.2-1.4-4.4-.3-1.2
1.1-1.2 3.3 0 4.4l11.3 11.9H3c-1.7 0-3 1.3 3s1.3 3 3h33.1l-11.3 11.9c-1 1-1.2 3.3 0 4.4
1.2 1.1 3.3.8 4.4-.3l16-17c.5-.5.8-1.1.8-1.9z"></path>
                </svg>
            </span>
        </div>
    </button>

```

```
</form>
```

CSS Link:

- The <link> tag includes an external CSS file named "detailcv.css" to style the elements within this view.

Model Declaration:

- The @model ApplyCV directive specifies that the model for this view is of type ApplyCV.

Heading:

- The <h1> tag displays the heading "Detail of CV".

Form:

- The <form> tag is used to encapsulate the form elements.
- The enctype="multipart/form-data" attribute specifies how form data should be encoded when submitting it to the server.

Form Fields:

- Four form groups are defined, each containing a label and an input or textarea element:
 - o Job Seeker's Email: Displayed in a disabled input field.
 - o Description: Displayed in a disabled textarea field.
 - o Time Create: Displayed in a disabled input field.
 - o Download CV Button: A button that allows downloading the CV file associated with this application.
 - This button triggers the "DownloadCV" action in the "Employer" area controller "Job" and passes the CV path as a route parameter.
- Conditional Rendering:

- If the Model.Status is false, an "Accept" button is rendered. This button triggers the "AcceptCV" action in the "Employer" area controller "Job" and passes the CV ID as a route parameter.
- Buttons:
 - A button with class "learn-more" is rendered, which triggers the "DetailOfJobSeeker" action in the "Employer" area controller "Job" and passes the CV ID as a route parameter. It displays an arrow icon and text "Detail of Job Seeker".
 - Another button with class "button" triggers the "Index" action and displays a left arrow icon.

Script Section:

- Inside the @section Scripts { }, a partial view _ValidationScriptsPartial is included. This partial view likely contains scripts for client-side validation.

JobSeeking.Areas.Employer.Views. Job. DetailOfJobSeeker

```
@model ApplicationUser
<h1>Detail of Job Seeker</h1>
<form enctype="multipart/form-data">
  <div class="form-group">
    <label asp-for="@Model.Name" class="form-label"></label>
    <input asp-for="@Model.Name" class="form-control" disabled />
  </div>
  <div class="form-group">
    <label asp-for="@Model.PhoneNumber" class="control-label"></label>
    <input asp-for="@Model.PhoneNumber" class="form-control" disabled></input>
  </div>
  <div class="form-group">
    <label asp-for="@Model.Email" class="control-label"></label>
    <input asp-for="@Model.Email" class="form-control" disabled></input>
  </div>
  <div class="form-group">
    <label asp-for="@Model.PhoneNumber" class="control-label"></label>
    <input asp-for="@Model.PhoneNumber" class="form-control" disabled></input>
  </div>
  <div class="form-group">
    <label asp-for="@Model.Address" class="control-label"></label>
    <input asp-for="@Model.Address" class="form-control" disabled />
  </div>
  <div class="form-group">
    <label asp-for="@Model.City" class="control-label"></label>
    <input asp-for="@Model.City" class="form-control" disabled />
  </div>
</form>
```

```
<a asp-action="Index" class="btn btn-primary">Back to list</a>
</form>
```

Model Declaration:

- The @model ApplicationUser directive specifies that the model for this view is of type ApplicationUser.

Heading:

- The <h1> tag displays the heading "Detail of Job Seeker".

Form:

- The <form> tag is used to encapsulate the form elements.

Form Fields:

- Six form groups are defined, each containing a label and an input element:
 - o Name: Displays the user's name in a disabled input field.
 - o PhoneNumber: Displays the user's phone number in a disabled input field.
 - o Email: Displays the user's email address in a disabled input field.
 - o PhoneNumber (again): This seems like a duplicate of the phone number field, displaying the user's phone number again.
 - o Address: Displays the user's address in a disabled input field.
 - o City: Displays the user's city in a disabled input field.

Back Button:

- An anchor tag <a> is provided with the text "Back to list" and a CSS class "btn btn-primary". This button links to the "Index" action.

JobSeeking.Areas.Employer.Views. Job. Edit

```
@using JobSeeking.Models.ViewModels
@model JobSeekingVM

<h1>Edit Category</h1>
```

```
<form asp-action="Edit" method="post">
  <input hidden asp-for="Job.Id" />
  <input hidden asp-for="Job.EmployerId" />
  <div class="form-group">
    <label asp-for="Job.Name" class="control-label"></label>
    <input asp-for="Job.Name" class="form-control" />
    <span asp-validation-for="Job.Name" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="Job.Description" class="control-label"></label>
    <textarea asp-for="Job.Description" class="form-control"></textarea>
    <span asp-validation-for="Job.Description" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="Job.Salary" class="control-label"></label>
    <input asp-for="Job.Salary" class="form-control" />
    <span asp-validation-for="Job.Salary" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="Job.Category" class="col-sm-4 col-form-label">Category</label>
    <select asp-for="Job.Category" class="form-select" asp-items="@Model.Categories"
multiple></select>
    <span asp-validation-for="Job.Category" class="text-danger"></span>
  </div>
  <button type="submit" class="btn btn-primary">Edit</button>
</form>
```

Model Declaration:

- @model JobSeekingVM: Indicates that the model for this view is of type JobSeekingVM.

Heading:

- <h1>Edit Category</h1>: Displays the heading "Edit Category" at the top of the form.

Form:

- <form asp-action="Edit" method="post">: Defines a form that will submit to the "Edit" action with an HTTP POST request.

Hidden Input Fields:

- <input hidden asp-for="Job.Id" />: Hidden input field for the Id property of the Job object.
- <input hidden asp-for="Job.EmployerId" />: Hidden input field for the EmployerId property of the Job object.

Form Groups:

- Job Name:
 - Label: `<label asp-for="Job.Name" class="control-label"></label>`: Label for the job name input.
 - Input Field: `<input asp-for="Job.Name" class="form-control" />`: Text input field for the job name.
 - Validation Message: ``: Displays validation errors for the job name input.
- Job Description:
 - Label: `<label asp-for="Job.Description" class="control-label"></label>`: Label for the job description textarea.
 - Textarea: `<textarea asp-for="Job.Description" class="form-control"></textarea>`: Textarea input for the job description.
 - Validation Message: ``: Displays validation errors for the job description textarea.
- Job Salary:
 - Label: `<label asp-for="Job.Salary" class="control-label"></label>`: Label for the job salary input.
 - Input Field: `<input asp-for="Job.Salary" class="form-control" />`: Text input field for the job salary.
 - Validation Message: ``: Displays validation errors for the job salary input.
- Job Category:
 - Label: `<label asp-for="Job.Category" class="col-sm-4 col-form-label">Category</label>`: Label for the job category select element.

- Select Element: `<select asp-for="Job.Category" class="form-select" asp-items="@Model.Categories" multiple></select>`: Dropdown select input for the job category, populated with options from the Model.Categories collection.
- Validation Message: ``: Displays validation errors for the job category select element.

Submit Button:

- `<button type="submit" class="btn btn-primary">Edit</button>`: Button to submit the form with the changes made to the job details.

JobSeeking.Areas.Employer.Views. Job. Index

```
<link href="~/css/indexjob.css" rel="stylesheet" />
@using Microsoft.AspNetCore.Identity
@model List<Job>
@inject SignInManager<ApplicationUser> SignInManager
@{
    var currentUser = await SignInManager.UserManager.GetUserAsync(User);
}
<h1>Your Jobs</h1>

<a asp-action="Create" class="btn btn-primary"><i class="bi bi-plus"></i></a>
<button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-
target="#exampleModal">
    <i class="bi bi-bell"></i>
</button>
<div class="row">
    @foreach (var job in Model)
    {
        <div class="col-lg-4 col-md-6 mb-4">
            <div class="card">
                <div class="card-header">
                    <h5>@job.Name</h5>
                </div>
                <div class="card-body">
                    <p><strong>Description:</strong> @job.Description</p>
                    <p><strong>Salary:</strong> @job.Salary</p>
                    <p><strong>Company Name:</strong> @job.CompanyName</p>
                    <p><strong>Email:</strong> @job.User.Email</p>
                    
                    <strong>Categories:</strong>
                    @foreach (var categoryName in job.Category)
                    {
                        <span class="badge bg-primary">@categoryName</span>
                    }
                </div>
                <div class="card-footer">
```

```

        <a asp-area="Employer" asp-controller="Job" asp-action="Edit" asp-route-
Id="@job.Id" class="button">
            <i class="bi bi-pen"></i>
        </a>
        <a asp-area="Employer" asp-controller="Job" asp-action="Delete" asp-
route-Id="@job.Id" class="comic-button"
            onclick="return confirm('Are you want to delete
user:@job.Name')">
            <i class="bi bi-trash3"></i>
        </a>

        <a asp-area="Employer" asp-controller="Job" asp-action="ViewAllCV" asp-
route-Id="@job.Id" asp-route-status="null" as class="btn btn-warning"><i class="bi bi-
eye"></i></a>
        <a asp-area="Employer" asp-controller="Job" asp-action="ViewAllCV" asp-
route-Id="@job.Id" asp-route-status="true" as class="btn btn-success"><i class="bi bi-
eye"></i></a>
        <a asp-area="Employer" asp-controller="Job" asp-action="ViewAllCV" asp-
route-Id="@job.Id" asp-route-status="false" as class="btn btn-primary"><i class="bi bi-
eye"></i></a>
    </div>
</div>
}
</div>
<div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModalLabel"
aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h1 class="modal-title fs-5" id="exampleModalLabel">Modal title</h1>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
            </div>
            <div class="modal-body">
                @foreach(var obj in Model)
                {
                    @if (obj.amountOfCV != 0)
                    {
                        <p><strong>New cv:</strong>@obj.Name has @obj.amountOfCV new CV</p>
                    }else{
                        <p><strong>Doesn't have any new CV'</strong></p>
                    }
                }
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Close</button>
                <button type="button" class="btn btn-primary">Save changes</button>
            </div>
        </div>
    </div>
</div>

```

```
</div>
```

CSS Link:

- `<link href="~/css/indexjob.css" rel="stylesheet" />`: This line links the HTML file with a CSS file named "indexjob.css" located in the "~/css" directory.

Using Directives and Model Declaration:

- `@using Microsoft.AspNetCore.Identity`: This line imports the Identity framework namespace to use its classes and functionalities.
- `@model List<Job>`: This declares the model for the view as a list of Job objects.

User Authentication:

- `@inject SignInManager<ApplicationUser> SignInManager`: This injects the SignInManager service for managing user sign-in into the view.
- `var currentUser = await SignInManager.UserManager.GetUserAsync(User);`: This retrieves the current user using the SignInManager service.

HTML Content:

- `<h1>Your Jobs</h1>`: This is a heading indicating the title of the section.
- `<a asp-action="Create" class="btn btn-primary"><i class="bi bi-plus"></i>`: This creates a link button for creating a new job. It uses the asp-action tag helper to generate the URL based on the specified action.
- `<button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-target="#exampleModal"><i class="bi bi-bell"></i></button>`: This button triggers a modal window when clicked. It contains a bell icon.
- `<div class="row">...</div>`: This div contains a row of job cards displayed using a Bootstrap grid system.
- Inside the foreach loop:
 - `<div class="col-lg-4 col-md-6 mb-4">...</div>`: This div represents a column in the Bootstrap grid system.

- Inside each job card:
 - Job Details: Displays job information such as name, description, salary, company name, email, logo, and categories.
 - Action Buttons: Provides buttons for editing, deleting, and viewing CVs related to the job.
- Modal Window (<div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModalLabel" aria-hidden="true">...</div>): This is a Bootstrap modal used for displaying notifications about new CVs related to jobs.
 - Modal Header: Contains the title and a close button.
 - Modal Body: Displays information about new CVs associated with each job.
 - Modal Footer: Contains buttons for closing the modal and saving changes.

JobSeeking.Areas.Employer.Views. Job. ViewAllCV

```
@model List<ApplyCV>
<h1>List CV of your Job</h1>
<style>
    body {
        height: 200px;
        background-color: #E1D5E7; /* Light pastel purple */
        background-image: linear-gradient(to right, #E1D5E7, #F8BBD0, #BBDEFB); /*
Pastel purple to pastel pink to pastel blue */
    }
    .card-header{
        background-color:darkcyan;
    }
</style>
@foreach(var obj in Model )
{
    <div class="col-lg-4 col-md-6 mb-4">
        <div class="card">
            <div class="card-header">
                <h5>Name: @obj.JobSeekerEmail</h5>
            </div>
            <div class="card-body">
                <h5>
                    Time Create: @obj.TimeCreate
                    Description: @obj.Description
                </h5>
            </div>
        </div>
    </div>
}
```



```

        </div>
        <div class="card-footer">
            <a asp-action="Detail" asp-route-Id="@obj.Id" class="Btn btn-
primary">Detail of CV</a>
        </div>
    </div>
</div>
}

```

Model Declaration:

- @model List<ApplyCV>: This specifies that the model for the view is a list of ApplyCV objects.

HTML Content:

- <h1>List CV of your Job</h1>: This is a heading indicating the title of the section.
- <style>...</style>: This contains CSS styles to customize the appearance of the HTML elements.
- Inside the foreach loop:
 - o <div class="col-lg-4 col-md-6 mb-4">...</div>: This div represents a column in the Bootstrap grid system.
 - o Inside each job application card:
 - card-header: This class is used to style the header of the card, setting its background color to dark cyan.
 - Name, Time Create, and Description: These are properties of the ApplyCV model object (obj) displayed within the card.
 - Action Button: Provides a link button labeled "Detail of CV" that directs users to view the details of the specific CV.

JobSeeking.Areas.JobSeeker.Views. ApplyCV. Create

