2024

Lanka Nippon Biztech Institute

SKILL HUNT PROJECT

JOB PORTAL SYSTEM

ONELI JAYODYA WIJESINGHE

SE BATCH 01

UGC0122015

DEDICATION

To express my thanks to all those who contributed to many
ways to the success of this study and help me.
To my dear parents thank you for giving me the
support to reach my dreams.
I would like to sincerely thank my lecturers for
your guidance, support, and patience throughout this project,
thank you for the comments and suggestions you shared
that are beneficial in the completion of this project.
I apologize for being a headache to you when
I was doing this study.
To all my lecturers, friends, and job finders, I also dedicate
this to all of you. Thank you very much

ACKNOWLEDGMENT

We would like to acknowledge everyone who played a role
in the success of the project.
First of all, our parents, who supported us with love and understanding,
without you all, we would never have reached this current level of success.
To our lecturers, we are deeply grateful and we acknowledge
your deep sense of gratitude, love, and a constant source
of inspiration and motivation. Most of all,
we offer our sincere appreciation for teaching us how to become
a fulfilled project until the end of our project.
Thank you for your full support, tips on
how to present our project, and all the efforts that you made
for the success of our project. Again, to all those people
who help us, guide us, and keep us all motivated we all want you to know
that we truly appreciate all of your love and support,
thank you for all the patience, time, and all the efforts that you made for us.
This experience will keep us all stronger and will keep us motivated
to create better projects in the future.

LIST OF ACRONYMS

DDL	-	Data Definition Language
ER	-	Entity Relationship
SQL	-	Structured query language
DML	-	Data Manipulation Language

Table of content

FIGURES AND TABLES

Chapter 1 : INTRODUCTION

1.1. Introduction

My project is a job portal website. In this chapter, you can learn the idea of this report and what will be discussed in this report. First, introduce my system and why I chose this system to build, including the motivation behind the project, the Aims, and Objectives of the Project, we will discuss the scope of this project.

1.2. Introduction of the system

This system's name is Skill Hunt. This software revolves around connecting students and undergraduates with job opportunities. The system specifically focuses on part-time and internship jobs, catering to the needs of undergraduates and students seeking to gain work experience and build their careers. Skill Hunt uses location-based features to help students find jobs. This system keeps members informed about job interview scheduling. Users and companies can register for this system and companies can post their jobs. There are job categories. The companies can post their jobs in a relevant job category.

Reasons for developing this system because students and undergraduates often face difficulties finding suitable part-time and internship opportunities that align with their skills. Some systems may not notify interview times so a lack of centralized job notifications can lead to missed opportunities. Skill Hunt's unique features, such as location-based job search and interview notification. Because some job systems do not provide a chosen area and show jobs near that area.

There are some similar job systems in global platforms like LinkedIn, indeed, Handshake provides a wide area of the world. They offer features like filtering jobs, job alerts, and messaging for candidates. But in the local platform, there are similar system names like top jobs and OnlineJobs.lk, Xpress Jobs. They provide location-based but full-time jobs some systems do not give interview notifications and they do not provide jobs for students. That system can filter jobs, notifications about updated jobs, and show jobs around the location of the district.

This system allows users to search for jobs based on location and industry. Priorities job listings and save time and effort. Enable registered members to connect directly with job owners. Allow job owners to schedule interviews with candidates directly through given by notification. Offer exclusive benefits to registered members such as messaging, interview notifications, and job alerts. Job owners can post jobs and if is there any new category of the job then the employee can add the category throw the admin confirmation.

1.3. Motivation

The motivation behind the development of Skill Hunt is to bridge the gap between students seeking part-time and intern opportunities. I started to create this system because there are no central locations to look up jobs near them.

1.4. Aims

The primary aim of the Skill Hunt system is to connect undergraduates and students with part-time and internship opportunities by providing a user-friendly platform that is location-based job search, communication between job seekers and owners, interview notification, and member experience.

1.5. Objectives of the Project

Objectives:

- Connect students with relevant part-time and internship opportunities.
- communication between job seekers and job owners.
- Provide exclusive features and benefits for registered members, such as the ability to message job listings and receive interview notifications.
- Develop a platform where companies can post part-time and intern job listings.
- admin can manage particularly when adding new job categories.

1.6. Scope of the Project

In-Scope Functionalities:

- User Registration and Membership
- Job Posting
- Notification System
- Location-Based Job Search
- Search Filters
- Add job categories
- Profile Enhancement

Out-of-Scope Functionalities:

- Resume Builder
- Company Pages
- Accessibility Features
- Payment Processing

1.7. Methodology

Skill Hunt is developed using a modern tech stack, with the front end built on HTML, and tailwind CSS for an intuitive user interface. Eclipse is used for admin part design. Databases are designed by using MYSQL Workbench. Frameworks are used for designing the website for example tailwind, Laravel, and Breeze using Vs Code. These tools and technologies are used to design the system.

1.8. Novelty of the system

Skill Hunt as a novel solution in the system is:

1. Location-Based Job Search:

Skill Hunt's connecting students with nearby job opportunities is a unique feature that addresses a common challenge faced by students seeking part-time and internship roles. By prioritizing job listings based on the student's location, the platform saves time and reduces commuting costs.

2. Interview Notifications:

The interview notifications within the platform reduce the potential for miscommunication and scheduling conflicts. This feature ensures that both students and job owners can efficiently manage interview arrangements and stay informed about upcoming interviews.

3. Member-Centric Approach:

Skill Hunt prioritizes the needs- of registered members by offering exclusive features such as job alerts, and personalized profiles.

4. Focus on Students and undergraduates:

While many job systems are used for a wide range of job seekers, Skill Hunt specifically targets undergraduates and students seeking part-time and internship opportunities.

1.9. Summary

In this chapter, we gave an introduction to the system and discussed the motivation to build this system, aims, objections, and solutions of the system. The scope of the Skill Hunt project is the development of a web and mobile platform and intern job opportunities. Key features include a targeted job search with location-based filtering. Direct messaging between job seekers and owners, interview notification system. Membership-based access to enhanced features. In the next chapter, we will discuss the analysis part of the system and what are the functions of the system.

Chapter 2 : ANALYSIS

2.1 Introduction

In the previous chapter, we discussed the system and the motivation to build this. This system aims to connect undergraduates and students with part-time and internship opportunities by providing a user-friendly platform. We discuss what are the scope and methodologies used for the system. In this chapter, we will discuss the functions and non-functions of this system and what process model is used for this system.

2.2 Review Similar Systems with References

There are some similar job systems like LinkedIn, indeed, Handshake, topjobs, and OnlineJobs.Lk, Xpress Jobs provides a wide area of the world. They offer features like filtering jobs, job alerts, and messaging for candidates. They provide sometimes location-based but full-time jobs some systems do not give interview notifications and they do not provide jobs for students.

2.3 Identifying/prioritization the functional and non-functional requirements

Functional requirements

1. Registration and login
    - User registration and login
    - Company registration and login

2. Search job
    - Search by location
    - Search by title
3. Job posting
    - Company Employees can post new jobs
4. Notification system
    - View the Interview schedule message
5. Delete account
    - Delete user and company account
6. Manage category
    - Admin can Add, delete, update, and view job categories to the list
7. Apply for a job
    - Member can apply for jobs

Non-functional requirements

1. Payment Process
    - Sign-in payment
    - Company monthly payment

2 Build Profiles
- User can build a profile
- The company can build a profile

3 Messaging
- Member can message the post and the company can reply to it

2.4 Identifying the suitable process model with a justification

First, I identify all the requirements for this system. So based on that implement the system. Among the process models waterfall model is used for this. The Waterfall Model is a traditional software development methodology that follows a linear and sequential approach.

Requirements Gathering:

Define and document the functional and non-functional requirements of the Skill Hunt system. Identify the features such as user registration, job search, messaging, membership, notification system, and admin functionalities.

System Design:

Create a detailed system architecture that includes components like user registration, job database, membership management, and notification system. Define the database schema for storing user information, job details, and other relevant data. Specify the user interface design for member registration, and job search.

Implementation:

Develop the core functionalities of the Skill Hunt system based on the design specifications. Implement user registration, membership management, job search, and notification system for interview alerts. Create an admin panel for CRUD (Create, Read, Update, Delete) operations and category management.

Testing:

Conduct unit testing to ensure each component works as intended. Perform integration testing to verify the proper functioning of the entire system.

Deployment:

Ensure that the system is accessible and stable for users.

Maintenance and Support:

Implement updates and improvements based on user feedback or changing requirements.

2.5 Summary

In this chapter, we discussed about analysis part of the system. We used the waterfall method for this system. We identified what are the functions and non-functions of this system. We know about what are similar systems like this system and the mistakes they have. In the next chapter, we will design the diagrams for the system. We will discuss how the system attaches to users.

Chapter 3 : DATA DESIGN

3.1 Introduction

In the previous chapter, we discussed the analysis part of the system and found what are the functions of the system. In this chapter, we will show the diagrams related to the system and how we design our system. Next, we will discuss user roles, ER diagrams, and SQL queries. Finally, you can get a clear idea about this before implementing the system.

3.2 Introduction to Client

This system's name is Skill Hunt. This system allows students and undergraduates to find jobs easily. Some companies can register to the system and post the jobs. Then job seekers can apply for jobs. Then the companies can see how many job seekers apply for their jobs and who are they. The user can register to the system as a student and then can search for jobs on location.

Reasons for developing this system because students and undergraduates often face difficulties finding suitable part-time and internship opportunities that align with their skills. This system allows users to search for jobs based on location and industry. Priorities job listings and save time and effort.

3.3 Scope Identification

The scope of the Skill Hunt project is the development of a web and mobile platform and intern job opportunities. Key features include a targeted job search with location-based filtering. Direct messaging between job seekers and owners, interview notification system with job updates. Membership-based access to enhanced features.

3.4 Use case diagram

3A use case diagram is a visual representation of the interactions between actors and a system under consideration. It depicts the functionality or behavior of a system from the user's perspective. Use case diagrams capture the functional requirements of a system and help to identify how different actors interact with the system to achieve specific goals or tasks.