```

@using JobSeeking.Models.ViewModels
@model JobSeekingVM
<h1>Apply for Job</h1>
<form method="post" enctype="multipart/form-data">
    <input hidden asp-for="applyCV.JobId" />
    <div class="form-group">
        <label asp-for="applyCV.Description" class="control-label">Description:</label>
        <textarea asp-for="applyCV.Description" class="form-control"></textarea>
    </div>

```

```

        <span asp-validation-for="applyCV.Description" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="applyCV.CV">CV (PDF only):</label>
        <input type="file" name="file" class="form-control" />
        <span asp-validation-for="applyCV.CV" class="text-danger"></span>
    </div>
    <div class="button-section">
        <button type="submit">
            <span>
                <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" width="24"
height="24"><path fill="none" d="M0 0h24v24H0z"></path><path fill="currentColor" d="M11
11V5h2v6h6v2h-6v6h-2v-6H5v-2z"></path></svg> Create
            </span>
        </button>
        <button class="button" asp-area="JobSeeker" asp-controller="ApplyCV" asp-
action="Index">
            <div class="button-box">
                <span class="button-elem">
                    <svg viewBox="0 0 46 40" xmlns="http://www.w3.org/2000/svg">
                        <path d="M46 20.038c0-.7-.3-1.5-.8-2.1l-16-17c-1.1-1-3.2-1.4-4.4-.3-
1.2 1.1-1.2 3.3 0 4.4l11.3 11.9H3c-1.7 0-3 1.3-3 3s1.3 3 3 3h33.1l-11.3 11.9c-1 1-1.2 3.3 0
4.4 1.2 1.1 3.3.8 4.4-.3l16-17c.5-.5.8-1.1.8-1.9z"></path>
                    </svg>
                </span>
                <span class="button-elem">
                    <svg viewBox="0 0 46 40">
                        <path d="M46 20.038c0-.7-.3-1.5-.8-2.1l-16-17c-1.1-1-3.2-1.4-4.4-.3-
1.2 1.1-1.2 3.3 0 4.4l11.3 11.9H3c-1.7 0-3 1.3-3 3s1.3 3 3 3h33.1l-11.3 11.9c-1 1-1.2 3.3 0
4.4 1.2 1.1 3.3.8 4.4-.3l16-17c.5-.5.8-1.1.8-1.9z"></path>
                    </svg>
                </span>
            </div>
        </button>
    </div>
</form>
<style>
    .button-section{
        display: flex;
        align-items: center
    }
    .button {
        display: block;
        position: relative;
        width: 56px;
        height: 56px;
        margin: 0;
        overflow: hidden;
        outline: none;
        background-color: transparent;
        cursor: pointer;
        border: 0;
    }

```

```

.button:before,
.button:after {
  content: "";
  position: absolute;
  border-radius: 50%;
  inset: 7px;
}

.button:before {
  border: 4px solid #f0eeef;
  transition: opacity 0.4s cubic-bezier(0.77, 0, 0.175, 1) 80ms, transform 0.5s
cubic-bezier(0.455, 0.03, 0.515, 0.955) 80ms;
}

.button:after {
  border: 4px solid #96daf0;
  transform: scale(1.3);
  transition: opacity 0.4s cubic-bezier(0.165, 0.84, 0.44, 1), transform 0.5s
cubic-bezier(0.25, 0.46, 0.45, 0.94);
  opacity: 0;
}

.button:hover:before,
.button:focus:before {
  opacity: 0;
  transform: scale(0.7);
  transition: opacity 0.4s cubic-bezier(0.165, 0.84, 0.44, 1), transform 0.5s
cubic-bezier(0.25, 0.46, 0.45, 0.94);
}

.button:hover:after,
.button:focus:after {
  opacity: 1;
  transform: scale(1);
  transition: opacity 0.4s cubic-bezier(0.77, 0, 0.175, 1) 80ms, transform 0.5s
cubic-bezier(0.455, 0.03, 0.515, 0.955) 80ms;
}

.button-box {
  display: flex;
  position: absolute;
  top: 0;
  left: 0;
}

.button-elem {
  display: block;
  width: 20px;
  height: 20px;
  margin: 17px 18px 0 18px;
  transform: rotate(180deg);
  fill: #f0eeef;
}

```

```
.button:hover .button-box,
.button:focus .button-box {
    transition: 0.4s;
    transform: translateX(-56px);
}

button {
    border: 2px solid #24b4fb;
    background-color: #24b4fb;
    border-radius: 0.9em;
    padding: 0.8em 1.2em 0.8em 1em;
    transition: all ease-in-out 0.2s;
    font-size: 16px;
}

button span {
    display: flex;
    justify-content: center;
    align-items: center;
    color: #fff;
    font-weight: 600;
}

button:hover {
    background-color: #0071e2;
}
</style>
```

Model Declaration:

- @model JobSeekingVM: Specifies that the model for the view is of type JobSeekingVM.

HTML Content:

- <h1>Apply for Job</h1>: Displays a heading indicating the purpose of the form.
- <form method="post" enctype="multipart/form-data">: Starts a form with the post method and specifies multipart/form-data encoding type, which is commonly used for file uploads.
- Inside the form:
 - o Hidden input fields for applyCV.JobId: This input field stores the ID of the job to which the application is being submitted. It's hidden from the user interface.
 - o Description field: Allows the user to input a description for their job application.
 - o CV file input: Allows the user to upload their CV file, restricted to PDF files only.

- Create Button: A submit button to submit the form data.
- Back Button: A button styled as a circle with an arrow icon, directing the user back to the index page.

CSS Styling:

- Defines styles for buttons and button animations.

JobSeeking.Areas.JobSeeker.Views. ApplyCV. Index

```
@model List<Job>
<h1>Jobs</h1>
<div class="row">
  <a asp-area="JobSeeker" asp-controller="ApplyCV" asp-action="ListCV" class="btn btn-
primary">List of CV</a>
  @foreach (var job in Model)
  {
    <div class="col-lg-4 col-md-6 mb-4">
      <div class="card">
        <div class="card-header">
          <h5>@job.Name</h5>
        </div>
        <div class="card-body">
          <p><strong>Description:</strong> @job.Description</p>
          <p><strong>Salary:</strong> @job.Salary</p>
          <p><strong>Company Name:</strong> @job.CompanyName</p>
          
          <strong>Categories:</strong>
          @foreach (var categoryName in job.Category)
          {
            <span class="badge bg-primary">@categoryName</span>
          }
        </div>
        <div class="card-footer">
          <a asp-area="JobSeeker" asp-controller="ApplyCV" asp-
action="ViewDetailOfJob" asp-route-Id="@job.Id" class="btn btn-primary">ApplyCV</a>
        </div>
      </div>
    </div>
  }
</div>
```

Model Declaration:

- @model List<Job>: Specifies that the model for the view is a list of Job objects.

HTML Content:

- `<h1>Jobs</h1>`: Displays a heading indicating the purpose of the content.
- `<div class="row">`: Starts a row to contain the job cards and other elements.
- `<a asp-area="JobSeeker" asp-controller="ApplyCV" asp-action="ListCV" class="btn btn-primary">List of CV`: Displays a button that directs users to a list of CVs. It is styled using Bootstrap classes.
- Inside the foreach loop:
 - Iterates over each Job object in the model.
 - `<div class="col-lg-4 col-md-6 mb-4">`: Specifies the column layout for each job card using Bootstrap grid classes.
 - `<div class="card">`: Represents a card containing details of a job.
 - `<div class="card-header">`: Displays the header section of the card containing the job name.
 - `<div class="card-body">`: Displays the body section of the card containing job details such as description, salary, company name, logo, and categories.
 - Inside the nested foreach loop:
 - Iterates over each category associated with the current job and displays it as a badge.
 - `<div class="card-footer">`: Displays the footer section of the card.
 - `<a asp-area="JobSeeker" asp-controller="ApplyCV" asp-action="ViewDetailOfJob" asp-route-Id="@job.Id" class="btn btn-primary">ApplyCV`: Represents a button that allows users to apply for the current job. It directs users to the action ViewDetailOfJob in the ApplyCV controller, passing the job ID as a route parameter.

JobSeeking.Areas.JobSeeker.Views. ApplyCV. ListCV

```
@using JobSeeking.Models.ViewModels
@model List<ApplyCV>
<h1>List CV you were applied</h1>
<button type="button" class="btn btn-warning" data-bs-toggle="modal" data-bs-
target="#exampleModal">
    <i class="bi bi-bell"></i>
</button>
```

```
<div class="row">
  @foreach (var obj in Model)
  {
    <div class="col-lg-4 col-md-6 mb-4">
      <div class="card">
        <div class="card-header">
          <h5>Name: @obj.Job.Name</h5>
        </div>
        <div class="card-body">
          <h5>
            @if (obj.Status==true)
            {
              <h5>Status: Accepted</h5>
            }
            @if (obj.Status == false)
            {
              <h5>Status: Failed</h5>
            }
            Time Create: @obj.TimeCreate
          </h5>
        </div>
        <div class="card-footer">
          <a asp-action="Delete" asp-route-Id="@obj.Id" class="btn btn-
danger"> Delete</a>
          <a asp-action="Index" class="btn btn-primary"> Back to list</a>
        </div>
      </div>
    </div>
    <div class="modal fade" id="exampleModal" tabindex="-1" aria-
labelledby="exampleModalLabel" aria-hidden="true">
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <h1 class="modal-title fs-5" id="exampleModalLabel">Modal title</h1>
            <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
          </div>
          <div class="modal-body">
            @if (obj.Status==true)
            {
              <p><strong>You was accepted CV in Job @obj.Job.Name</strong></p>
            }
            @if (obj.Status==false)
            {
              <p><strong>You was failed CV in Job @obj.Job.Name</strong></p>
            }
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Close</button>
            <button type="button" class="btn btn-primary">Save changes</button>
          </div>
        </div>
      </div>
    </div>
  }
</div>
```

```

        </div>
    </div>
}
</div>

```

Model Declaration:

- @model List<ApplyCV>: Specifies that the model for the view is a list of ApplyCV objects.

HTML Content:

- <h1>List CV you were applied</h1>: Displays a heading indicating the purpose of the content.
- <button type="button" class="btn btn-warning" data-bs-toggle="modal" data-bs-target="#exampleModal">: Represents a button that triggers a modal when clicked. It is styled using Bootstrap classes and contains an icon.
- Inside the foreach loop:
 - o Iterates over each ApplyCV object in the model.
 - o <div class="col-lg-4 col-md-6 mb-4">: Specifies the column layout for each card using Bootstrap grid classes.
 - o <div class="card">: Represents a card containing details of a CV application.
 - o <div class="card-header">: Displays the header section of the card containing the job name associated with the CV.
 - o <div class="card-body">: Displays the body section of the card containing details such as the status of the application and the time of creation.
 - o Uses conditional statements to display different status messages based on the value of the Status property.
 - o <div class="card-footer">: Displays the footer section of the card containing action buttons.
 - o <a asp-action="Delete" asp-route-Id="@obj.Id" class="btn btn-danger"> Delete: Represents a button to delete the CV application. It sends a request to the "Delete" action in the controller, passing the ID of the application as a route parameter.

- `<a asp-action="Index" class="btn btn-primary"> Back to list`: Represents a button to navigate back to the list of CV applications.

Modal:

- The modal is triggered by the button with the `data-bs-target="#exampleModal"` attribute.
- Inside the modal:
 - Displays a title and a body section with conditional statements to show different messages based on the status of the CV application.
 - Contains a footer with buttons to close the modal and save changes.

JobSeeking.Areas.JobSeeker.Views. ApplyCV. ViewDetailOfJob

```
@using JobSeeking.Models.ViewModels
@model JobSeekingVM

<h1>Edit Category</h1>
<form asp-action="Edit" method="post">
    <input hidden asp-for="Job.Id" />
    <div class="form-group">
        <label asp-for="Job.Name" class="control-label"></label>
        <input disabled asp-for="Job.Name" class="form-control" />
    </div>
    <div class="form-group">
        <label asp-for="Job.Description" class="control-label"></label>
        <textarea disabled asp-for="Job.Description" class="form-control"></textarea>
    </div>
    <div class="form-group">
        <label asp-for="Job.Salary" class="control-label"></label>
        <input disabled asp-for="Job.Salary" class="form-control" />
    </div>
    <div class="form-group">
        <label asp-for="Job.Category" class="col-sm-4 col-form-label">Category</label>
        <select disabled asp-for="Job.Category" class="form-select" asp-
items="@Model.Categories" multiple></select>
    </div>
    <a asp-area="JobSeeker" asp-controller="ApplyCV" asp-action="Create" asp-route-
Id="@Model.Job.Id">ApplyCV</a>
</form>
```

Model Declaration:

- `@model JobSeekingVM`: Specifies that the model for the view is of type `JobSeekingVM`.

HTML Content:

- `<h1>Edit Category</h1>`: Displays a heading indicating the purpose of the form, which is to edit job categories.
- `<form asp-action="Edit" method="post">`: Defines a form that will submit data to the "Edit" action in the associated controller using the HTTP POST method.
- `<input hidden asp-for="Job.Id" />`: Hidden input field for the job ID. This is used to send the job ID along with the form submission.
- Inside the form:
 - `<div class="form-group">`: Represents a form group containing form elements.
 - `<label asp-for="Job.Name" class="control-label"></label>`: Label for the job name input field.
 - `<input disabled asp-for="Job.Name" class="form-control" />`: Disabled input field for the job name. Users cannot edit this field.
 - Similar `<div>` elements exist for job description, salary, and category fields, each containing a disabled input or textarea element.
 - `<select disabled asp-for="Job.Category" class="form-select" asp-items="@Model.Categories" multiple></select>`: Disabled dropdown list for selecting job categories. Users cannot edit this field.
- `<a asp-area="JobSeeker" asp-controller="ApplyCV" asp-action="Create" asp-route-Id="@Model.Job.Id">ApplyCV`: Anchor tag that links to the "Create" action in the "ApplyCV" controller with the job ID as a route parameter. This allows users to apply for the job.

JobSeeking.Areas.JobSeeker.Views. Home. AboutUs

```
<link rel="stylesheet" href="~/css/AboutUs.css" asp-append-version="true" />

<h1>Introduction</h1>

<div class="Introduction">

    <div class="skill">
```

```
<div class="skill-box">

  <span class="title">Jobs</span>

  <div class="skill-bar">

    <span class="skill-per html">

      <span class="tooltip">80%</span>

    </span>

  </div>

</div>

<div class="skill-box">

  <span class="title">Categories</span>

  <div class="skill-bar">

    <span class="skill-per css">

      <span class="tooltip">60%</span>

    </span>

  </div>

</div>

<div class="skill-box">

  <span class="title">Security</span>

  <div class="skill-bar">

    <span class="skill-per javascript">

      <span class="tooltip">70%</span>

    </span>

  </div>

</div>
```

```

    </span>

  </div>

</div>

<div class="skill-box">

  <span class="title">Assistance</span>

  <div class="skill-bar">

    <span class="skill-per nodejs">

      <span class="tooltip">80%</span>

    </span>

  </div>

</div>

<p>

  Welcome to JobSeeking, a revolutionary online platform at the forefront of redefining
  job posting and hiring processes. In today's fast-paced society,
  addressing employment concerns is more crucial than ever, and JobSeeking
  stands ready to meet this challenge head-on. With its innovative approach,
  JobSeeking simplifies recruitment experiences for both employers and job seekers
  through an intuitive interface. From effortless job posting to streamlined application
  management and candidate selection, our cutting-edge software offers a seamless solution
  tailored to adapt to the dynamic demands of the modern job market. Stay tuned for an in-depth
  exploration of JobSeeking's extensive features and functionalities, as we embark on a journey
  into a new era of job matching and recruitment efficiency. Join us and embrace the future of
  employment with

```

JobSeeking.

</p>

</div>

</div>

<style>

body {

height: 200px;

background-color: #E1D5E7; /* Light pastel purple */

background-image: linear-gradient(to right, #E1D5E7, #F8BBD0, #BBDEFB); /* Pastel purple to pastel pink to pastel blue */

}

</style>

<div class="card-container">

<div class="card" >

<center>

<div class="profileimage">

</div>

<div class="Name">

<p>Quốc Cường</p>

</div>

<div class="socialbar">

<svg viewBox="0 0 16 16" class="bi bi-github" fill="currentColor" height="16" width="16" xmlns="http://www.w3.org/2000/svg">

<path d="M8 0C3.58 0 0 3.58 0 8c0 3.54 2.29 6.53 5.47 7.59.4.07.55-.17.55-.38 0-.19-.01-.82-.01-1.49-2.01-.37-2.53-.49-2.69-.94-.09-.23-.48-.94-.82-1.13-.28-.15-.68-.52-.01-.53.63-.01 1.08.58 1.23.82.72 1.21 1.87.87 2.33.66.07-.52-.28-.87.51-1.07-1.78-.2-3.64-.89-3.64-3.95 0-.87.31-1.59.82-2.15-.08-.2-.36-1.02-.08-2.12 0 0 .67-.21 2.28-.64.18 1.32-.27 2-.27.68 0 1.36.09 2 .27 1.53-1.04 2.2-.82 2.2-.82.44 1.1.16 1.92.08 2.12.51.56.82 1.27.82 2.15 0 3.07-1.87 3.75-3.65 3.95.29.25.54.73.54 1.48 0 1.07-.01 1.93-.01 2.2 0 .21.15.46.55.38A8.012 8.012 0 0 0 16 8c0-4.42-3.58-8-8-8z"></path>

</svg>

<svg viewBox="0 0 16 16" class="bi bi-instagram" fill="currentColor" height="16" width="16" xmlns="http://www.w3.org/2000/svg">

<path d="M8 0C5.829 0 5.556.01 4.703.048 3.85.088 3.269.222 2.76.42a3.917 3.917 0 0 0-1.417.923A3.927 3.927 0 0 0 .42 2.76C.222 3.268.087 3.85.048 4.7.01 5.555 0 5.827 0 8.001c0 2.172.01 2.444.048 3.297.048.852.174 1.433.372 1.942.205.526.478.972.923 1.417.444.445.89.719 1.416.923.51.198 1.09.333 1.942.372C5.555 15.99 5.827 16 8 16s2.444-.01 3.298-.048c.851-.04 1.434-.174 1.943-.372a3.916 3.916 0 0 0 1.416-.923c.445-.445.718-.891.923-1.417.197-.509.332-1.09.372-1.942C15.99 10.445 16 10.173 16 8s-.01-2.445-.048-3.299c-.04-.851-.175-1.433-.372-1.941a3.926 3.926 0 0 0-.923-1.417A3.911 3.911 0 0 0 13.24.42c-.51-.198-1.092-.333-1.943-.372C10.443.01 10.172 0 7.998 0h.003zm-.717 1.442h.718c.2.136 0 2.389.007 3.232.046.78.035 1.204.166 1.486.275.373.145.64.319.92.599.28.28.453.546.598.92.11.281.24.705.275 1.485.039.843.047 1.096.047 3.231s-.008 2.389-.047 3.232c-.035.78-.166 1.203-.275 1.485a2.47 2.47 0 0 1-.599.919c-.28.28-.546.453-.92.598-.28.11-.704.24-1.485.276-.843.038-1.096.047-3.232.047s-2.39-.009-3.233-.047c-.78-.036-1.203-.166-1.485-.276a2.478 2.478 0 0 1-.92-.598.248.248 0 0 1-.6-.92c-.109-.281-.24-.705-.275-1.485-.038-.843-.046-1.096-.046-3.233 0-2.136.008-2.388.046-3.231.036-.78.166-1.204.276-

```
1.486.145-.373.319-.64.599-.92.28-.28.546-.453.92-.598.282-.11.705-.24 1.485-.276.738-.034 1.024-
.044 2.515-.045v.002zm4.988 1.328a.96.96 0 1 0 0 1.92.96.96 0 0 0 0-1.92zm-4.27 1.122a4.109 4.109 0
1 0 0 8.217 4.109 4.109 0 0 0 0-8.217zm0 1.441a2.667 2.667 0 1 1 0 5.334 2.667 2.667 0 0 1 0-
5.334z"></path>

</svg>

</a>

<a id="facebook"
href="https://www.facebook.com/HuynhVanQuocCuong?mibextid=LQQJ4d" target="_blank">

<svg viewBox="0 0 16 16" class="bi bi-facebook" fill="currentColor" height="16"
width="16" xmlns="http://www.w3.org/2000/svg">

<path d="M16 8.049c0-4.446-3.582-8.05-8.05-8.05C3.58 0-.002 3.603-.002 8.05c0 4.017
2.926 7.347 6.75 7.951v-5.625h-2.03V8.05H6.75V6.275c0-2.017 1.195-3.131 3.022-3.131.876 0
1.791.157 1.791.157v1.98h-1.009c-.993 0-1.303.621-1.303 1.258v1.51h2.218l-.354
2.326H9.25V16c3.824-.604 6.75-3.934 6.75-7.951z"></path>

</svg>

</a>

</div>

</center>

</div>

<div class="card">

<center>

<div class="profileimage">


```

</div>

<div class="Name">

<p>Quốc Trung</p>

</div>

<div class="socialbar">

<svg viewBox="0 0 16 16" class="bi bi-github" fill="currentColor" height="16" width="16" xmlns="http://www.w3.org/2000/svg">

<path d="M8 0C3.58 0 0 3.58 0 8c0 3.54 2.29 6.53 5.47 7.59.4.07.55-.17.55-.38 0-.19-.01-.82-.01-1.49-2.01-.37-2.53-.49-2.69-.94-.09-.23-.48-.94-.82-1.13-.28-.15-.68-.52-.01-.53.63-.01 1.08.58 1.23.82.72 1.21 1.87.87 2.33.66.07-.52.28-.87.51-1.07-1.78-.2-3.64-.89-3.64-3.95 0-.87.31-1.59.82-2.15-.08-.2-.36-1.02-.08-2.12 0 0 .67-.21 2.28-.64.18 1.32-.27 2-.27.68 0 1.36.09 2 .27 1.53-1.04 2.2-.82.22-.82.44 1.1.16 1.92.08 2.12.51.56.82 1.27.82 2.15 0 3.07-1.87 3.75-3.65 3.95.29.25.54.73.54 1.48 0 1.07-.01 1.93-.01 2.2 0 .21.15.46.55.38A8.012 8.012 0 0 0 16 8c0-4.42-3.58-8-8-8z"></path>

</svg>

<svg viewBox="0 0 16 16" class="bi bi-instagram" fill="currentColor" height="16" width="16" xmlns="http://www.w3.org/2000/svg">

<path d="M8 0C5.829 0 5.556.01 4.703.048 3.85.088 3.269.222 2.76.42a3.917 3.917 0 0 0-1.417.923A3.927 3.927 0 0 0 .42 2.76C.222 3.268.087 3.85.048 4.7.01 5.555 0 5.827 0 8.001c0 2.172.01 2.444.048 3.297.048.852.174 1.433.372 1.942.205.526.478.972.923 1.417.444.445.89.719 1.416.923.51.198 1.09.333 1.942.372C5.555 15.99 5.827 16 8 16s2.444-.01 3.298-.048c.851-.04 1.434-.174 1.943-.372a3.916 3.916 0 0 0 1.416-.923c.445-.445.718-.891.923-1.417.197-.509.332-1.09.372-1.942C15.99 10.445 16 10.173 16 8s-.01-2.445-.048-3.299c-.04-.851-.175-1.433-.372-1.941a3.926 3.926 0 0 0-.923-1.417A3.911 3.911 0 0 0 13.24.42c-.51-.198-1.092-.333-1.943-.372C10.443.01 10.172 0 7.998 0h.003zm-.717 1.442h.718c2.136 0 2.389.007 3.232.046.78.035 1.204.166


```
1.486.275.373.145.64.319.92.599.28.28.453.546.598.92.11.281.24.705.275 1.485.039.843.047
1.096.047 3.231s-.008 2.389-.047 3.232c-.035.78-.166 1.203-.275 1.485a2.47 2.47 0 0 1-.599.919c-
.28.28-.546.453-.92.598-.28.11-.704.24-1.485.276-.843.038-1.096.047-3.232.047s-2.39-.009-3.233-
.047c-.78-.036-1.203-.166-1.485-.276a2.478 2.478 0 0 1-.92-.598 2.48 2.48 0 0 1-.6-.92c-.109-.281-.24-
.705-.275-1.485-.038-.843-.046-1.096-.046-3.233 0-2.136.008-2.388.046-3.231.036-.78.166-1.204.276-
1.486.145-.373.319-.64.599-.92.28-.28.546-.453.92-.598.282-.11.705-.24 1.485-.276.738-.034 1.024-
.044 2.515-.045v.002zm4.988 1.328a.96.96 0 1 0 0 1.92.96.96 0 0 0 0-1.92zm-4.27 1.122a4.109 4.109 0
1 0 0 8.217 4.109 4.109 0 0 0 0-8.217zm0 1.441a2.667 2.667 0 1 1 0 5.334 2.667 2.667 0 0 1 0-
5.334z"></path>

</svg>

</a>

<a id="facebook" href="https://m.facebook.com/l.Am.Quoc.Trungg?mibextid=LQQJ4d"
target="_blank">

<svg viewBox="0 0 16 16" class="bi bi-facebook" fill="currentColor" height="16"
width="16" xmlns="http://www.w3.org/2000/svg">

<path d="M16 8.049c0-4.446-3.582-8.05-8-8.05C3.58 0-.002 3.603-.002 8.05c0 4.017
2.926 7.347 6.75 7.951v-5.625h-2.03V8.05H6.75V6.275c0-2.017 1.195-3.131 3.022-3.131.876 0
1.791.157 1.791.157v1.98h-1.009c-.993 0-1.303.621-1.303 1.258v1.51h2.218l-.354
2.326H9.25V16c3.824-.604 6.75-3.934 6.75-7.951z"></path>

</svg>

</a>

</div>

</center>

</div>

<div class="card">

<center>
```

```
<div class="profileimage">

</div>

<div class="Name">

    <p>Hoàng Nam</p>

</div>

<div class="socialbar">

    <a id="github" href="https://github.com/hoangnam150303" target="_blank">

        <svg viewBox="0 0 16 16" class="bi bi-github" fill="currentColor" height="16" width="16"
xmlns="http://www.w3.org/2000/svg">