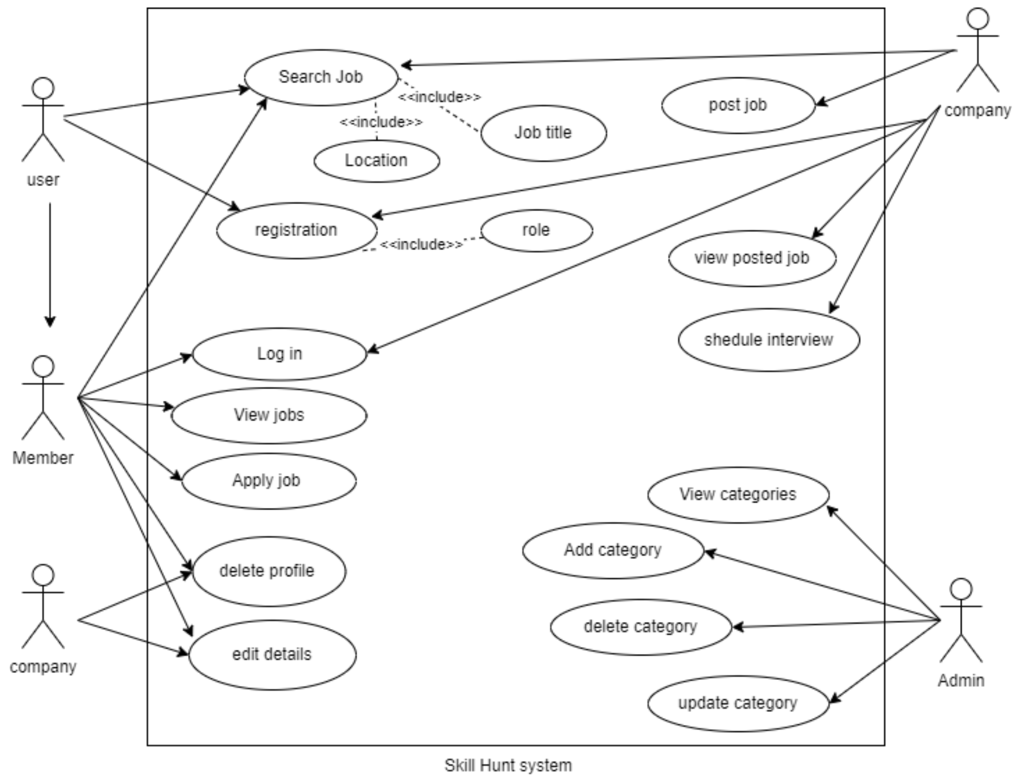


Figure 3.1: Use case diagram of Skill Hunt system

Use case diagrams provide a high-level overview of the system's functionality, showing the different features or capabilities it offers and how users or external systems interact with it. They serve as a communication tool between stakeholders, helping to clarify and validate requirements, identify system boundaries, and support the development and testing processes. Figure 3.1 shows how the user interacts with the Skill Hunt system. (see Figure 3.1)

3.4.1 Identification of Roles

Identifying the roles involved in a project is crucial for effective project management. Different roles have specific responsibilities and accountabilities, and understanding these roles helps ensure that tasks are assigned appropriately. Actors are external entities that interact with the system. Identifying actors and their roles is a crucial step in use case analysis. In this use case diagram (see Figure 3.1) there are 3 actors Admin, Member, Company. Admin handles the categories. The student can search for jobs and apply for them. They are the members of the system. Non-registered people are users. Job posted by Company.

3.4.2 Identification of Cases

The identification of use cases helps define the system scope, ensuring that the requirements to be found will all be aligned with the business values, needs, and strategy. In above use case diagram shows how actors interact with the skill hunt system. (see Figure 3.1) first, we identified what are the roles now we will the what are the cases. Users can search for jobs and view jobs. If the user wants to apply for a job, then the user should get a membership by registering to the system. The user becomes a member.

So, members can log in to the system, view and search for jobs, apply for jobs, and view the message. Users can search for jobs by job title and location. The company can also log in to the system. They can post jobs, view posted jobs, and schedule interviews with applied members. Both Member and companies can delete their profiles and edit details. There is admin also. Admin can view, add, delete, and update the categories.

3.4.3 Identification of Dependencies

Dependency relationships between use cases indicate that one use case relies on another, but it's not necessarily a direct association or inclusion. They can be used to show how use cases are related to each other and how they interact. In the use case diagram shown above (see Figure 3.1) you can see two relations one is Association and the other one is include relation. Users can search for jobs based on selecting job titles and locations. It is a must when you want to search for a job. So, the job title and location are included in the search job use case. Also, registration case you want to select a role because your activities depend on which role you selected. These two cases include relation. Actors and use cases connect with association relations. These are the dependencies of this use case.

3.5 Class Diagram

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling, translating the models into programming code. Class diagrams can also be used for data modeling.

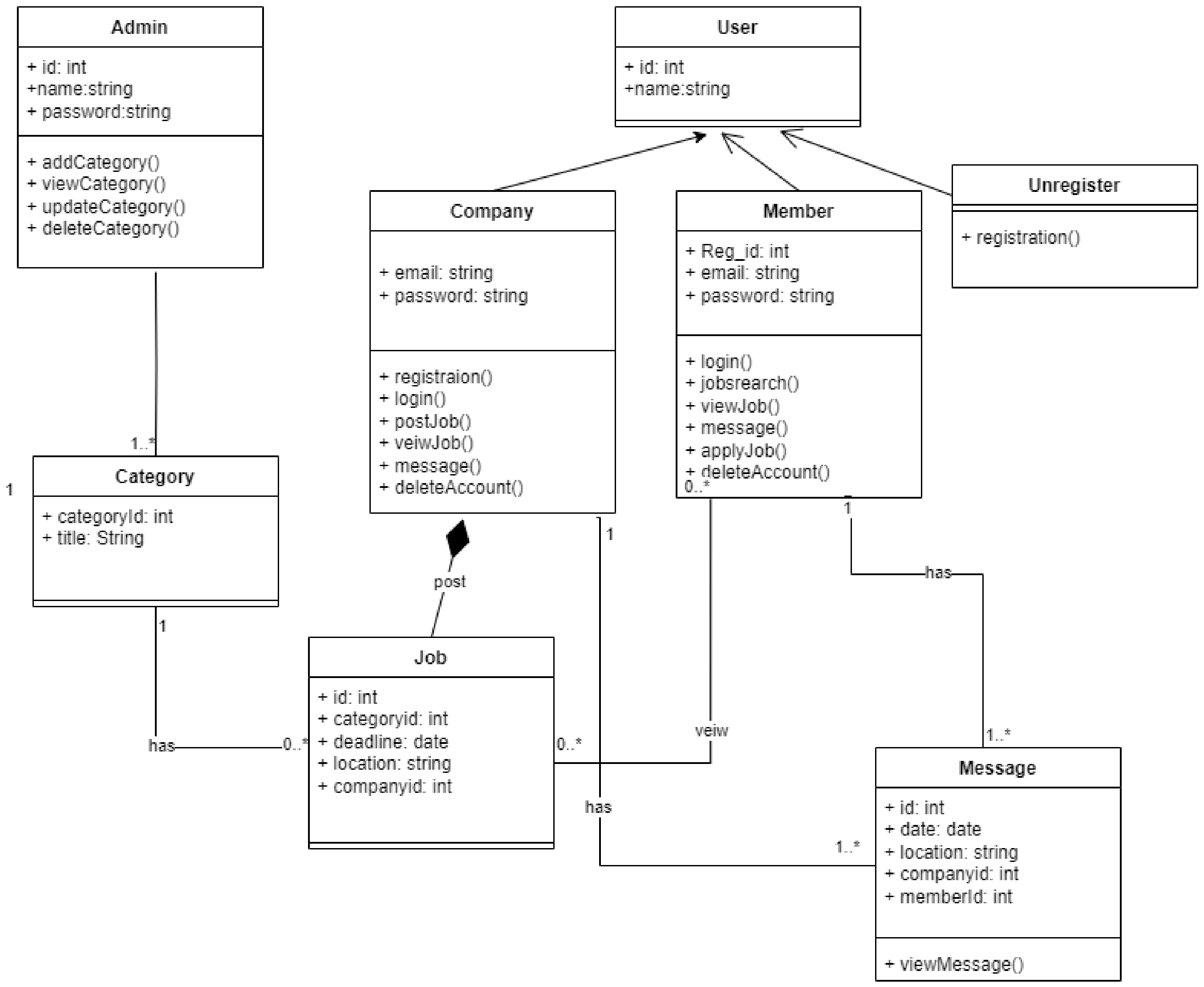


Figure 3.2: Class diagram of skill Hunt

The above diagram shows the class diagram of the system. (see Figure 3.2) There is a user class generalized to company, member(student), and unregistered users. Job class whole depends on company class. The company has registration, login, post job, Delete account, and view job functions. Admin class manages the category class.

3.6 Sequence Diagram

A sequence diagram is a type of interaction diagram that shows how processes operate with one another and in what order. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the application. Now, we will see some Sequence diagrams in this system.

This diagram shows how the interactivity with the system when adding a

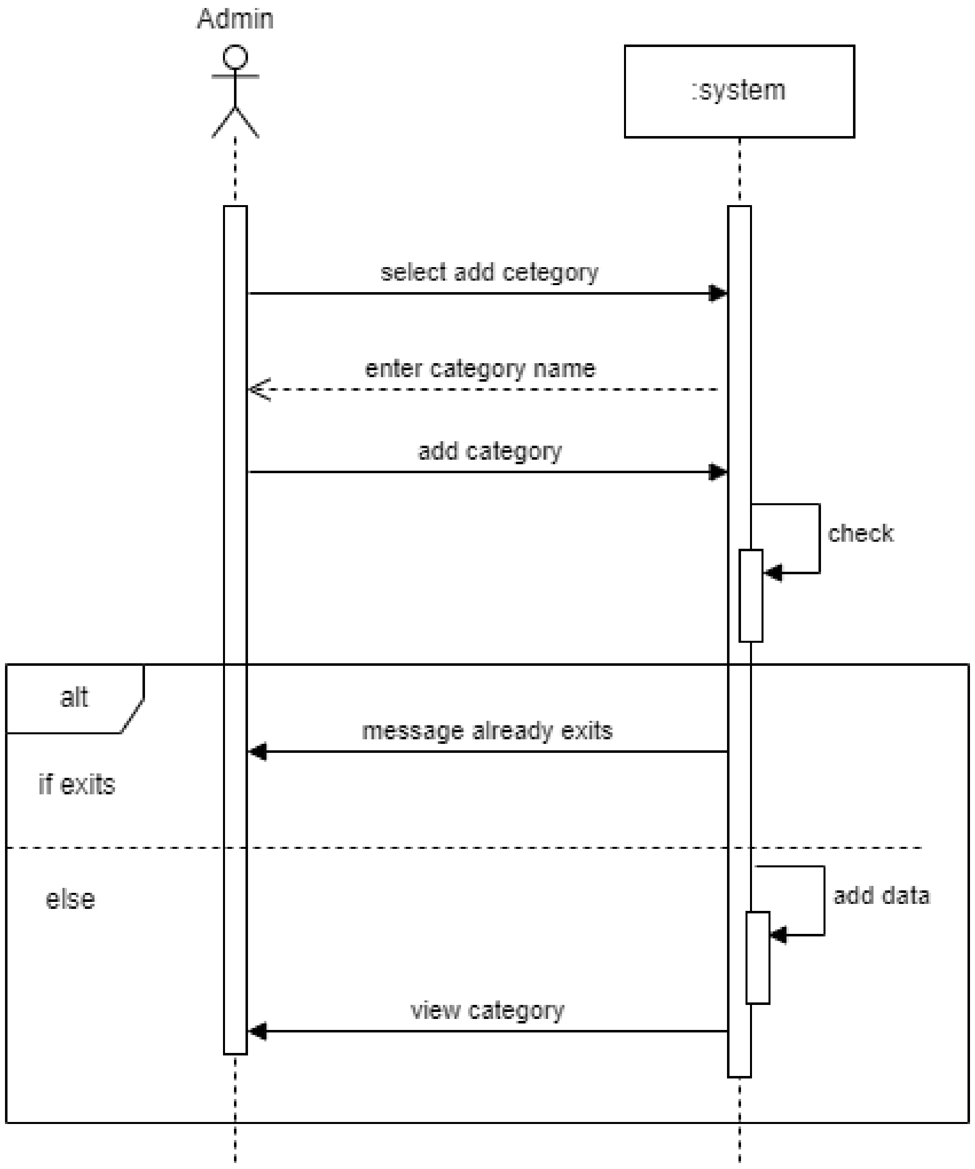


Figure 3.3: Sequence diagram of add category

category by admin. (see Figure 3.3) when admin selects add category the system shows enter the name of the category then the system adds the category to the database and checks whether that name has already been added or not. If it exits then show the message as already added. If it is not then show added successfully. Then you can see the categories that the admin added.