            <path d="M8 0C3.58 0 0 3.58 0 8c0 3.54 2.29 6.53 5.47 7.59 4.07 5.5-.17 5.5-.38 0-.19-.01-
.82-.01-1.49-2.01-3.7-2.53-.49-2.69-.94-.09-.23-.48-.94-.82-1.13-.28-.15-.68-.52-.01-.53 6.3-.01 1.08 5.8
1.23 8.2 7.2 1.21 1.87 8.7 2.33 6.6 0.7-.52 2.8-.87 5.1-1.07 1.78-.2 3.64-.89 3.64-3.95 0-.87 3.1-1.59 8.2-2.15-
.08-.2-.36-1.02 0.8-2.12 0 0 .67-.21 2.2 8.2 6.4-.18 1.32-.27 2-.27 6.8 0 1.36 0.9 2.27 1.53-1.04 2.2-.82 2.2-
.82 4.4 1.1 1.16 1.92 0.8 2.12 5.1 5.6 8.2 1.27 8.2 2.15 0 3.07-1.87 3.75-3.65 3.95 2.9 2.5 5.4 7.3 5.4 1.48 0 1.07-
.01 1.93-.01 2.2 0 .21 1.5 4.6 5.3 8.0 1.2 8.0 12 0 0 16 8c0-4.42-3.58-8-8-8z"></path>

        </svg>

    </a>

    &nbsp;

    &nbsp;

    &nbsp;

    <a id="instagram" href="https://www.instagram.com/duela.alq/" target="_blank">

        <svg viewBox="0 0 16 16" class="bi bi-instagram" fill="currentColor" height="16"
width="16" xmlns="http://www.w3.org/2000/svg">

            <path d="M8 0C5.829 0 5.556 0.1 4.703 0.48 3.85 0.88 3.269 2.22 2.76 4.2a3.917 3.917 0 0 0-
1.417 9.23A3.927 3.927 0 0 0 .42 2.76C.222 3.268 0.87 3.85 0.48 4.7 5.555 0 5.827 0 8.001c0 2.172 0.1
2.444 0.48 3.297 0.4 8.52 1.74 1.433 3.72 1.942 2.05 5.26 4.78 9.72 9.23 1.417 4.44 5.89 7.19
```

```

1.416.923.51.198 1.09.333 1.942.372C5.555 15.99 5.827 16 8 16s2.444-.01 3.298-.048c.851-.04 1.434-
.174 1.943-.372a3.916 3.916 0 0 0 1.416-.923c.445-.445.718-.891.923-1.417.197-.509.332-1.09.372-
1.942C15.99 10.445 16 10.173 16 8s-.01-2.445-.048-3.299c-.04-.851-.175-1.433-.372-1.941a3.926 3.926
0 0 0-.923-1.417A3.911 3.911 0 0 0 13.24.42c-.51-.198-1.092-.333-1.943-.372C10.443.01 10.172 0 7.998
0h.003zm-.717 1.442h.718c2.136 0 2.389.007 3.232.046.78.035 1.204.166
1.486.275.373.145.64.319.92.599.28.28.453.546.598.92.11.281.24.705.275 1.485.039.843.047
1.096.047 3.231s-.008 2.389-.047 3.232c-.035.78-.166 1.203-.275 1.485a2.47 2.47 0 0 1-.599.919c-
.28.28-.546.453-.92.598-.28.11-.704.24-1.485.276-.843.038-1.096.047-3.232.047s-2.39-.009-3.233-
.047c-.78-.036-1.203-.166-1.485-.276a2.478 2.478 0 0 1-.92-.598 2.48 2.48 0 0 1-.6-.92c-.109-.281-.24-
.705-.275-1.485-.038-.843-.046-1.096-.046-3.233 0-2.136.008-2.388.046-3.231.036-.78.166-1.204.276-
1.486.145-.373.319-.64.599-.92.28-.28.546-.453.92-.598.282-.11.705-.24 1.485-.276.738-.034 1.024-
.044 2.515-.045v.002zm4.988 1.328a.96.96 0 1 0 0 1.92.96.96 0 0 0-1.92zm-4.27 1.122a4.109 4.109 0
1 0 0 8.217 4.109 4.109 0 0 0-8.217zm0 1.441a2.667 2.667 0 1 1 0 5.334 2.667 2.667 0 0 1 0-
5.334z"></path>

</svg>

</a>

&nbsp;

&nbsp;

&nbsp;

<a id="facebook" href="https://m.facebook.com/hoangnam.vu.50767?mibextid=LQQJ4d"
target="_blank">

<svg viewBox="0 0 16 16" class="bi bi-facebook" fill="currentColor" height="16"
width="16" xmlns="http://www.w3.org/2000/svg">

<path d="M16 8.049c0-4.446-3.582-8.05-8-8.05C3.58 0-.002 3.603-.002 8.05c0 4.017
2.926 7.347 6.75 7.951v-5.625h-2.03V8.05H6.75V6.275c0-2.017 1.195-3.131 3.022-3.131.876 0
1.791.157 1.791.157v1.98h-1.009c-.993 0-1.303.621-1.303 1.258v1.51h2.218l-.354
2.326H9.25V16c3.824-.604 6.75-3.934 6.75-7.951z"></path>

</svg>

</a>

</div>

</center>

```

`</div>`

`</div>`

- `<link rel="stylesheet" href="~/css/AboutUs.css" asp-append-version="true" />`: This line includes a CSS file named "AboutUs.css" located in the "css" folder of the project. The `asp-append-version="true"` attribute appends a unique version string to the URL to ensure that browsers always fetch the latest version of the CSS file when it changes.
- `<h1>Introduction</h1>`: This line creates a heading element with the text "Introduction".
- `<div class="Introduction"> ... </div>`: This div contains the introduction section of the webpage. It includes information about the platform and its features.
- Inside the introduction div, there are several div elements with the class "skill-box", each representing a skill or feature with a corresponding progress bar. The progress bar is implemented using CSS classes like "html", "css", etc., and the percentage is displayed as a tooltip.
- Following the introduction section, there's a paragraph (`<p>`) containing a welcome message and an overview of the platform's features.
- After the introduction, there's a `
` element three times, which adds vertical spacing between sections.
- Then, there's a `<div>` with the class "card-container", which seems to contain a set of user profile cards. Each card includes an image, a name, and links to social media profiles (GitHub, Instagram, Facebook).
- Inside each card, there's a profile image (``), a name (`<p>`), and a set of social media links (`<a>`). Each social media link is represented by an SVG icon.
- The SVG icons for social media links are embedded inline with `<svg>` elements, and they use the Bootstrap Icons library (indicated by the class "bi bi-*").

JobSeeking.Areas.JobSeeker.Views. Home.Index

```
@model List<News>
<!DOCTYPE html>
<html lang="en">
<head>
```

```

<link rel="stylesheet" href="~/css/IndexHome.css" asp-append-version="true" />
<style>
    body {
        height: 200px;
        background-color: #E1D5E7; /* Light pastel purple */
        background-image: linear-gradient(to right, #E1D5E7, #F8BBD0, #BBDEFB); /*
Pastel purple to pastel pink to pastel blue */
    }
    a{
        text-decoration:none;
        color:black;
    }
</style>
</head>
<body>
    <div class="container">
        <div class="card">
            
            <div class="card-content">
                <svg>...</svg>
                <h5 class="title"><a asp-action="AboutUs">About Us </a></h5>
            </div>
            <div class="backdrop"> </div>
        </div>
        <div class="card">
            
            <div class="card-content">
                <svg>...</svg>
                <h5 class="title">Contact Us</h5>
            </div>
            <div class="backdrop"> </div>
        </div>
        <div class="card">
            
            <div class="card-content">
                <svg>...</svg>
                <h5 class="title">Contact Us</h5>
            </div>
            <div class="backdrop"> </div>
        </div>
    </div>
</body>
</html>
<br />
<br />
<div class="title">
    <h5><strong>News</strong></h5>
</div>
<br />
<br />
@foreach (var newsItem in Model)
{

```

```
<div class="card-news">
  <a class="card1" asp-action="ShowNews" asp-route-Id="@newsItem.Id">
    <p>@newsItem.Name</p>
    <p class="small">@newsItem.Description</p>
    <div class="go-corner">
      <div class="go-arrow">
        →
      </div>
    </div>
  </a>
</div>
}
```

Head Section

- Stylesheet Link: This section links the HTML file to an external CSS file named IndexHome.css.
- Inline Styles: Sets some styles for the body element, including height and background colors.

Body Section

- Container Div: Contains a set of cards, each representing a different section or feature of the website.
 - Card 1: Represents the "About Us" section.
 - Background Image: A decorative image for the card.
 - Card Content: Includes a title ("About Us").
 - Card 2: Represents the "Contact Us" section.
 - Background Image: Another decorative image for the card.
 - Card Content: Includes a title ("Contact Us").
 - Card 3: Represents a section (possibly "Privacy Policy").
 - Background Image: Another decorative image for the card.
 - Card Content: Includes a title ("Contact Us").

News Section

- Title Div: Contains a title ("News") for the news section.

- @foreach Loop: Iterates over each news item in the Model (presumably a list of news items).
 - Card-news Div: Represents each news item as a card.
 - Anchor Tag (<a>): Links to a specific action (ShowNews) with the news item's ID as a route parameter.
 - News Name: Displays the name of the news item.
 - News Description: Displays a short description of the news item.
 - Go-corner and Go-arrow: Possibly decorative elements indicating a link or action.

JobSeeking.Areas.JobSeeker.Views. Home. Privacy