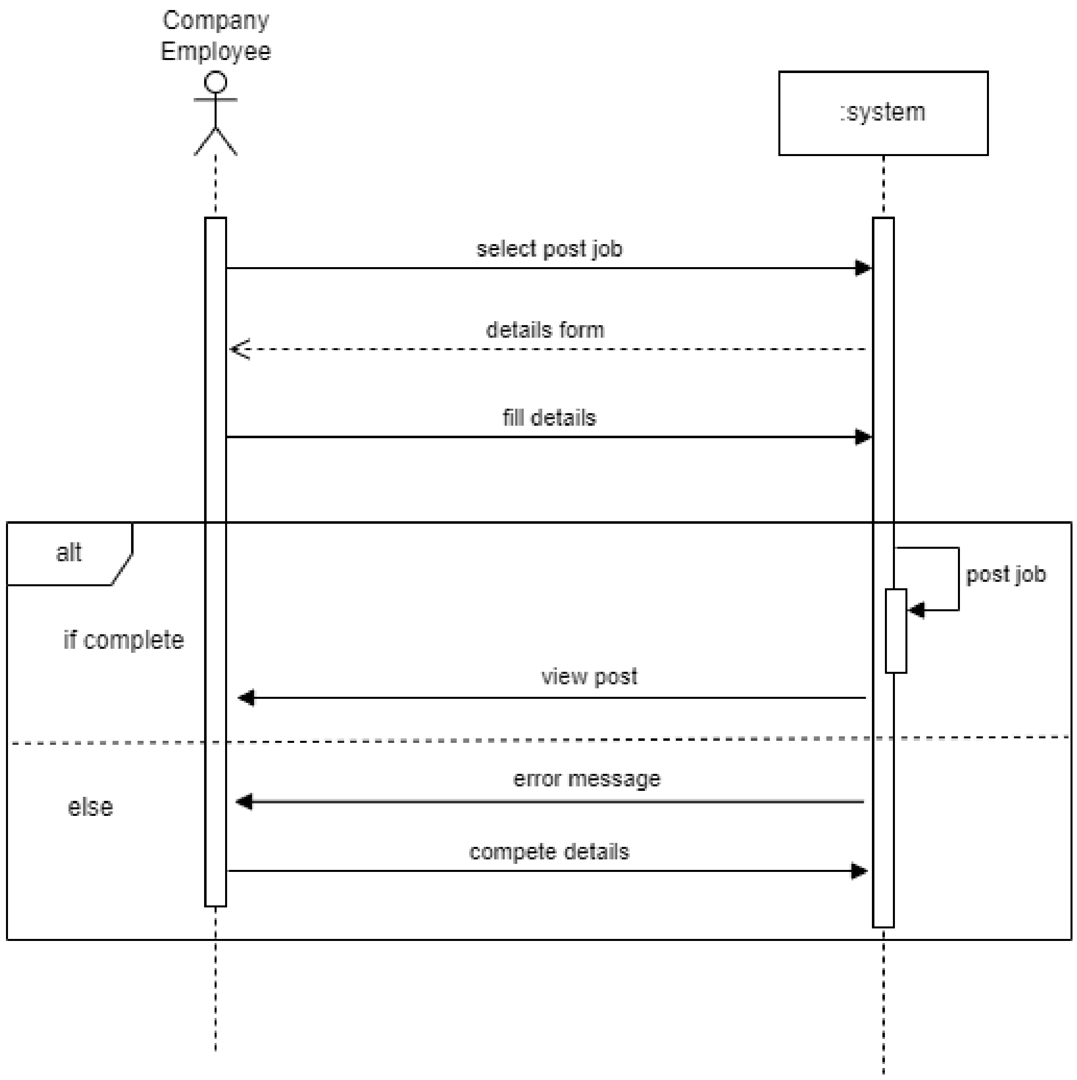


Figure 3.4: Sequence diagram of post job

This diagram shows how the company posts the job in this system. (see Figure 3.4) when the company selects a post job the system shows the job posting form. You can fill in details and submit the form then post that form. The company can view the post. If you do not fill it correctly the shows an error message.

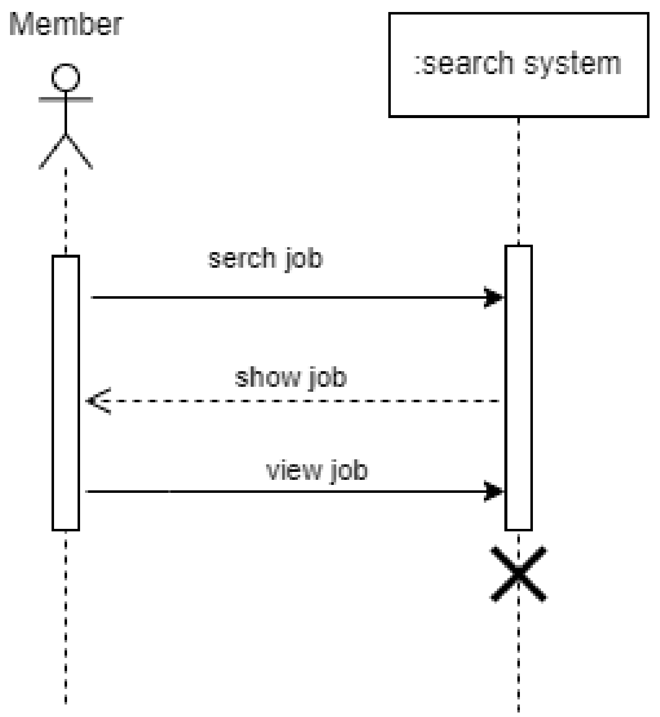


Figure 3.5: Sequence diagram of search job

This diagram shows the searching part of the system. (see Figure 3.5) When users register to the system they become members of the system. Then the member(student) has access to search for jobs. Selecting the job title and the location the member can search job then the system shows the relevant job for search and the member can view the jobs.

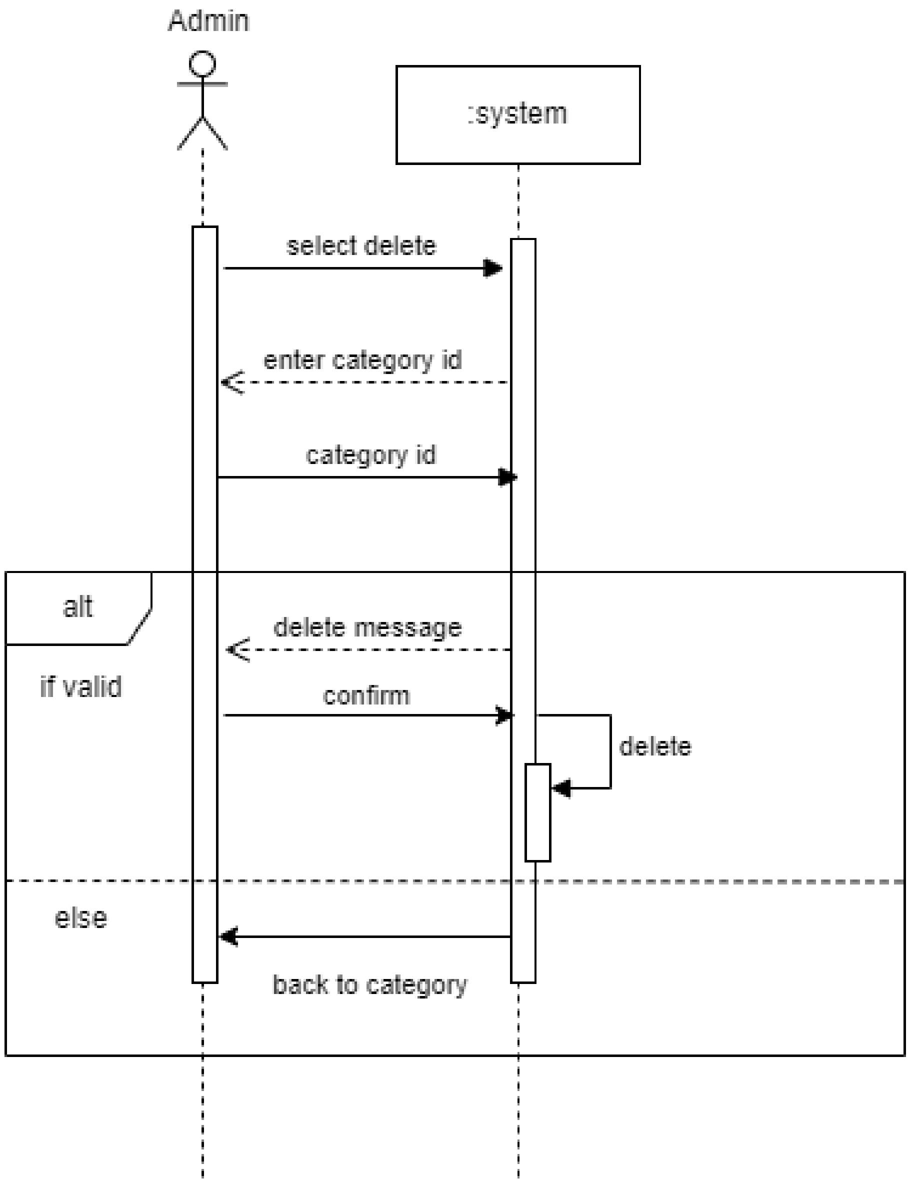


Figure 3.6: Sequence diagram of deleted category

This diagram shows how the admin deletes the category in the system. (see Figure 3.6) when the admin chooses the delete button then shows enter category and then enter the category after deleting the category when the user confirms the delete message. Then you can back to the main.