```
@{
    ViewData["Title"] = "Privacy Policy";
}
<h1>@ViewData["Title"]</h1>
<p>Use this page to detail your site's privacy policy.</p>
```

Razor Syntax

- @{} Block: This is a Razor code block, enclosed within curly braces {}. Razor code blocks are used to write C# code within a Razor view file.
 - ViewData: ViewData is a dictionary-like object that allows you to pass data between a controller and a view in ASP.NET MVC or ASP.NET Core applications.
 - "Title" Key: Sets the title of the view to "Privacy Policy" using ViewData["Title"].
- <h1> Tag: This is an HTML heading element (<h1>) that displays the value of the "Title" key in the ViewData dictionary. In this case, it will render "Privacy Policy" as the main heading of the page.
- <p> Tag: This is an HTML paragraph element (<p>) that provides a brief description, instructing users to use the page to find details about the site's privacy policy.

JobSeeking.Areas.JobSeeker.Views. Home.ShowNews

@model News

<h1>@Model.Name</h1>

<h5>@Model.DateCreated</h5>

<p>@Model.Description</p>

Razor Syntax

- **@model Directive:** This directive specifies the type of the model used by the view. In this case, the view expects a single instance of the News model.
- **@{ } Block:** This is a Razor code block, enclosed within curly braces { }, used to write C# code.
- **@Model Property Access:**
 - **@Model.Name:** Outputs the value of the Name property of the News model as an HTML heading (<h1>).
 - **@Model.DateCreated:** Outputs the value of the DateCreated property of the News model as an HTML subheading (<h5>).
 - **@Model.Description:** Outputs the value of the Description property of the News model as an HTML paragraph (<p>).

3. Provide Final screenshots of the application:

This is Home page

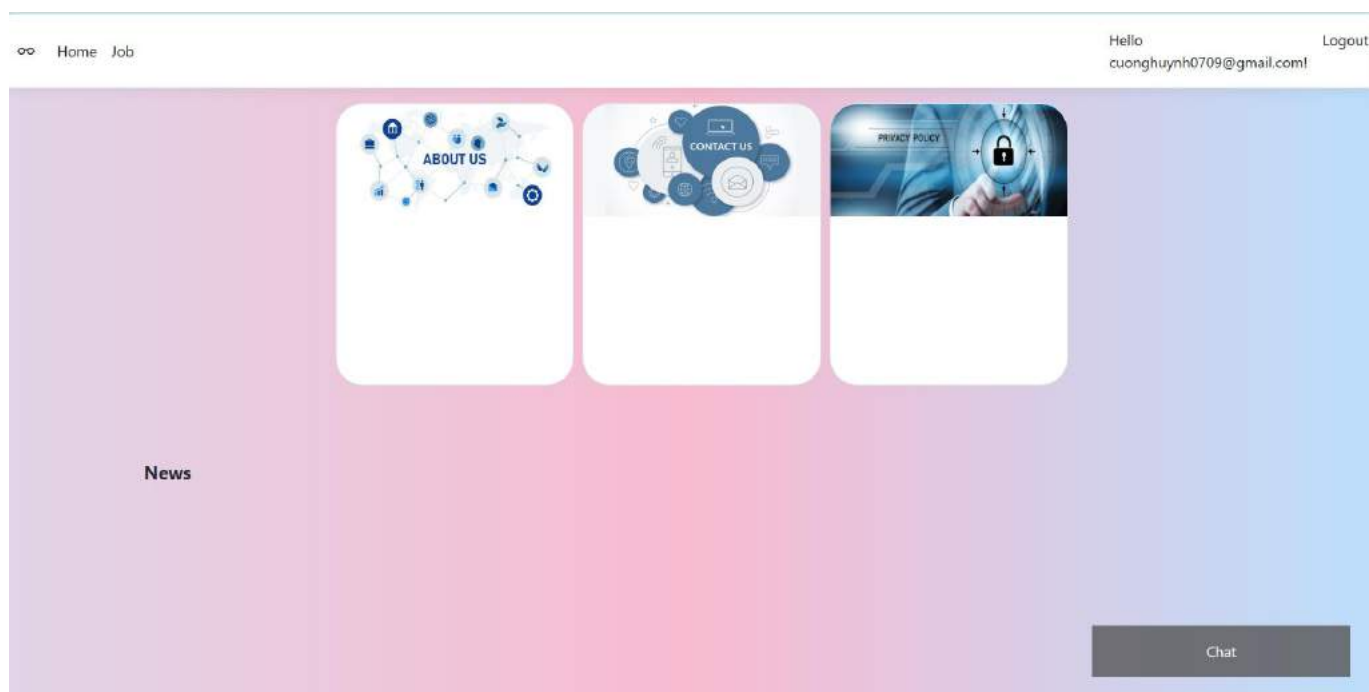


Figure 11 Home page

Login page

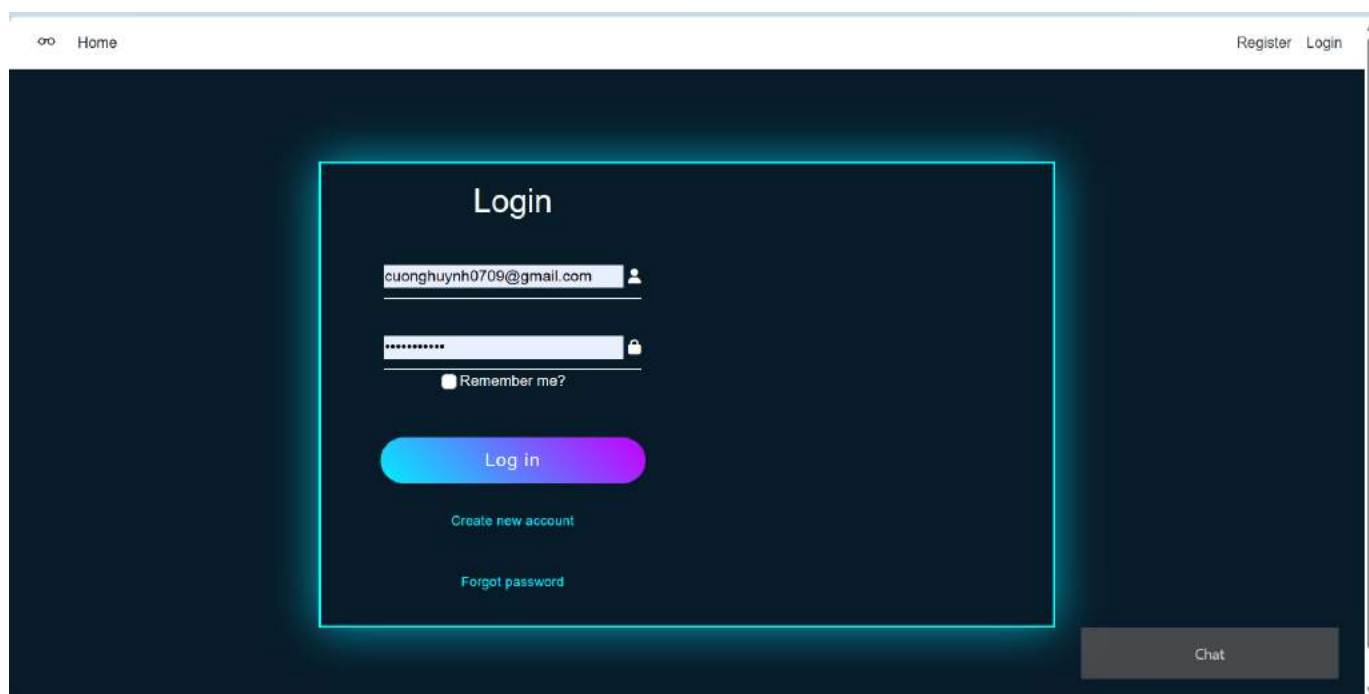


Figure 12 Login page

Register page:

Home Register Login

Register

Create a new account

Email:

Password:

Confirm Password:

Name:

Address:

City:

Avatar: No file chosen

Already registered? [Login](#)

Figure 13 Register for Admin and Employer

Role Admin:

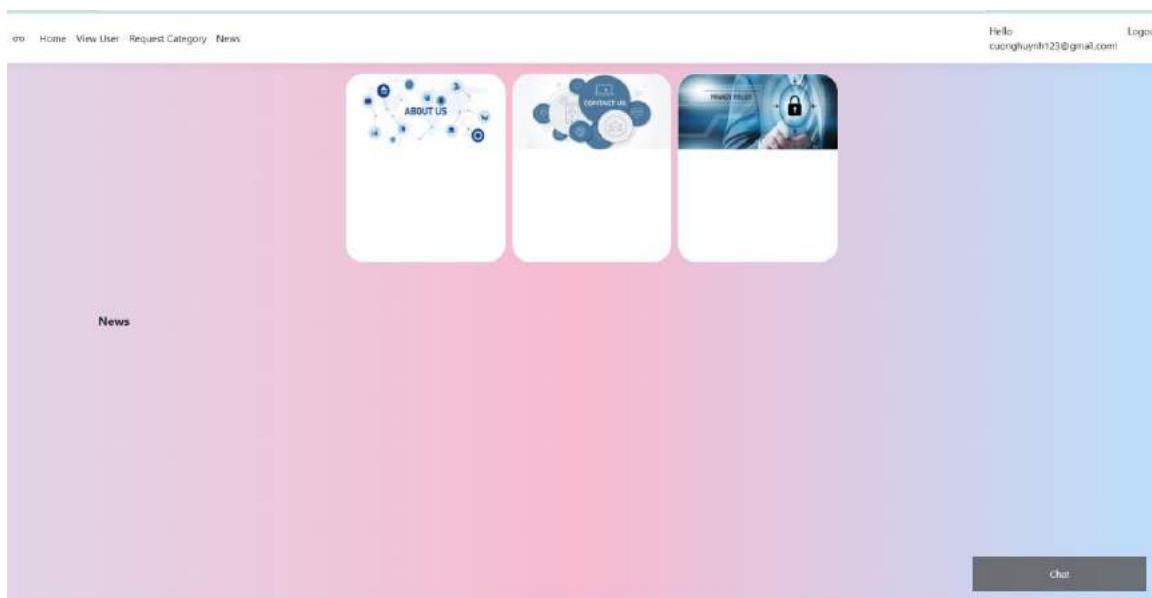
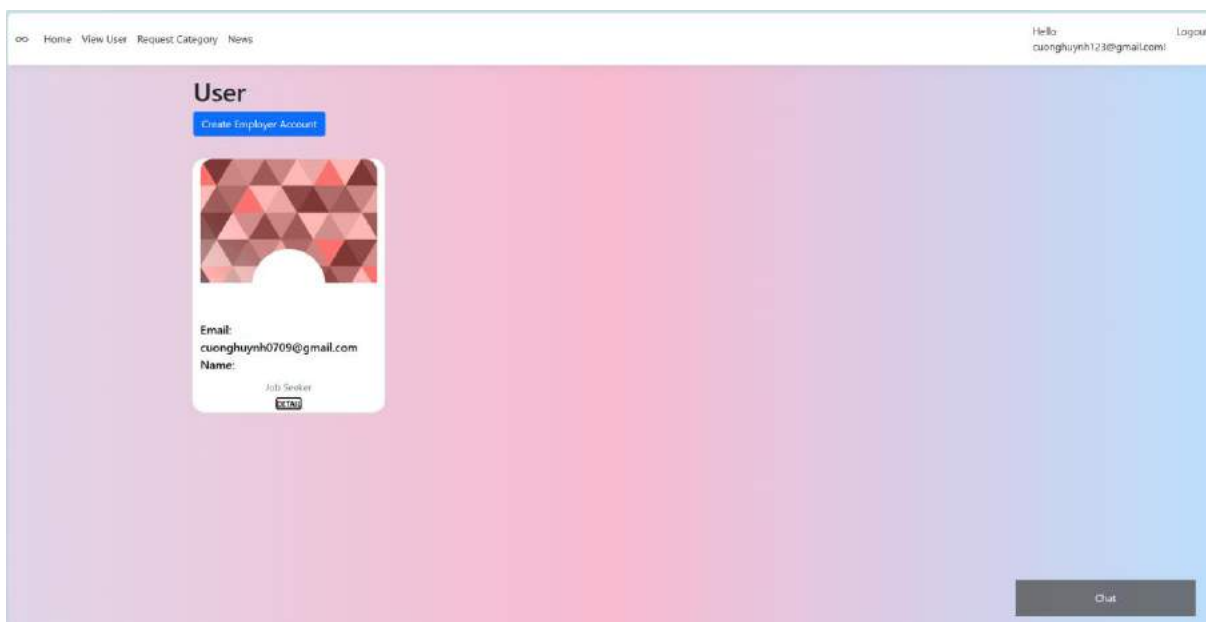


Figure 14 Admin page

View User of Admin:



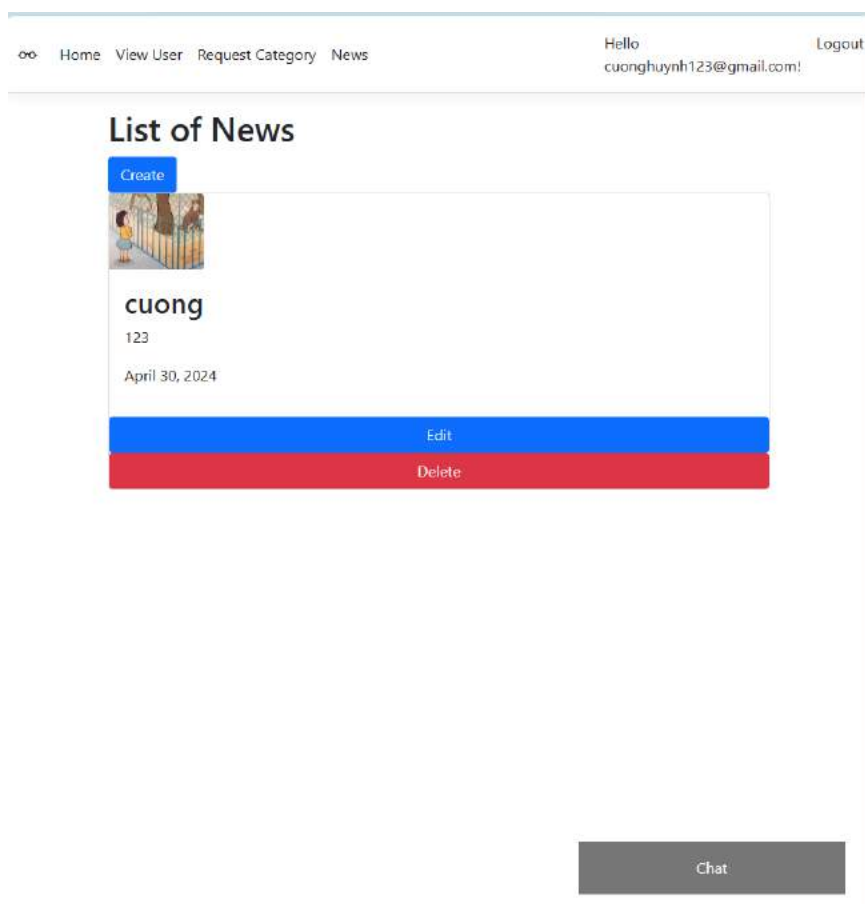
Create a new account page:

Figure 15 Register for User

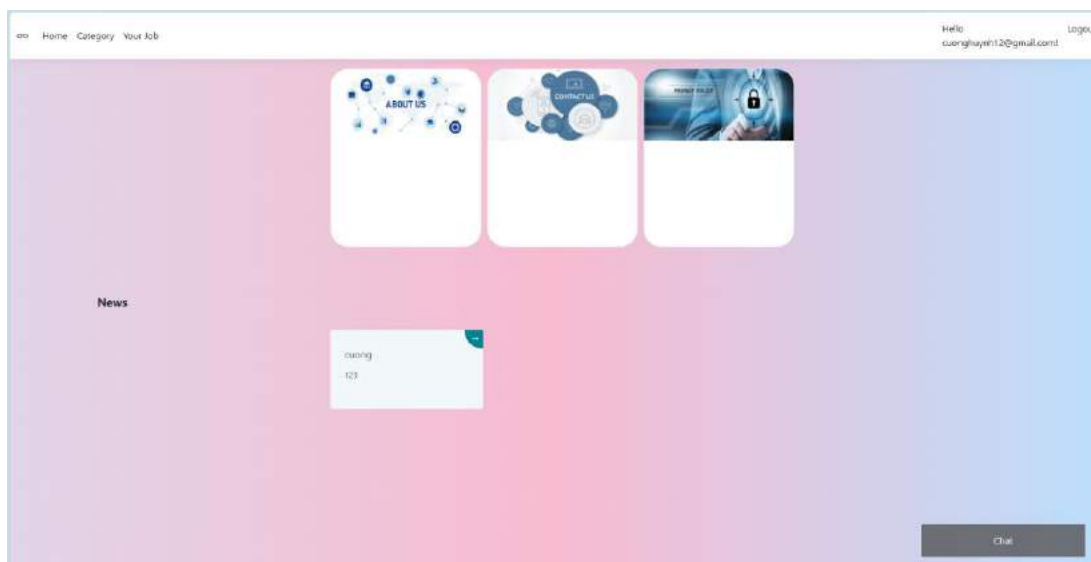
Request Category page:



117 | Page



Role Employer:



Create Category page:

00 Home Category Your Job Hello, cuonghuynh12@gmail.com Logout

Create Category

Name: Data Science

Create

Chat

Category page:

00 Home Category Your Job Hello, cuonghuynh12@gmail.com Logout

All Categories

Create Category Notification

Name: Design Valid: True Time Create: 4/30/2024 2:15:19 PM	Name: IoT Valid: True Time Create: 4/30/2024 2:15:38 PM	Name: Project Manager for IT Valid: True Time Create: 4/30/2024 2:16:10 PM
Name: Data Science Valid: True Time Create: 4/30/2024 2:38:46 PM		

Chat

Figure 17 Category page

Create Job page:

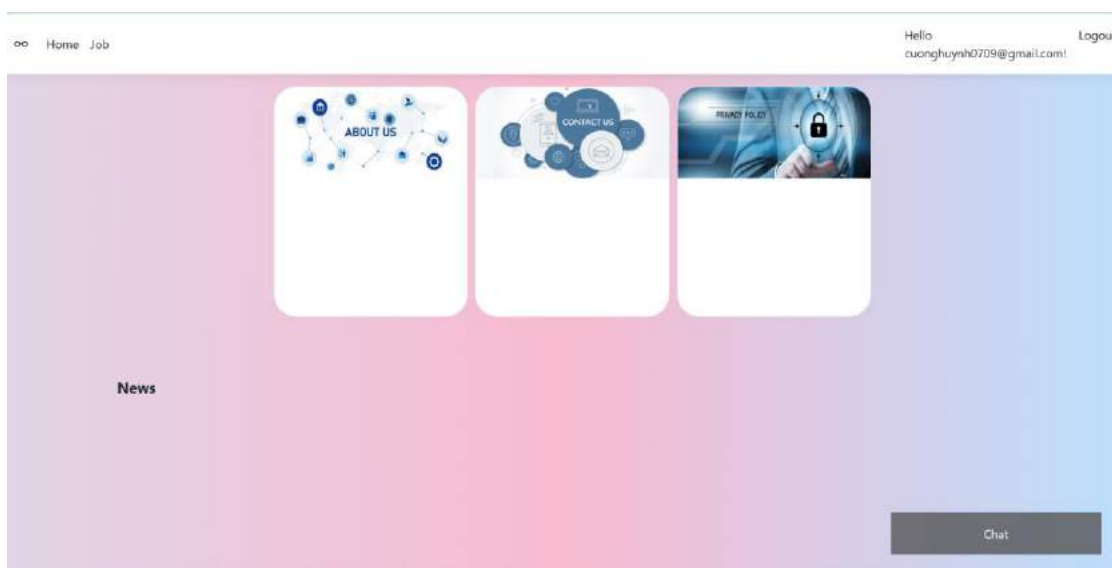
The screenshot shows a web application interface for creating a job. At the top, there is a navigation bar with links: Home, Category, and Your Job. On the right, a user is logged in as 'Hello cuonghuynh12@gmail.com' with a 'Logout' link. The main heading is 'Create Job'. Below it, there are input fields for 'Name' (containing 'tuoi'), 'Description' (containing 'abcd'), 'Salary' (containing '10000'), and 'Category' (a dropdown menu with 'Data Science' selected). A blue 'Create' button is at the bottom left. A 'Chat' button is at the bottom right.

Job page:

The screenshot shows the 'Your Jobs' page. The navigation bar is the same as the previous page. The main heading is 'Your Jobs'. Below it, there are four job cards. Each card has a title, description, salary, company name, email, company logo, and categories. The jobs are: 'tuoi' (Salary: 10000, Category: Data Science), 'nam' (Salary: 15000, Category: Project Manager for IT), 'trung' (Salary: 10000, Category: Design), and 'tuoi' (Salary: 10000, Category: Data Science). Each card has a set of icons at the bottom: a red trash icon, a yellow star icon, a green plus icon, and a blue minus icon. A 'Chat' button is at the bottom right.

Figure 18 Job page

Role JobSeeker



Apply CV page:

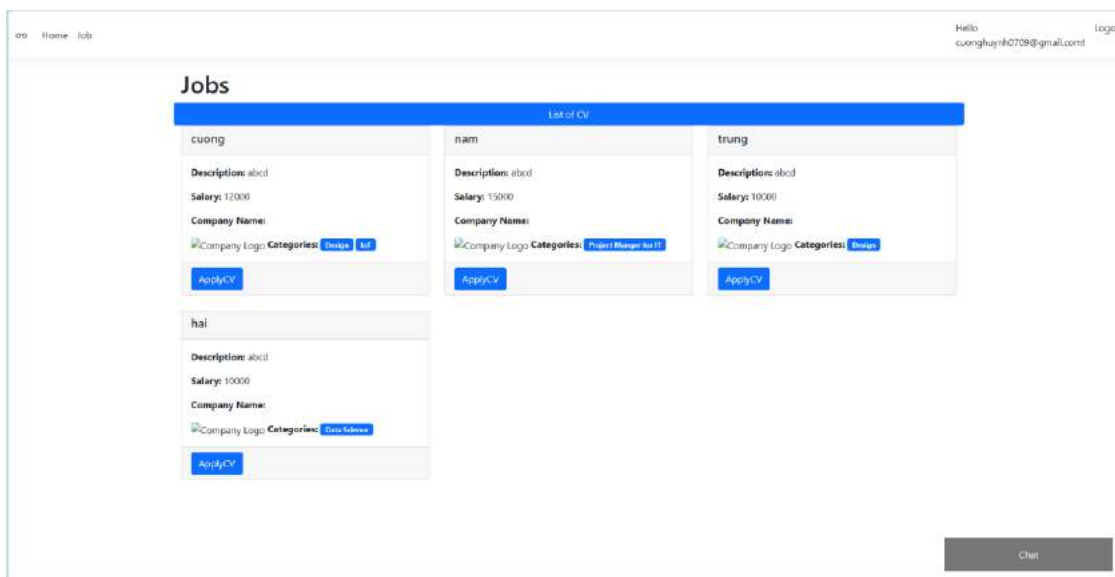


Figure 19 Apply job page

Home Job Help Logout
cuonghuynh0709@gmail.com

Edit Category

Name
cuong

Description
abcd

Salary
12000

Category
Design IoT

[Add CV](#)

Chat

Home Job Hello Logout
cuonghuynh0709@gmail.com

Apply for Job

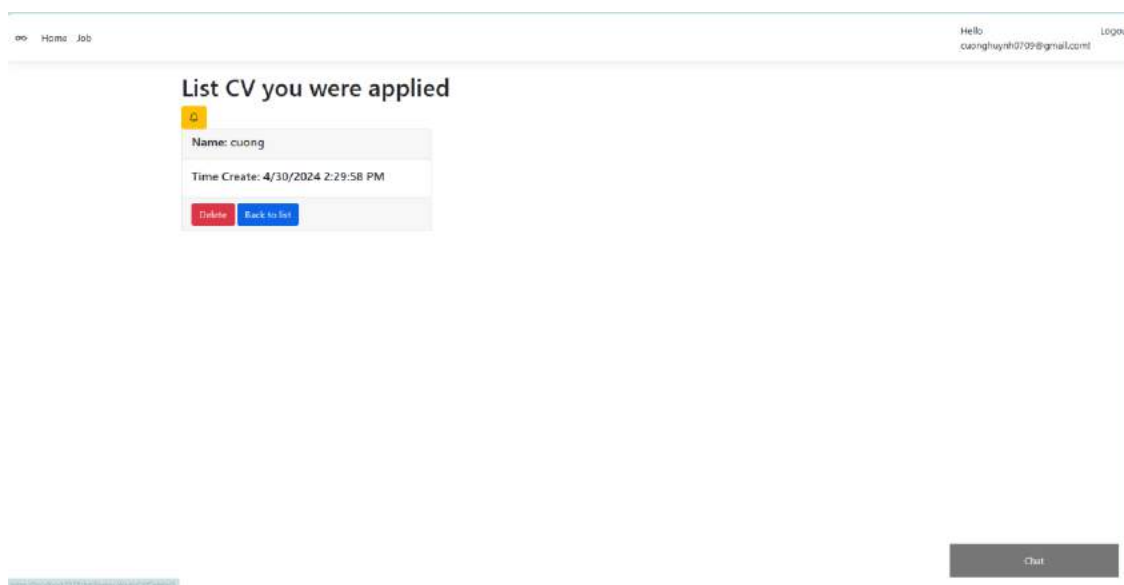
Description

CV (PDF only)
Choose File No file chosen

+ Create

Chat

List CV page:



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

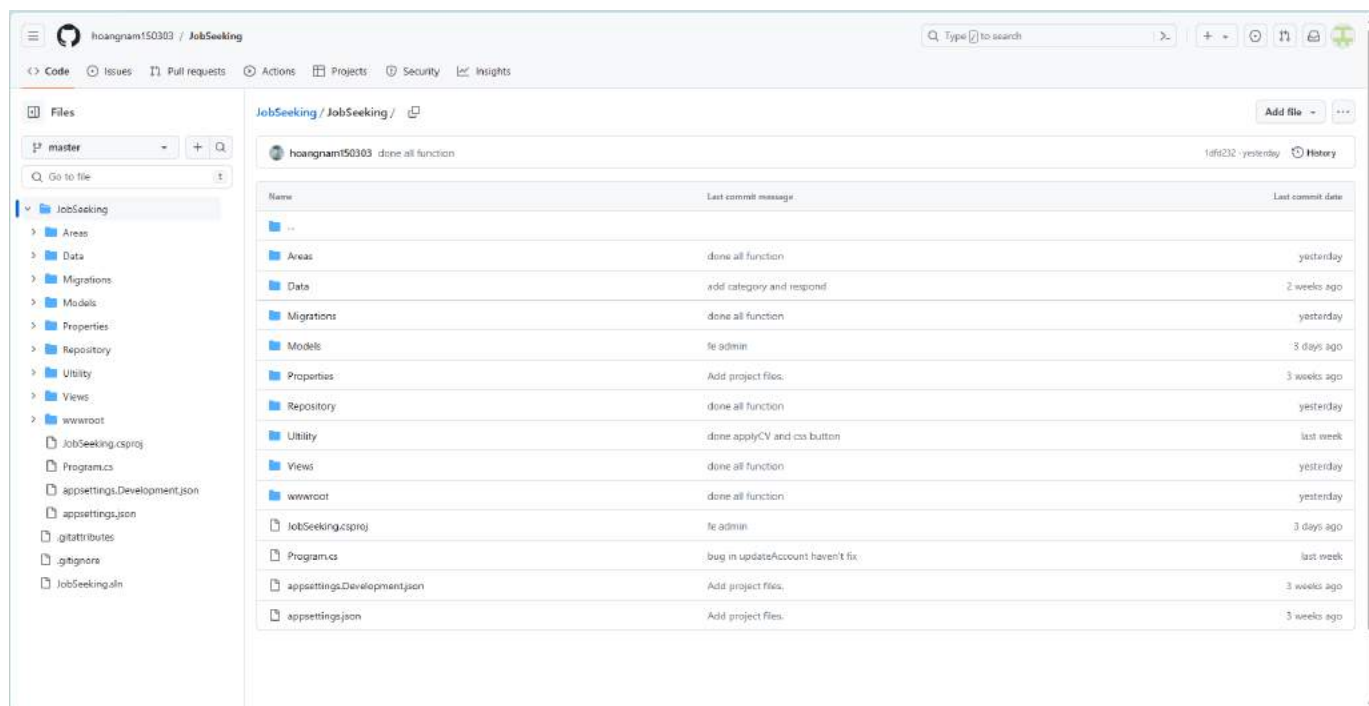


Figure 20 Source code in Github

GitHub, along with Git, is a crucial tool for both individual developers and teams working on software projects. Git's version control capabilities allow for precise tracking of code changes, easy rollback to previous versions, and collaboration without the risk of losing data. GitHub serves as a collaborative platform, enabling seamless teamwork among developers.

A notable feature of Git is its support for concurrent work by multiple team members, each in their own branches. This approach ensures that individual contributions are kept separate until they are ready to be merged into the main codebase. Pull Requests (PRs) play a key role in GitHub by facilitating code review, discussion, and integration of changes.