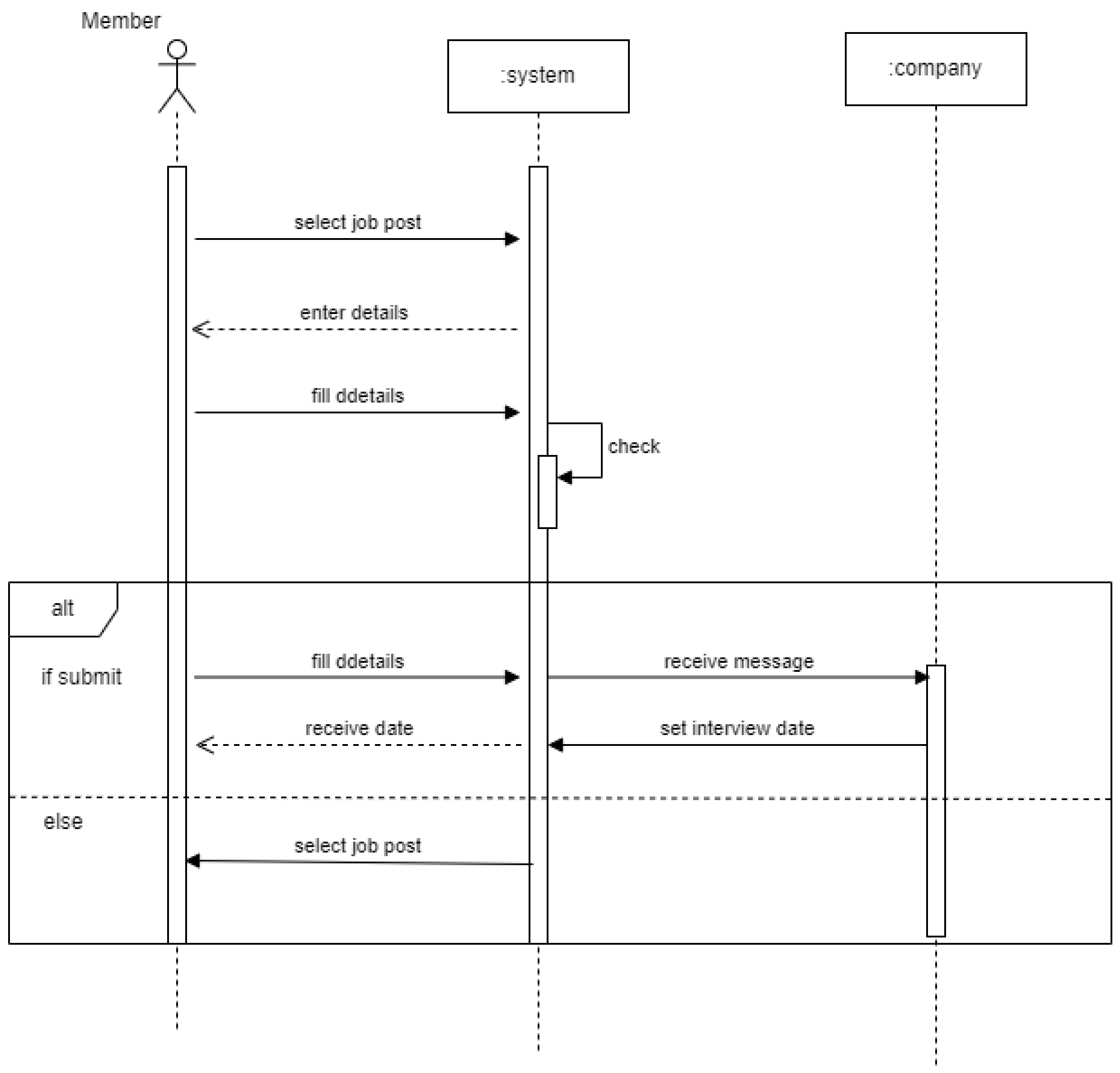


Figure 3.7: Sequence diagram of notification

This diagram shows the company and member notification system. (see Figure 3.7) when the job and posted by filling details the company can see who has applied and the company can give an interview date for that member. Then members can see the interview date.

The above diagrams represent sequences of the system and the end users. It helps to understand well how they work together.

3.7 State Chart

State diagrams are used to describe the behavior of a system, object, or part of a system. They show how an object moves from one state to another in response to events, including the actions that may result from those transitions.

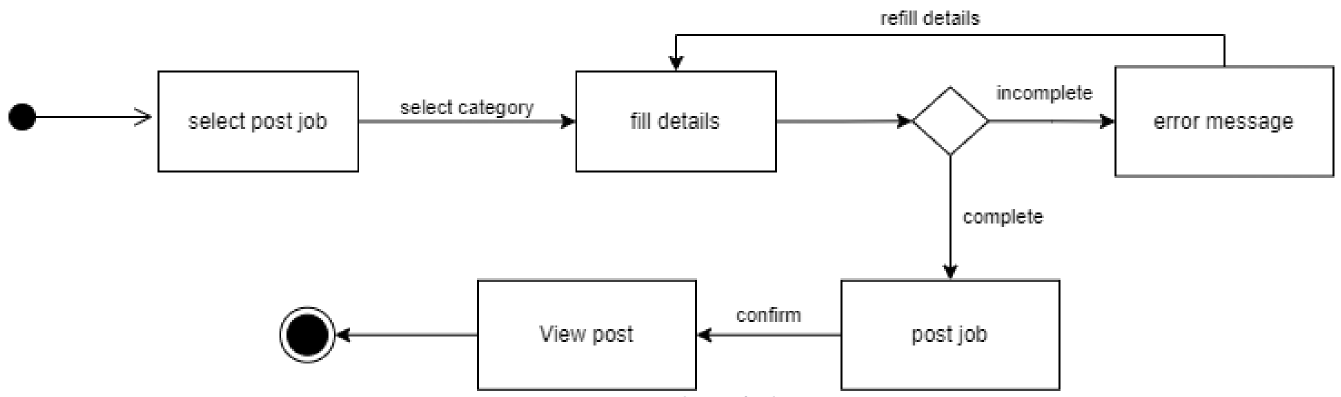


Figure 3.8: State chart of job post

The above diagram shows the job posting state. (see Figure 3.8) The possible outcomes are to post the job and view the post. First, select the posted job and then select categories on the form and fill details. After that check whether all details are fill not. If there is any blank they show an error message to the user. After correcting that then post the job.

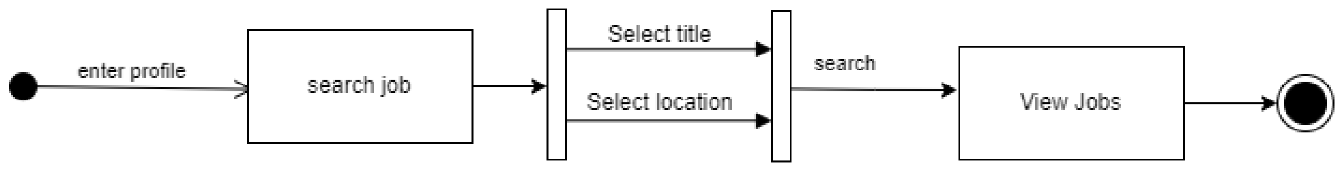


Figure 3.9: State chart of search job

This diagram shows the search job state. (see Figure 3.9) when the user login to the system then can see the dashboard. If members want to search the location then select title and location. This is done personally. After that members can view the results of the jobs.

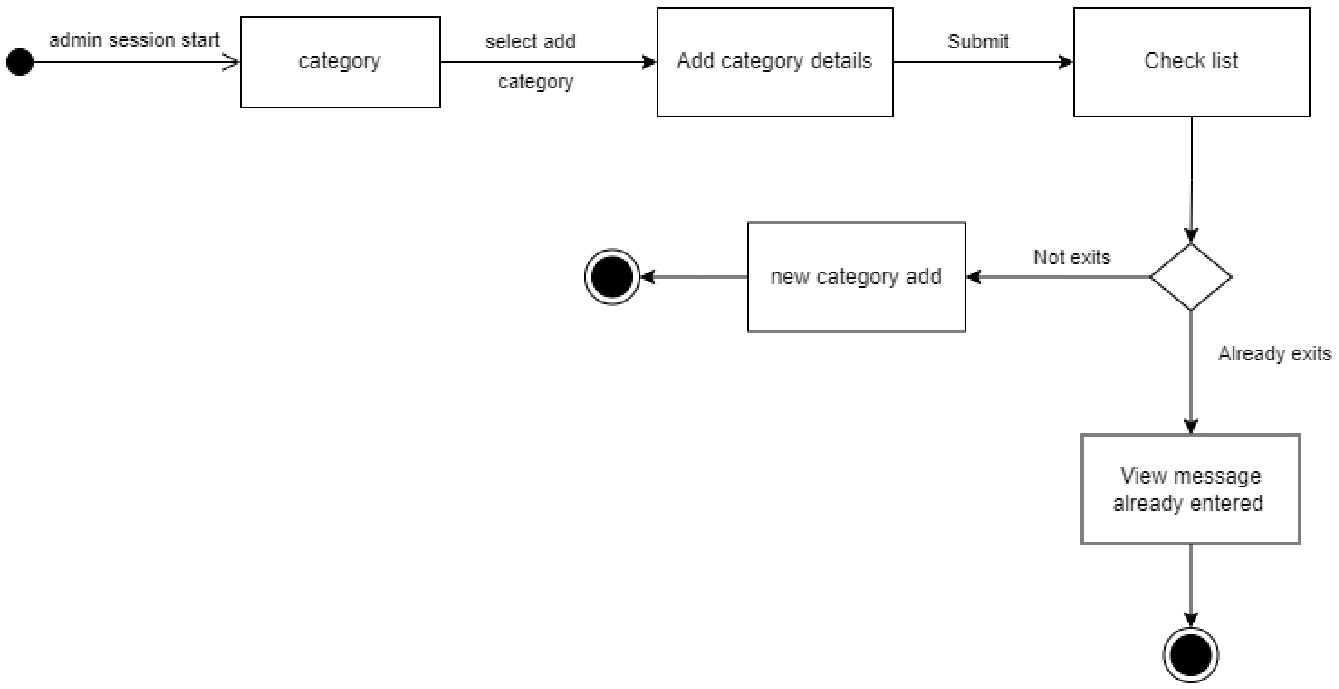


Figure 3.10: State chart of add category

This state chart represents the add category by admin. (see

Figure 3.10) When the admin session starts admin can choose category add. Then add the category name and submit. After that system checks whether it already add or not. If it corrects then shoe insert successfully message. If it is not then show already added.