In GitHub, PRs provide a structured way for team members to review each other's code changes. This systematic process helps maintain code integrity and enables early detection and correction of potential bugs. Additionally, PRs promote knowledge sharing and best practices among team members, contributing to a more cohesive and knowledgeable development team.

Overall, GitHub, in conjunction with Git, goes beyond being a version control system; it serves as a dynamic collaboration platform that empowers developers to collectively improve code quality, streamline workflows, and establish a solid foundation for successful software projects.

This is link Github of my project: [My link github](#)

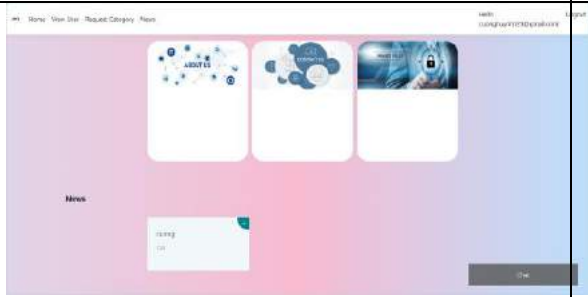


III. APPLICATION EVALUATION


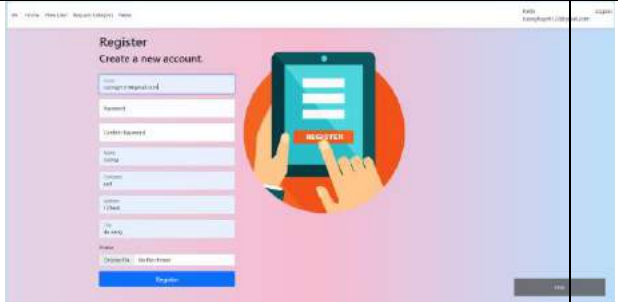
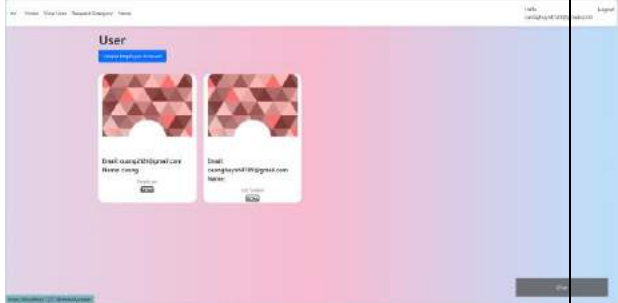
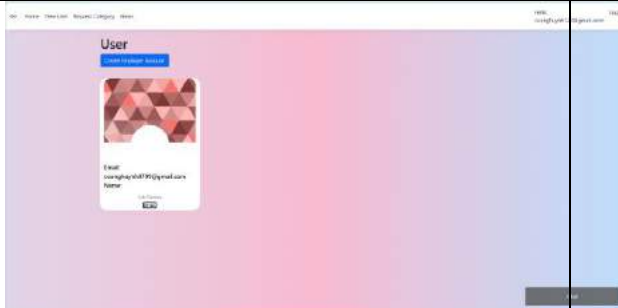
1. Review the performance of the application(P6)





	Test Case	Expect Result	Actual Result
1	Login	Success	Not Yet
2	Register	Success	Not Yet
3	Create Employer Account	Success	Not Yet
4	Delete Employer Account	Success	Not Yet
5	Create news	Success	Not Yet

6	Create Category	Success	Not Yet
7	Create Job	Success	Not Yet
8	Lock Account	Success	Not Yet

Table 1 Test Case

ID	Action	Expected Result	Actual Result	Conclusion
1	Login with Admin	Login Successfully and Move to Home Page	 The screenshot shows the Admin Home Page. It features a top navigation bar with links: Home, View User, Request Category, News, and Logout. The main content area has a sidebar with 'ABOUT US' and 'CONTACT US' buttons, and a central section with a 'News' card and a 'Login' button. The background is a light blue gradient.	PASS
	Login with Employer	Login Successfully and Move to Home Page	 The screenshot shows the Employer Home Page. It features a top navigation bar with links: Home, Category, View Job, and Logout. The main content area has a sidebar with 'ABOUT US' and 'CONTACT US' buttons, and a central section with a 'News' card and a 'Login' button. The background is a light blue gradient.	PASS
	Login with Job Seeker	Login Successfully and Move to Home Page	 The screenshot shows the Job Seeker Home Page. It features a top navigation bar with links: Home, Profile, Add, and Logout. The main content area has a sidebar with 'ABOUT US' and 'CONTACT US' buttons, and a central section with a 'News' card and a 'Login' button. The background is a light blue gradient.	PASS

2	Users register for an account on the Website	Register Successful and Move to Login Page		PASS
3	Create Employer Account	Create Successfully and Move to Home Page		PASS
				PASS
4	Delete Employer Account	Notification 'Are you sure you want to delete user: cuong2121@gmail.com' Click Ok for delete		PASS

5				PASS
				PASS
6	Create Category	Create Successfully and Move to All Category		PASS
				PASS

7	Create Job	Create Successfully and Move to Job		PASS
				PASS
8	Lock Account	Click to  and notifications 'Are you sure you want to Lock & Unlock user: cuong2121@gmail.com'		PASS
				PASS

2. Conclude whether the application adapts all requirements or it needs to be improved later(P6)

Continuous Monitoring and Iterative Improvement:

The critical review suggests that the application is well-prepared to meet the current requirements. However, the dynamic nature of technology and user expectations necessitates continuous monitoring and iterative improvements. Regular assessments and updates should be part of the development cycle to address emerging challenges and incorporate new features.

User Feedback and Evolution of Requirements:

User feedback is crucial for gauging the actual user experience and identifying areas for improvement. As users interact with the platform, their feedback can reveal insights that may lead to enhancements or new requirements. The application should be adaptable to evolving user needs and industry trends.

Scalability for Future Growth:

While the application demonstrates scalability considerations, its ability to handle future growth needs ongoing assessment. As the user base and transaction volume increase, the application should be equipped to scale seamlessly. Regular performance monitoring and scalability testing can help ensure that the system remains robust under varying workloads.

Security Posture and Compliance:

Security is an ongoing concern, and the application should adapt to new security threats and compliance requirements. Regular security audits, updates, and patches are essential to maintaining a secure environment.

In summary, while the JobSeeking application appears to meet the current requirements well, its long-term success will depend on the commitment to continuous improvement, responsiveness to user feedback, scalability for future growth, and vigilance in addressing security considerations. Regular updates and adaptability to changing circumstances will contribute to the sustained success of the application.

3. Develop a functional business application based on a specific Software Design Document with supportive evidence of using the preferred tools, techniques and methodologies(M4).

In asm1 I used tools to help in my project. In this section I will list out how I used those tools in the JobSeeking project.

3.1. C#

In this project I use C# as the main programming language for my back-end. With many benefits such as support libraries.

```
24 references
public int Id { get; set; }
[Required]
18 references
public string Name { get; set; }
[Required]
11 references
public string Description { get; set; }
[Required]
11 references
public double Salary { get; set; }
```

Figure 21: C#

```
if (id == null || id == 0)
{
    return NotFound();
}

var job = _unitOfWork.JobRepository.Get(c => c.Id == id);

if (job == null)
{
    return NotFound();
}

job.amountOfCV += 1;
_unitOfWork.JobRepository.Update(job);
_unitOfWork.JobRepository.Save();
var jobSeekingVM = new JobSeekingVM();
jobSeekingVM.applyCV = new ApplyCV();
jobSeekingVM.applyCV.JobId = job.Id;
```

Figure 22: C#

3.2. MVC .net core

MVC .net core is a powerful C# framework. It assisted me very well in using the MVC pattern in this project. It helps me divide and manage folders and classes with different functions.

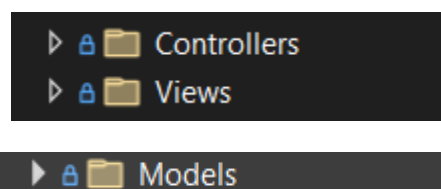


Figure 23: MVC .net core

3.3. Html, Css and Javascript

Html and Css are two extremely useful tools and almost every website today needs them. Javascript is a powerful language, it helps my website become more lively. All three of these tools have helped make my web interface user-friendly and easy to use.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] JobSeeking</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="~/JobSeeking.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 href="https://cdn.jsdelivr.net/npm/select2@4.1.0-rc.0/dist/css/select2.min.css" rel="stylesheet" />
</head>
<body>
  <section class="loader">
    <div class="slider" style="--i:0">
    </div>
    <div class="slider" style="--i:1">
    </div>
    <div class="slider" style="--i:2">
    </div>
    <div class="slider" style="--i:3">
    </div>
    <div class="slider" style="--i:4">
    </div>
  </section>
```

Figure 24: HTML

```
body {
  font-family: Arial, Helvetica, sans-serif;
}

* {
  box-sizing: border-box;
}

a.navbar-brand {
  white-space: normal;
  text-align: center;
  word-break: break-all;
}

a {
  color: #0077cc;
}

.btn-primary {
  color: #fff;
  background-color: #1b6ec2;
  border-color: #1861ac;
}

.nav-pills .nav-link.active, .nav-pills .show > .nav-link {
  color: #fff;
  background-color: #1b6ec2;
  border-color: #1861ac;
}
```

Figure 25: CSS

```
<script>
  $(document).ready(function () {
    $('.form-select').select2();
  });
</script>
@await RenderSectionAsync("Scripts", required: false)
<script>
  function openForm() {
    document.getElementById("myForm").style.display = "block";
  }

  function closeForm() {
    document.getElementById("myForm").style.display = "none";
  }
</script>
<script>
  $(document).ready(function () {
    $('.loader').hide();

    $(document).on('click', 'a', function () {
      $('.loader').show();
    });
    $(window).on('load', function () {
      $('.loader').hide();
    });
  });
</script>
```

Figure 26: Javascript

3.4. Bootstrap

I use Bootstrap to add icons along with using useful supports such as displaying notifications like Modal. Bootstrap has helped me very well in implementing the interface design.

I add Bootstrap library in Layout.cshtml

```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css">
```

Figure 27: Bootstrap library link

I use Bootstrap to add icon in my website.

```
<i class="bi bi-eyeglasses"></i></a>
```

Figure 28: Bootstrap icon

I use Bootstrap to display notification.

```
<!-- Modal -->
<div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModallabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h1 class="modal-title fs-5" id="exampleModallabel">Modal title</h1>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        @foreach (var category in Model.Where(c => c.isValid))
        {
          @if (category.isValid && !category.categoryValid)
          {
            <div>
              @if (!category.categoryValid)
              {
                <div class="notification-status">
                  <form asp-controller="Category" asp-action="ToggleNotification" method="post">
                    Category <strong>@category.Name</strong> has been confirmed.
                    <input type="hidden" name="id" value="@category.Id" />
                    <button type="submit" class="btn-close"></button>
                  </form>
                </div>
              }
            </div>
          }
        }
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button>
      </div>
    </div>
  </div>
</div>

<script>
function confirmSeen(categoryId) {
  document.getElementById('notification-' + categoryId).style.display = 'none';
}
</script>
```

Figure 29: Bootstrap for notification

3.5. Figma

I use figma to create a look at my website before coding. From there, I can receive more customer reviews about the website we are about to create. From these templates, I can collect all the ideas and opinions from customers to be able to bring the most suitable website.

3.6. Project libre

Project libre is a useful tool in this project. It helps me to follow the progress of the project and know what processes are being implemented. If there are any errors or omissions, Project libre helps me keep track of them.

3.7. ERD

I use the ERD design to design the Database. It helps me connect tables together so I can understand how objects interact with each other.

```
public string EmployerId { get; set; }
[ForeignKey("EmployerId")]
[ValidateNever]
1 reference
public ApplicationUser User { get; set; }
```

Figure 30: Create relationship of model from ERD

```
public int JobId { get; set; }
[ForeignKey("JobId")]
[ValidateNever]
3 references
public Job Job { get; set; }
```

Figure 31: Create relationship of model from ERD

3.8. SQL Server

SQL server is a database created by Microsoft. This is an SQL database, helping to create relationships between tables. From there I can easily manage the data of the tables.

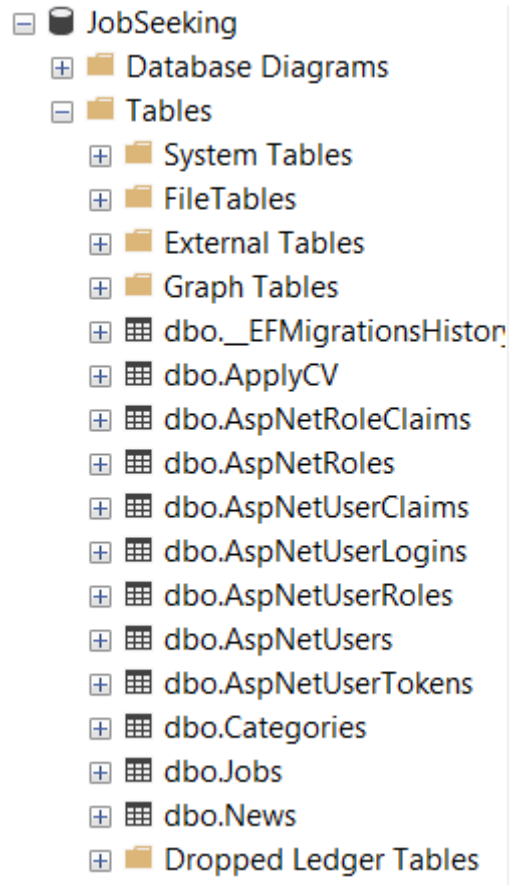


Figure 32: SQL Server

3.9. Waterfall Model.

Requirements Phase:

During this initial stage, you determine what the system should do for my JobSeeking website.

- **Key tasks include:**
 - Specifying the resources required for the project.
 - Assigning tasks to team members and determining at which stage they will work.
 - Creating a timeline for the entire project, outlining how long each stage will take.
 - Providing details on each stage of the process.

- Keep in mind that requirements can range from abstract concepts to detailed mathematical specifications¹.

Design Phase:

After gathering all the requirements, the project moves to the design stage.

Designers create solutions that meet the specified requirements.

- **Tasks during this phase include:**

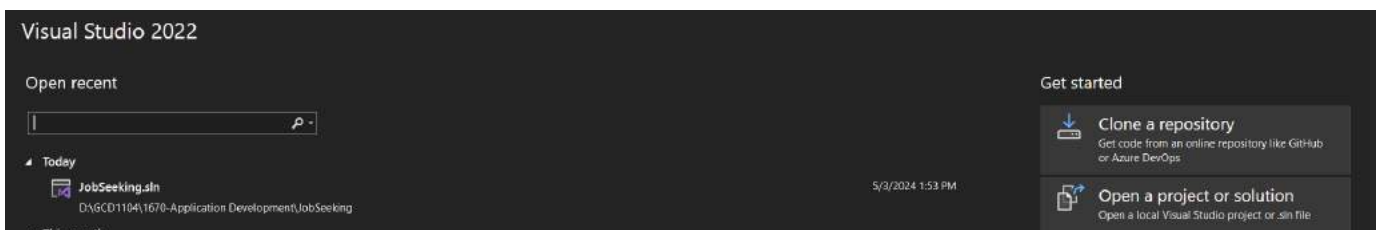
- Developing schedules and project milestones.
- Determining the exact deliverables (which could be software or physical products).
- Creating designs or blueprints for these deliverables.
- For my JobSeeking website, designers would also define the system architecture and use cases.