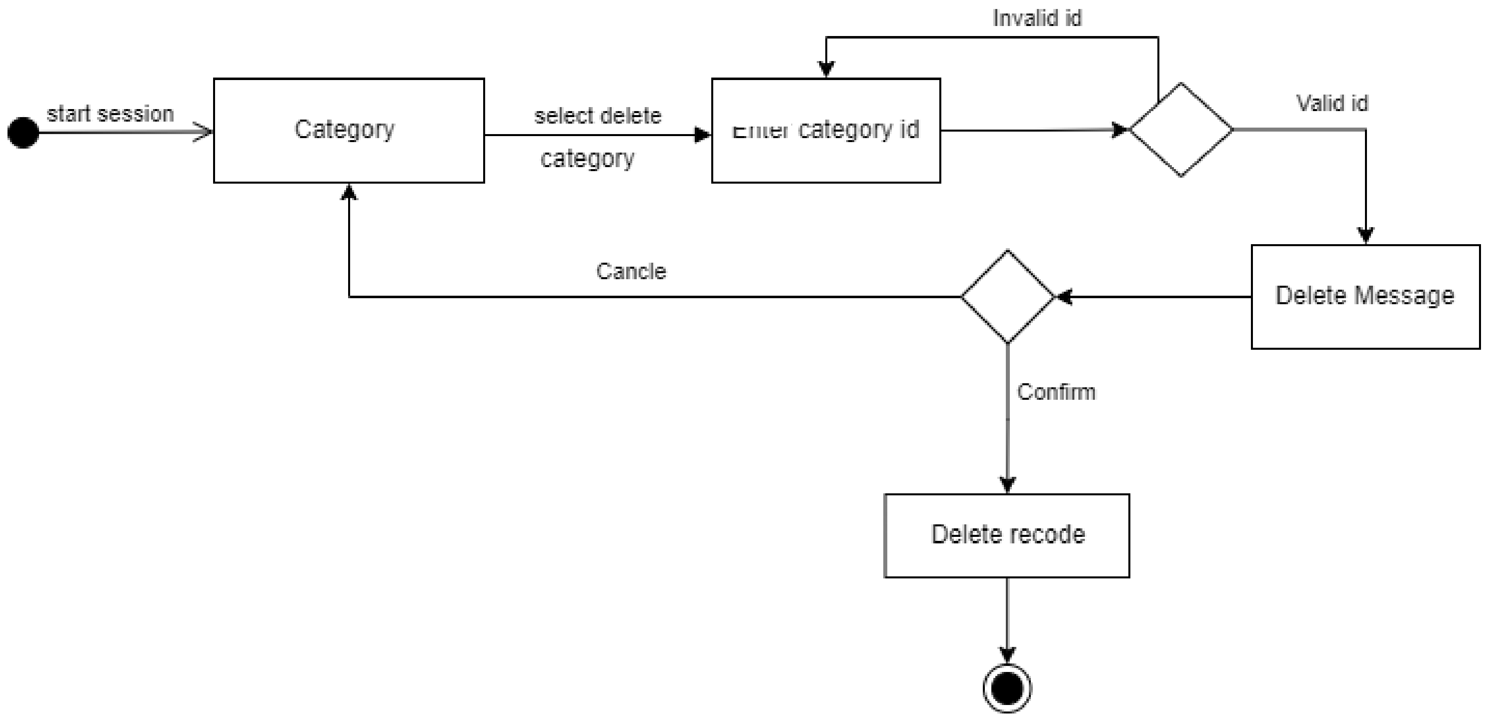


Figure 3.11: State chart of deleted category

This represents the delete category state. (see Figure 3.11) first admin wants to delete the category then select delete the category. After that admin selects the row of what they want and then selects delete. Then checks id are the same by the system then shows deleted successfully. If there is nothing to delete or not select show the select the row first.

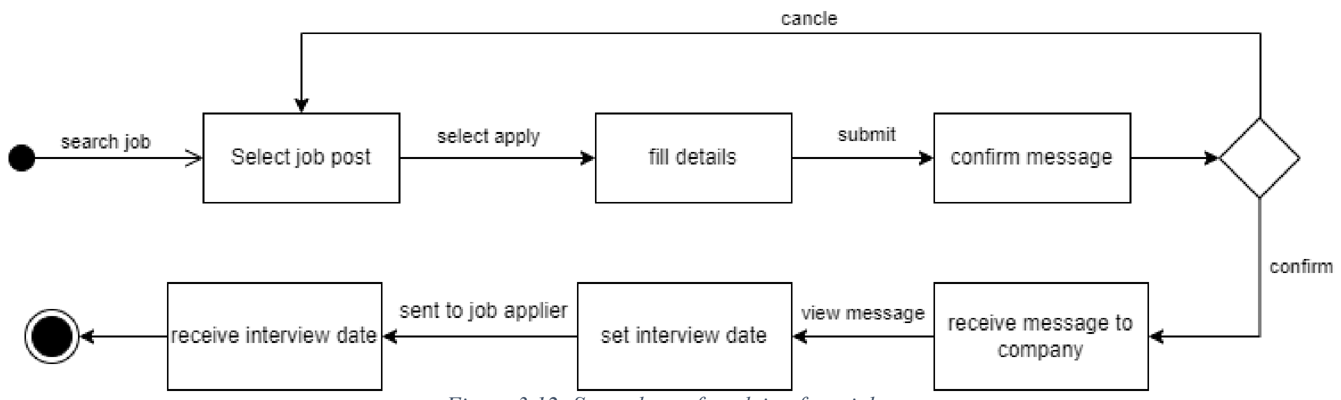


Figure 3.12: State chart of applying for a job

The above diagram shows the job posting state. (see Figure 3.12Figure 3.8) The possible outcomes are applying for the job and viewing the interview messages. First, select the job and then select Apply job on the list and fill details. After that check whether all details are fill not. If there is any blank they show an error message to the user. After correcting that then apply for the job. Then company receives the application message and gives an interview date for them.

The above diagram shows the states of the functions in the system. We can get a clear idea about how the flow goes. It helps to create easily the system.

3.8 Activity Diagram

Activity diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity. They may be used to detail situations where parallel processing may occur in the execution of some activities.

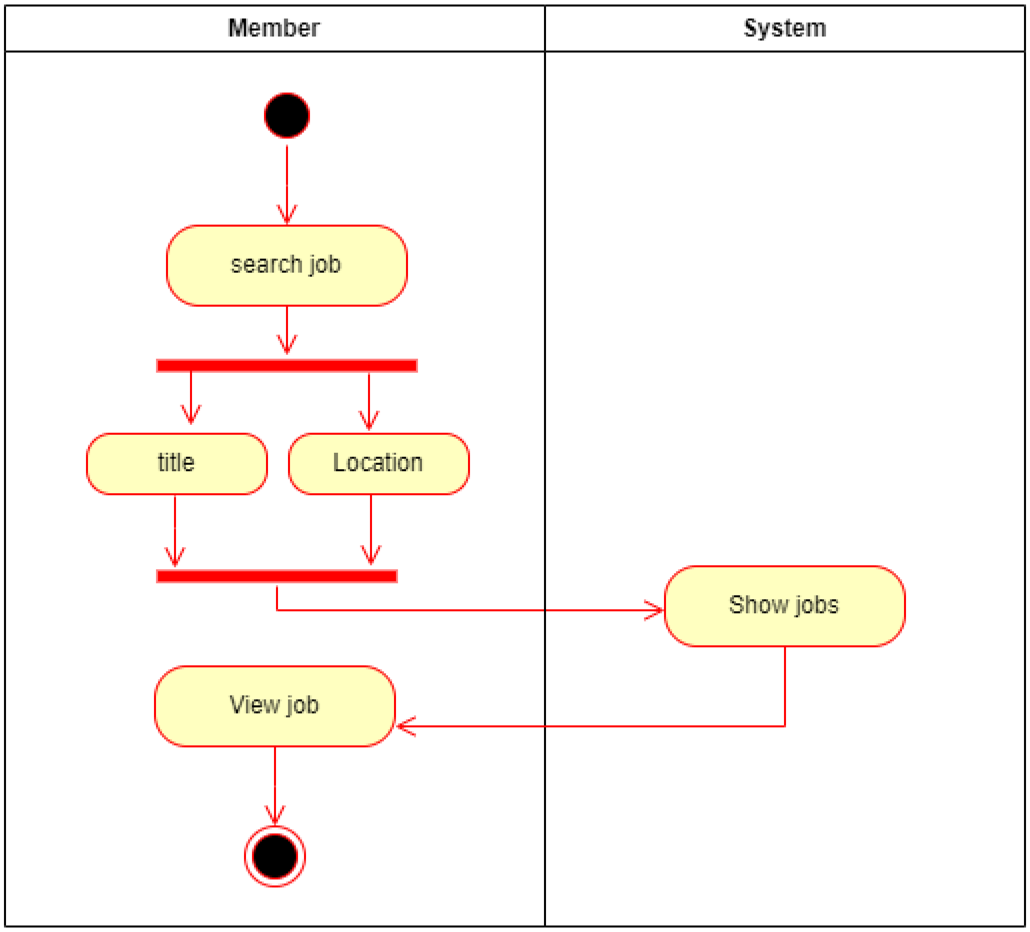


Figure 3.13: Activity diagram of search job

The above diagram shows how the search activity goes. (see Figure 3.13) when the search job first wants to select the title and the location after that you can search job and the system lists down the jobs. After members can view jobs.