- **Implementation, Verification, and Maintenance Phases:**

- Once the design is complete, the project moves into implementation (coding), followed by verification (testing) and maintenance.
- Each phase must be completed before progressing to the next.

3.10. Visual Studio Code

Visual studio code is a code compilation tool. It supports the use of many programming languages and of course C# can be used on this tool. Besides, there is also support for using .net MVC. In short, this is a popular tool for programmers.



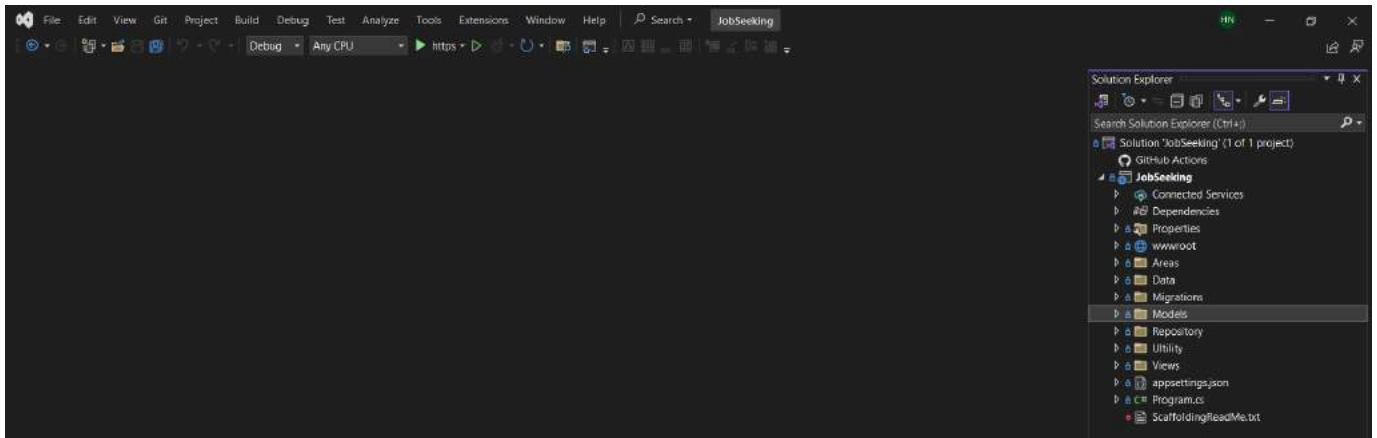


Figure 33: Visual Studio Code

3.11. Github

After each code, there are any changes to the code. I push to Github so I can store changes or deploy new functionality. That can let project participants know about code changes in the project.

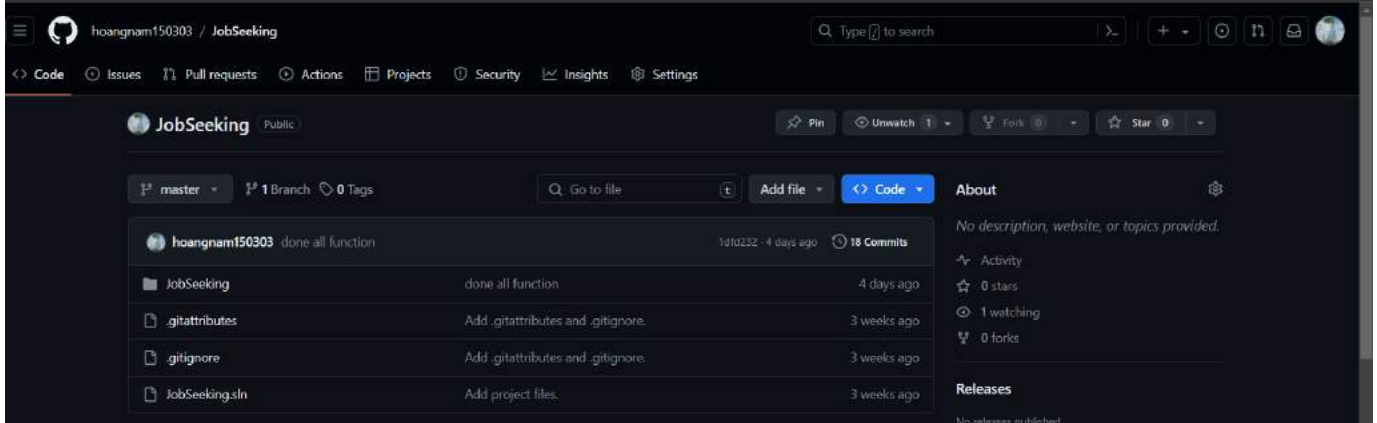


Figure 34: Github

4. Analyze the factors that influence the performance of the application(M5)

- **Design Stage:**

Scalability:

Review: The application architecture has been designed to scale efficiently.

Observation: Both horizontal and vertical scaling options have been considered.

Recommendation: The choice of technologies and frameworks aligns well with scalability goals.

User Interface (UI) Design:

Review: The UI is intuitive and responsive.

Observation: The UI design meets user expectations and industry standards.

Recommendation: Usability testing has been recommended to further refine the UI.

- **Development Stage:**

Code Quality:

Review: Code quality is emphasized, and maintainable code is encouraged.

Observation: Adherence to coding standards and best practices is evident.

Recommendation: Ongoing emphasis on modular and reusable code is recommended.

Database Optimization:

Review: Database schema and queries are optimized.

Observation: Indexing and caching strategies have been implemented.

Recommendation: Regular reviews and optimizations of database queries are encouraged.

Security Measures:

Review: Robust security measures are integrated into the application.

Observation: Regular updates and patching of security vulnerabilities are in place.

Recommendation: The continued conduct of security audits during development is suggested.

- **Testing Stage:**

Performance Testing:

Review: Thorough performance testing has been conducted.

Observation: Scalability bottlenecks have been identified and addressed.

Recommendation: The implementation of simulated user loads for peak traffic scenarios is a positive practice.

Usability Testing:

Review: Usability testing has been conducted to validate the UI.

Observation: User experience issues have been identified and addressed.

Recommendation: The integration of usability testing as a regular practice is beneficial.

Security Testing:

Review: Thorough security testing has been performed.

Observation: Effective implementation of security measures has been ensured.

Recommendation: The ongoing identification and remediation of security vulnerabilities during testing are crucial.

- **Reflective Discussion on Previously Identified Risks:**

User Authentication and Authorization:

Review: Authentication mechanisms are robust, and regular testing is emphasized.

Observation: Multi-factor authentication has been recommended for enhanced security.

Recommendation: Ongoing vigilance and updates in authentication procedures are encouraged.

Scalability Challenges:

Review: The scalability of the application architecture has been evaluated.

Observation: Proactive monitoring mechanisms for identifying scalability issues are in place.

Recommendation: Continued emphasis on scalability, especially as the user base grows, is recommended.

Conclusion:

The critical review indicates that the JobSeeking application has been well-designed, developed, and tested with a focus on performance, security, and scalability. The identified recommendations are aimed at further enhancing the application's robustness and sustainability. Continuous monitoring, updates, and adherence to best practices will contribute to the longterm success of the platform.

5. Evaluate any new insights, ideas or potential improvements to your system and justify the reasons why you have chosen to include (or not to include) them as part of this business application(D2).

- **AI Integration:** Including AI in my website can significantly enhance user experience and provide valuable functionalities. AI can be used for personalization, content recommendation, and automating certain processes, making the website more engaging and efficient for users. Additionally, AI-powered features can help differentiate my website from competitors and attract users looking for innovative solutions.
- **Chatbox with AI:** Integrating a chatbot with AI capabilities can improve customer support and interaction on my website. Users can get immediate assistance, answers to their queries, and guidance through the CV evaluation process. A well-designed chatbot can enhance user satisfaction, reduce support costs, and streamline communication, ultimately leading to a better overall user experience.
- **CV Evaluation Services:** Offering CV evaluation services can be a valuable resource for job seekers, helping them improve their chances of success in the job market. By leveraging AI algorithms, I can provide insightful feedback, identify strengths and weaknesses in CVs, and offer personalized recommendations for improvement. This service can attract job seekers to my website and establish it as a trusted resource in the industry.

- **Payment for Additional Functions:** While the idea of implementing a payment system for additional functions may initially seem appealing for generating revenue, it's crucial to consider user preferences and the value proposition it offers. If I believe that users may not find the additional functions compelling enough to pay for, or if it adds unnecessary complexity to the user experience, then it makes sense to exclude it from my website. Instead, I can focus on monetization strategies that align more closely with my users' needs and preferences, such as offering premium services or partnering with other businesses for mutually beneficial arrangements.


6. Critically evaluate the strengths and weaknesses of your business application and fully justify opportunities for improvement and further development(D3).

- **Strengths:**
 - **User Experience (UX) and User Interface (UI):** The application excels in providing a user-friendly interface with intuitive navigation and visually appealing design. Users find it easy to interact with various features, enhancing overall satisfaction and engagement.
 - **Abundance of Features:** The application boasts a wide array of features, catering to diverse user needs and requirements. This comprehensive feature set adds value to the user experience and increases the application's utility for its target audience.
 - **Functionality:** Core functionalities of the application are robust and efficient, providing users with the tools they need to accomplish their tasks effectively. This enhances user productivity and fosters loyalty towards the application.
- **Weaknesses:**
 - **Incomplete Features:** Some features, such as the chatbox, are not fully implemented or optimized. This can lead to user frustration and dissatisfaction, as they may encounter functionality gaps or limitations when trying to utilize certain aspects of the application.
 - **Website Optimization:** The application's website is not yet optimized for performance, potentially leading to slow loading times, poor responsiveness, and suboptimal user experiences. This can adversely impact user retention and conversion rates, especially in

today's fast-paced digital environment where users expect instant access to information and services.

- Opportunities for Improvement and Further Development:
 - Complete and Enhance Incomplete Features: Allocate resources to finalize and enhance incomplete features, such as the chatbox, to provide users with a seamless and fully functional experience. Solicit user feedback to prioritize feature improvements based on user needs and preferences.
 - Optimize Website Performance: Invest in website optimization strategies to improve loading times, responsiveness, and overall performance. This may involve optimizing images and multimedia content, leveraging caching mechanisms, and implementing responsive web design principles to ensure a consistent user experience across devices.
 - Continuous Iteration Based on User Feedback: Establish a feedback loop with users to gather insights on their experiences, pain points, and feature requests. Use this feedback to inform iterative improvements and prioritize development efforts to address the most pressing user needs and concerns.
 - Explore Emerging Technologies: Stay abreast of emerging technologies and industry trends to identify opportunities for innovation and differentiation. Consider incorporating new technologies, such as artificial intelligence, machine learning, or augmented reality, to enhance the application's capabilities and user experience.
 - Invest in Quality Assurance: Strengthen quality assurance processes to identify and address issues proactively throughout the development lifecycle. Implement rigorous testing protocols to ensure the stability, reliability, and security of the application across different environments and usage scenarios.

REFERENCES

1. Allam, A. M., n.d. [Online]
Available at: Allam, A. M. (2023) 8 top system design drawing tools for software developers  , DEV Community, DEV Community, [online] Available at: <https://dev.to/abdelrahmanallam/8-top-system-design-drawing-tools-for-software-developers-3ol7> (Accessed 4 April 2022)

2. Anon, n.d. [Online]
Available at: Anon (2024) Top front end technologies you must know [2024], InterviewBit, [online]
Available at: <https://www.interviewbit.com/blog/front-end-technologies/> (Accessed 4 April 2024).
3. Anon, n.d. [Online]
Available at: Anon (n.d.) C#: Modern, open-source programming language for .NET, Microsoft, [online]
Available at: <https://dotnet.microsoft.com/en-us/languages/csharp> (Accessed 4 April 2024a).
4. Anon, n.d. [Online]
Available at: Anon (n.d.) Compare Draw.io vs. Figma, [online] Available at:
<https://www.g2.com/compare/draw-io-vs-figma> (Accessed 3 April 2024b).
5. GeeksforGeeks, n.d. [Online]
Available at: GfG (2023) List of backend technologies, GeeksforGeeks, GeeksforGeeks, [online] Available
at: <https://www.geeksforgeeks.org/list-of-backend-technologies/> (Accessed 4 April 2024).
6. GeeksforGeeks, n.d. [Online]
Available at: GfG (2024a) SDLC V-model - software engineering, GeeksforGeeks, GeeksforGeeks, [online]
Available at: <https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/> (Accessed 4 April 2024).
7. GeeksforGeeks, n.d. [Online]
Available at: GfG (2024b) Waterfall Model - Software Engineering, GeeksforGeeks, GeeksforGeeks, [online]
Available at: <https://www.geeksforgeeks.org/waterfall-model/> (Accessed 4 April 2024).
8. Gehman, n.d. [Online]
Available at: Gehman, C. (n.d.) What is source control and why is it important?: Perforce, Perforce
Software, [online] Available at: <https://www.perforce.com/blog/vcs/what-source-control> (Accessed 4
April 2024).
9. MozDevNet, n.d. [Online]
Available at: MozDevNet (n.d.) What is a web server? - learn web development: MDN, MDN Web Docs,
[online] Available at: [https://developer.mozilla.org/en-
US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server) (Accessed 4 April 2024).