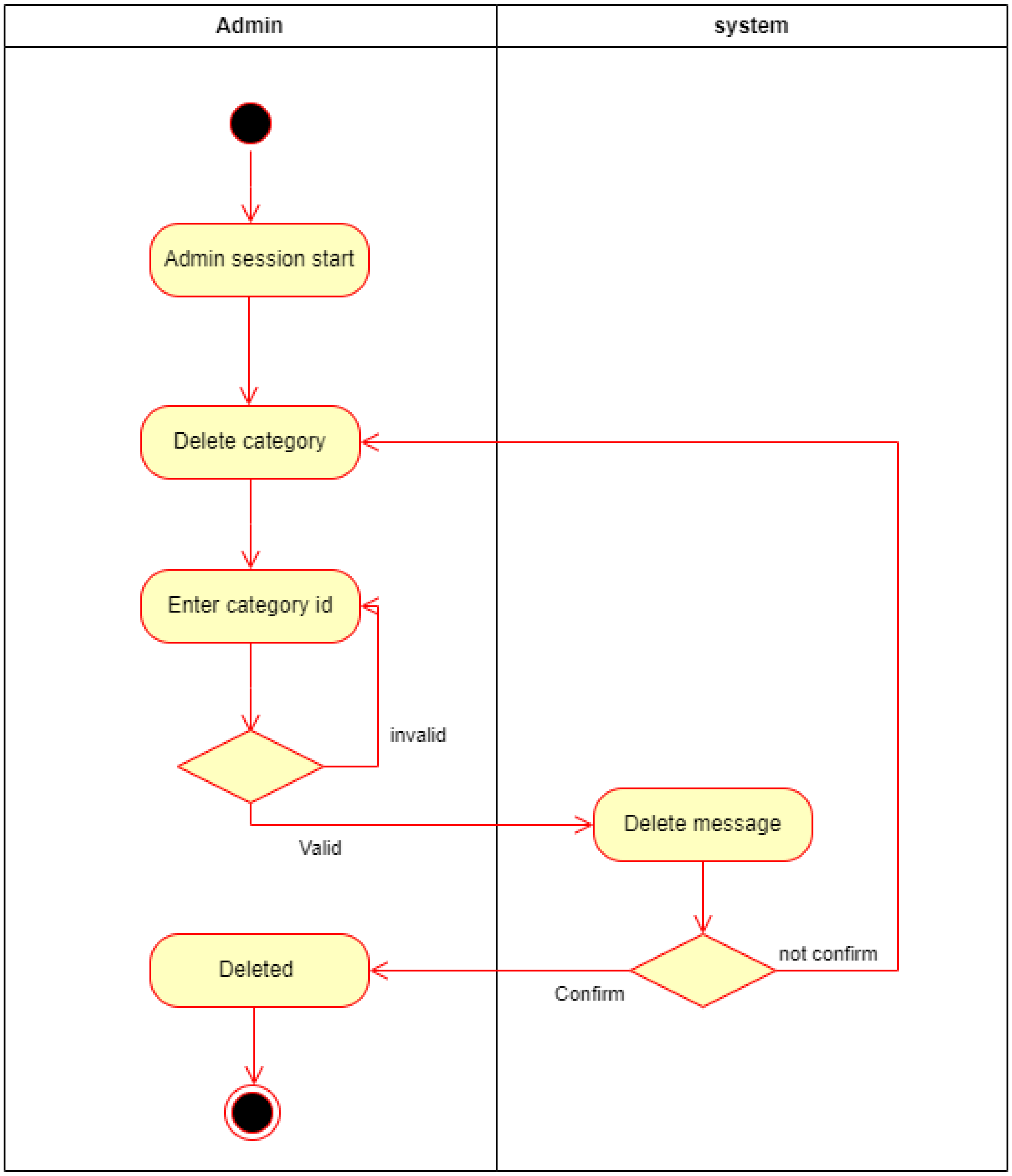


Figure 3.14: Activity diagram of delete category

This represents delete category activity. (see Figure 3.14) when admin selects delete category first choose the category they want to delete after that shows the confirmation message and confirms it then delete the category.

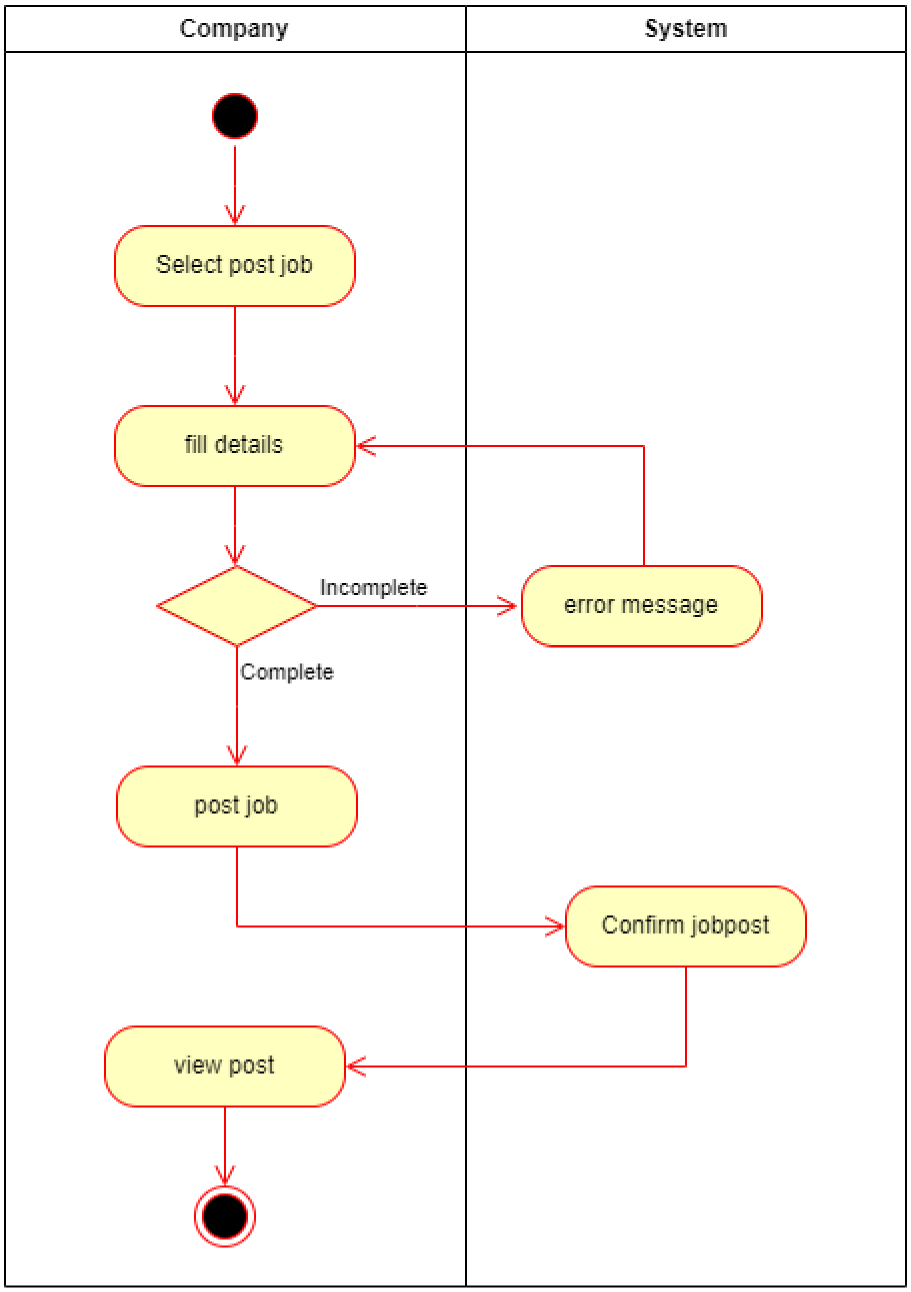


Figure 3.15: Activity diagram of post job

The image shows an activity of a job posting process. (see Figure 3.15) The process starts with the company selecting a post job. Then, the company fills in the details of the job. If the details are incomplete, an error message is displayed. If the details are complete, the job is posted. The company can then view the post. The system confirms the job post.

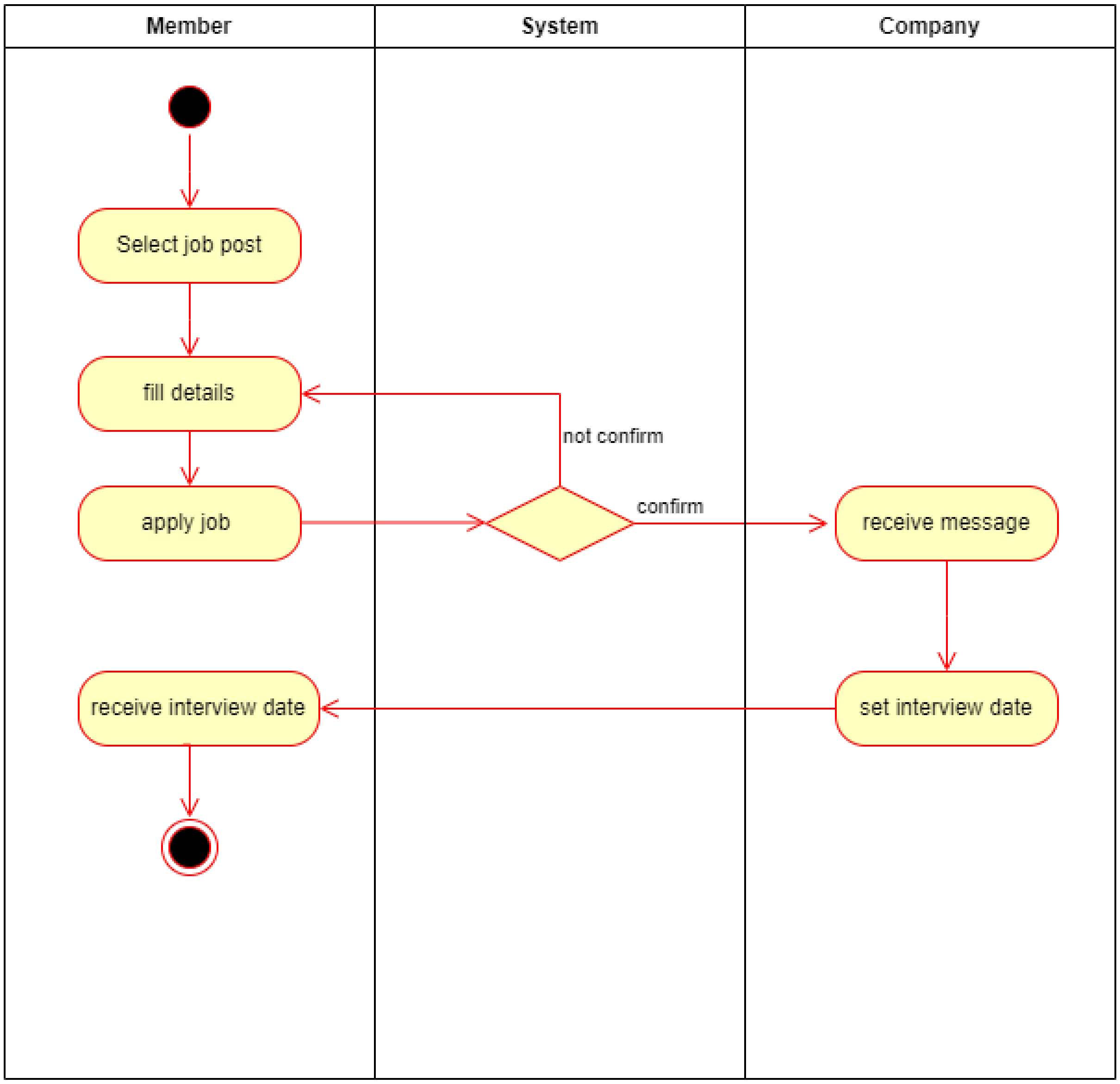


Figure 3.16: Activity diagram of applying for a job

The activity diagram shows the job application process. (see Figure 3.16) The process starts with the member selecting a job to apply. If they confirm the application, they fill in their details and apply for the job. The system then sends a message to the company and sets an interview date. The member then receives the interview date.

3.9 ER Diagram

ER diagrams are visual representations of the entities and relationships within a database. They help visualize the structure of a database and the connections between different Entities. ER diagrams are created using specific symbols to represent entities, attributes, and relationships. ER diagrams help identify potential problems early in the design phase.

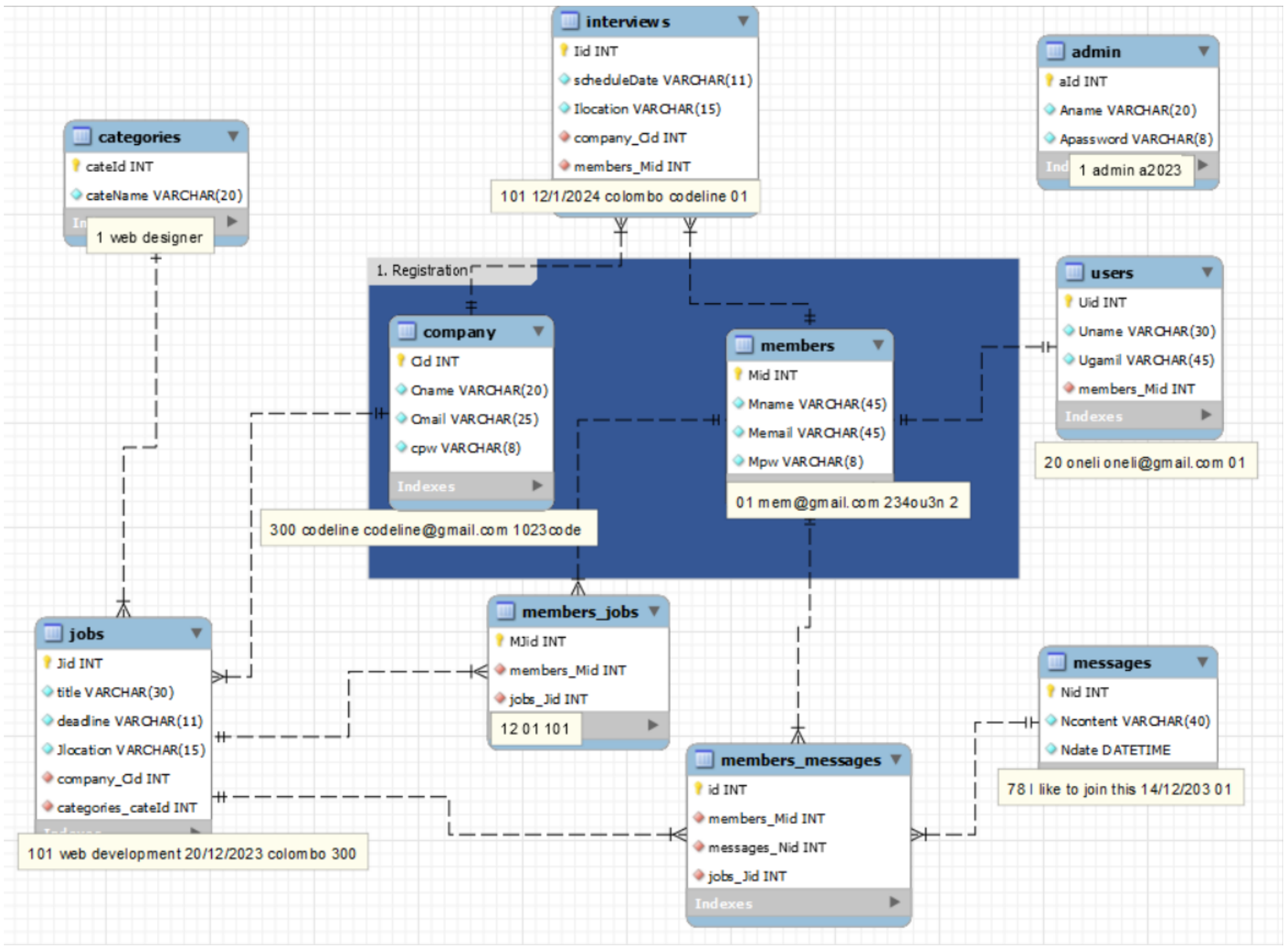


Figure 3.17: ER diagram of Skill-Hunt

This is the Entity relationship diagram of this skill hunt project. It shows what are the tables related to the system and relationships.

3.9.1 Identification of Entities

Entity in an ER diagram represents a real-world object, concept, or event. Entities are typically represented as rectangles in the diagram, and they have attributes that describe their characteristics. These are the entities of this system. (See Figure 3.17)

- Members: Registered users
- Companies: posting job
- Jobs: Job opportunities available
- Messages: Communication between members and companies
- Categories: Job classifications
- Interviews: Scheduled meetings between members and companies
- Admin: System administrators

3.9.2 Justification of Attributes

Attributes are included to include details of the various entities that are highlighted in a conceptual ER diagram. Key attributes are those that uniquely identify each entity within an ER diagram. Attributes are characteristics of an entity, a many-to-many relationship, or a one-to-one relationship. Multivalued attributes are those that can take on more than one value. In this diagram, there are attributes related to entities. (see Figure 3.17)

• Members: In the member table I get the name because it wants to show in profile, email, and password as login details.

• Jobs: In the job table, It has title, deadline, and location because when showing jobs we want to know when the deadline is and where it is held.

- Interview: In this table, when the user applies to the jobs the company gives a date for the user to interview then the location also shows.

3.9.3 Identification of Relations

The above diagram shows how entities are connected. (see Figure 3.17) Attributes are characteristics of an entity, a many-to-many relationship, or a one-to-one relationship. The relationships in the relational database schema are as follows:

The categories table has a one-to-many relationship with the jobs table, meaning that each category can have many jobs, but each job can only belong to one category.

The interviews table has a one-to-many relationship with the jobs table, meaning that each interview can only be for one job, but each job can have many interviews.

The interview table has a one-to-many relationship with the members table, meaning that each interview can only be conducted by one member, but each member can conduct many interviews.

The jobs table has a one-to-many relationship with the members jobs table, meaning that each job can be applied for by many members, but each member can only apply for one job.

The members table has a one-to-many relationship with the members jobs table, meaning that each member can apply for many jobs, but each job can only be applied for by one member.

The members table has a one-to-many relationship with the members messages table, meaning that each member can send many messages, but each message can only be sent by one member.

The jobs table has a one-to-many relationship with the members messages table, meaning that each job can receive many messages, but each message can only be sent to one job.

The admin table has a one-to-many relationship with the users table, meaning that each admin can create many users, but each user can only be created by one admin.

The members table has a one-to-many relationship with the users table, meaning that each member can create many users, but each user can only be created by one member.

These relationships are important because they allow us to understand how the data in the database is organized and how it can be accessed.

• Member-User: One-to-One (Each member can be one user)

• Company-Jobs: One-to-Many (A company can post multiple jobs)

• Jobs-Categories: One-to-Many (A job belongs to a specific category)

• Jobs-Members: Many-to-Many (Members can apply for multiple jobs and jobs can have multiple candidates)

• Member-Interviews: One-to-Many (Members can have multiple interviews from different companies)

• Interviews-Company: One-to-Many (Each interview is associated with a single Company)

3.10 Wireframe

3.10.1 Admin design

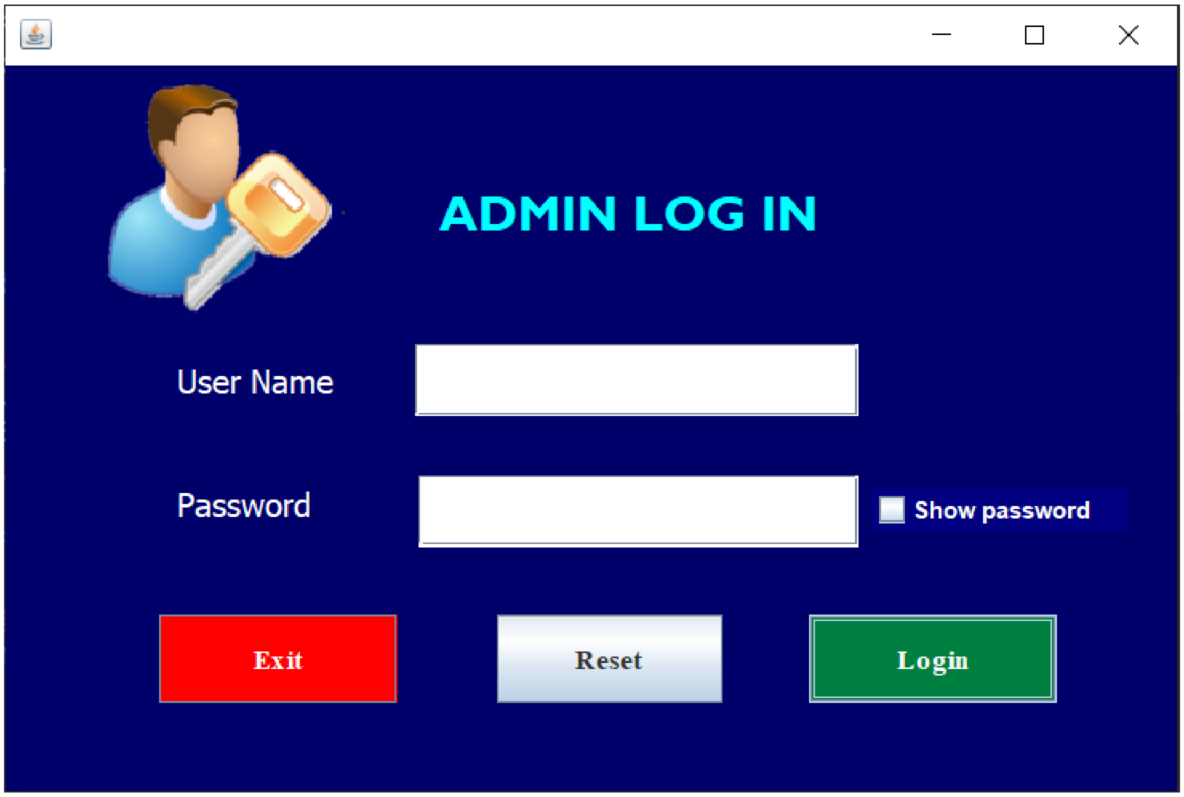


Figure 3.18: Admin login wireframe

This is an admin login to the interface. The admin is already added to the database with a username and password. Then the admin should enter that username and password to log into the system. If it is correct then the admin can log in but if it is incorrect admin has an error message and try again. There is a password showing option then the admin can check whether the entered password is correct or not. Then there is the reset button admin can erase the values and when clicking the exit button admin can exit the system. (see Figure 3.18)

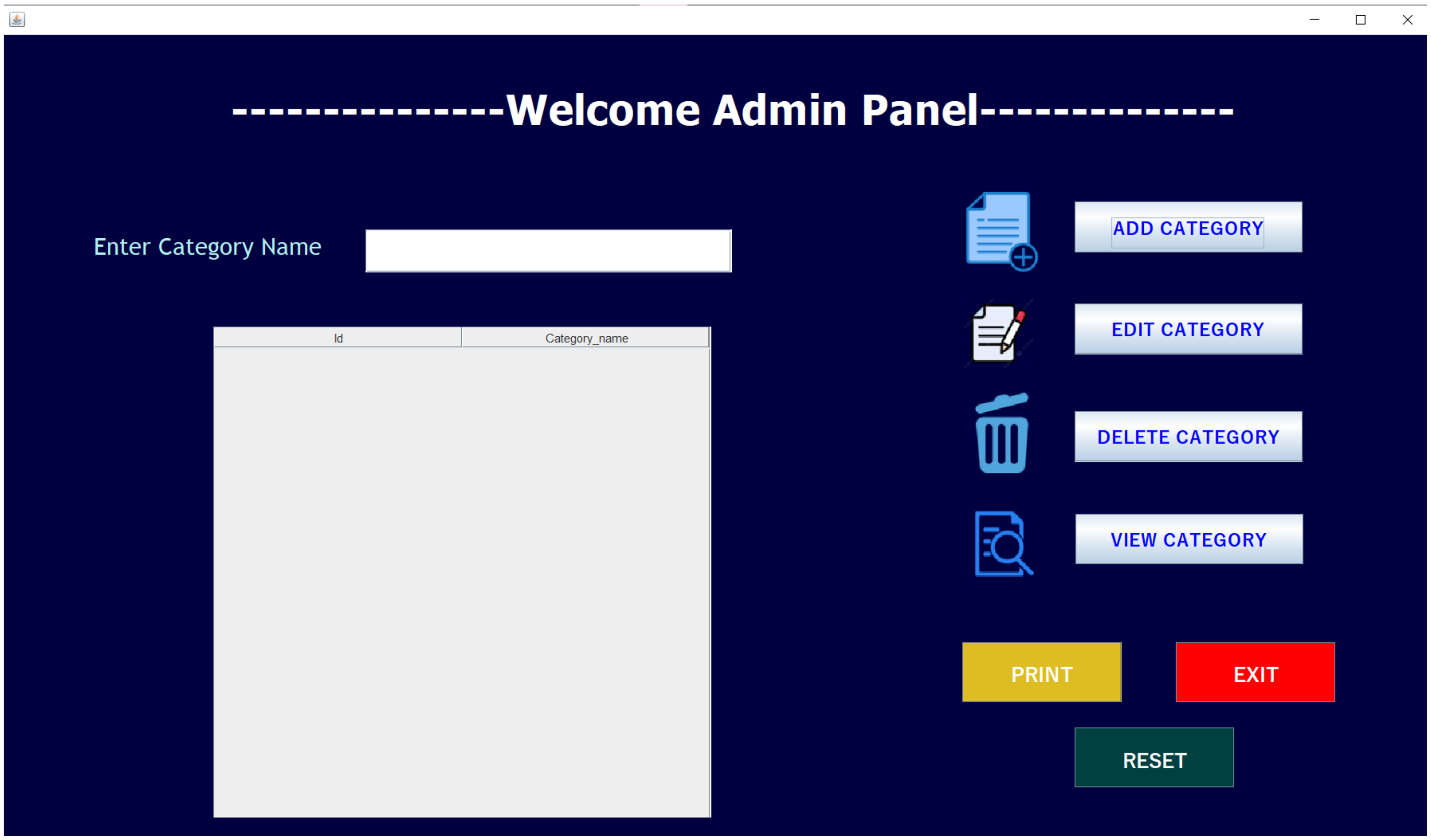


Figure 3.19: Admin panel wireframe

When the Admin login the admin can see this system interface. So there are Add category, update category, edit category, and delete category operations for admin. And if want the hard copy of the added categories there is a print button for that and if the admin wants to exit then the admin can exit. If you want to reset value you can click on reset. (see Figure 3.19)

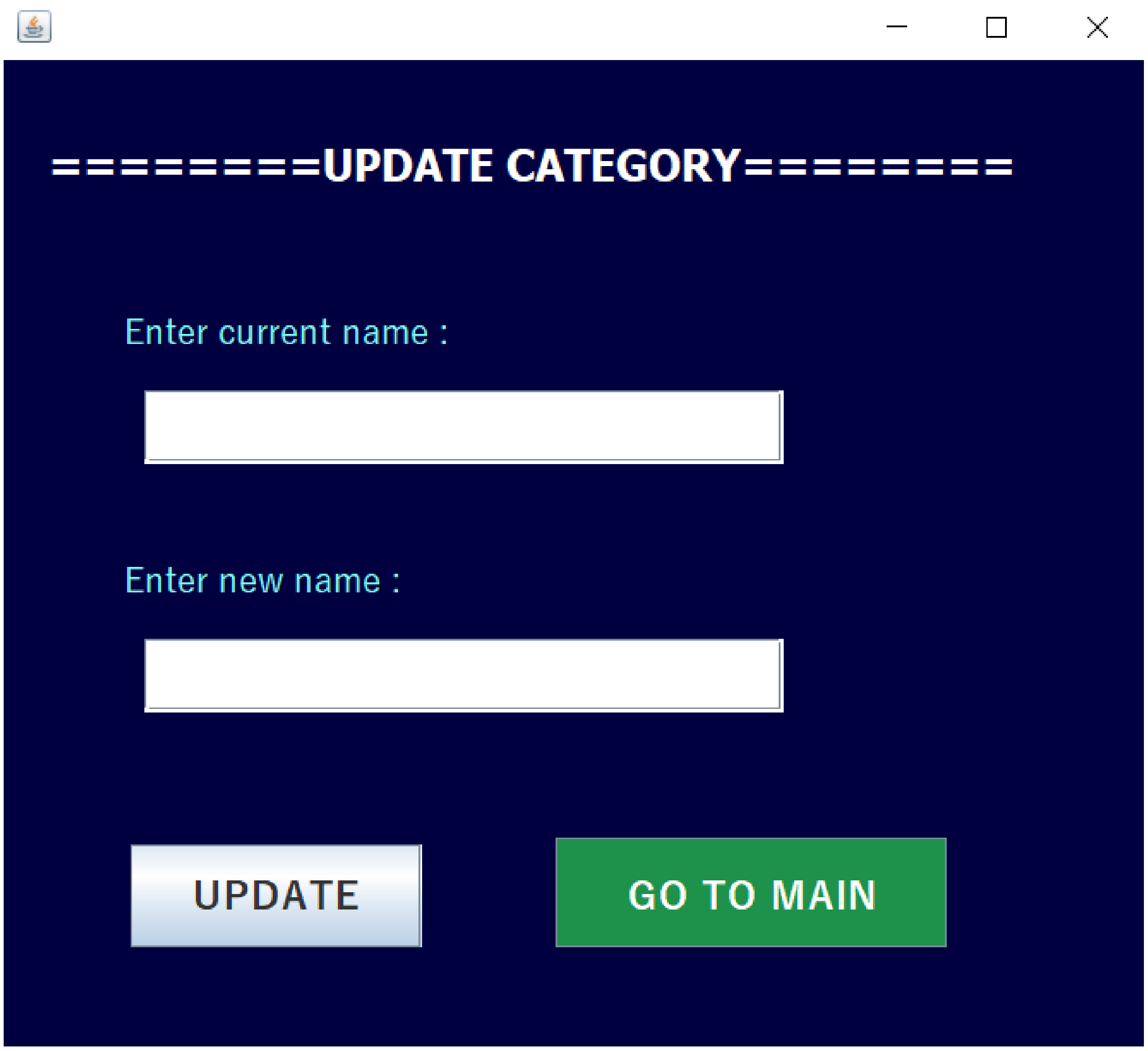
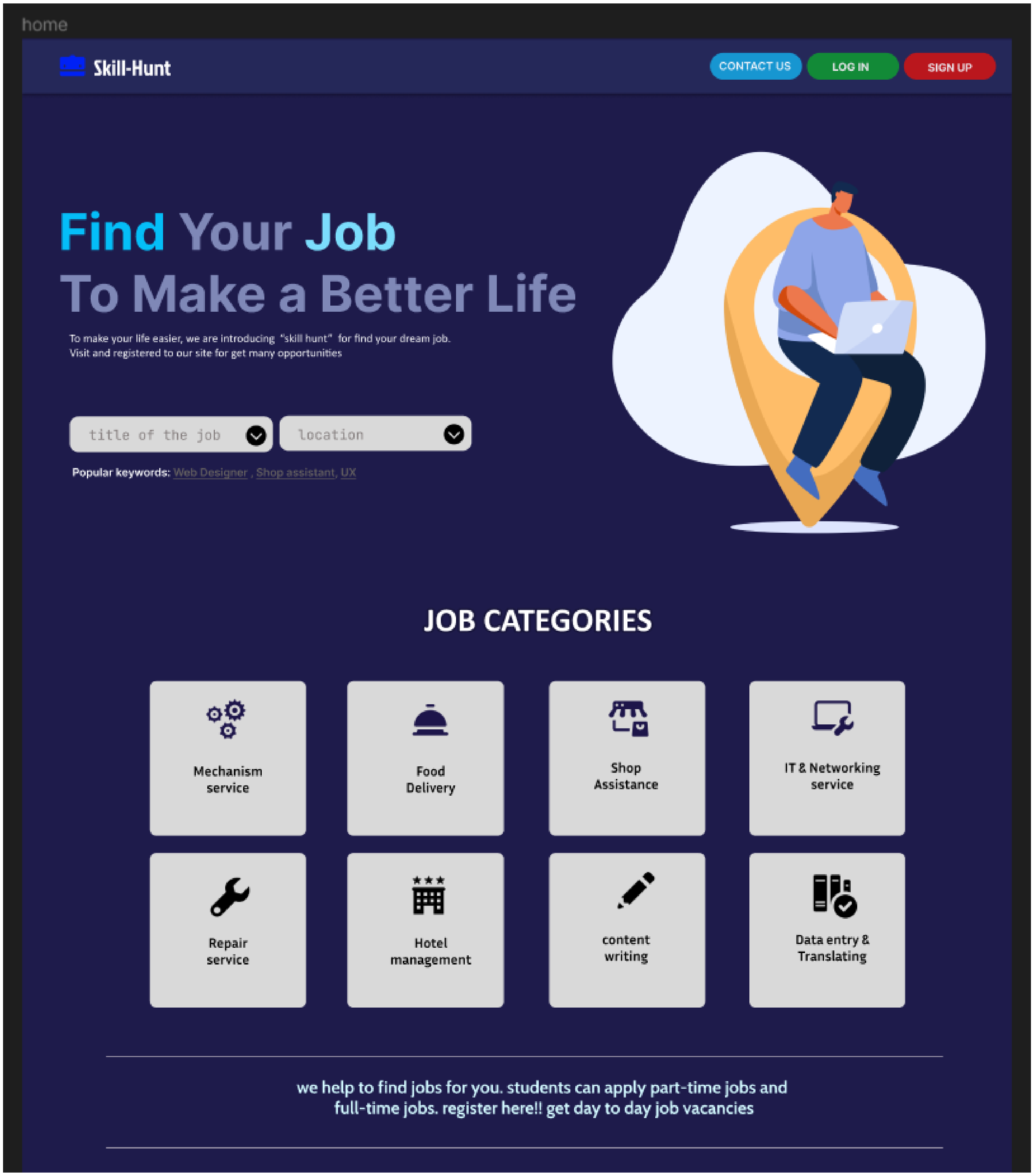


Figure 3.20: update wireframe

If you select the edit category button on the admin panel you can see this interface for updating categories by name. you can enter your current name and after entering a new name then click on update. Once you finish you can back to the admin panel in Figure 3.19.

3.10.2 Website Design

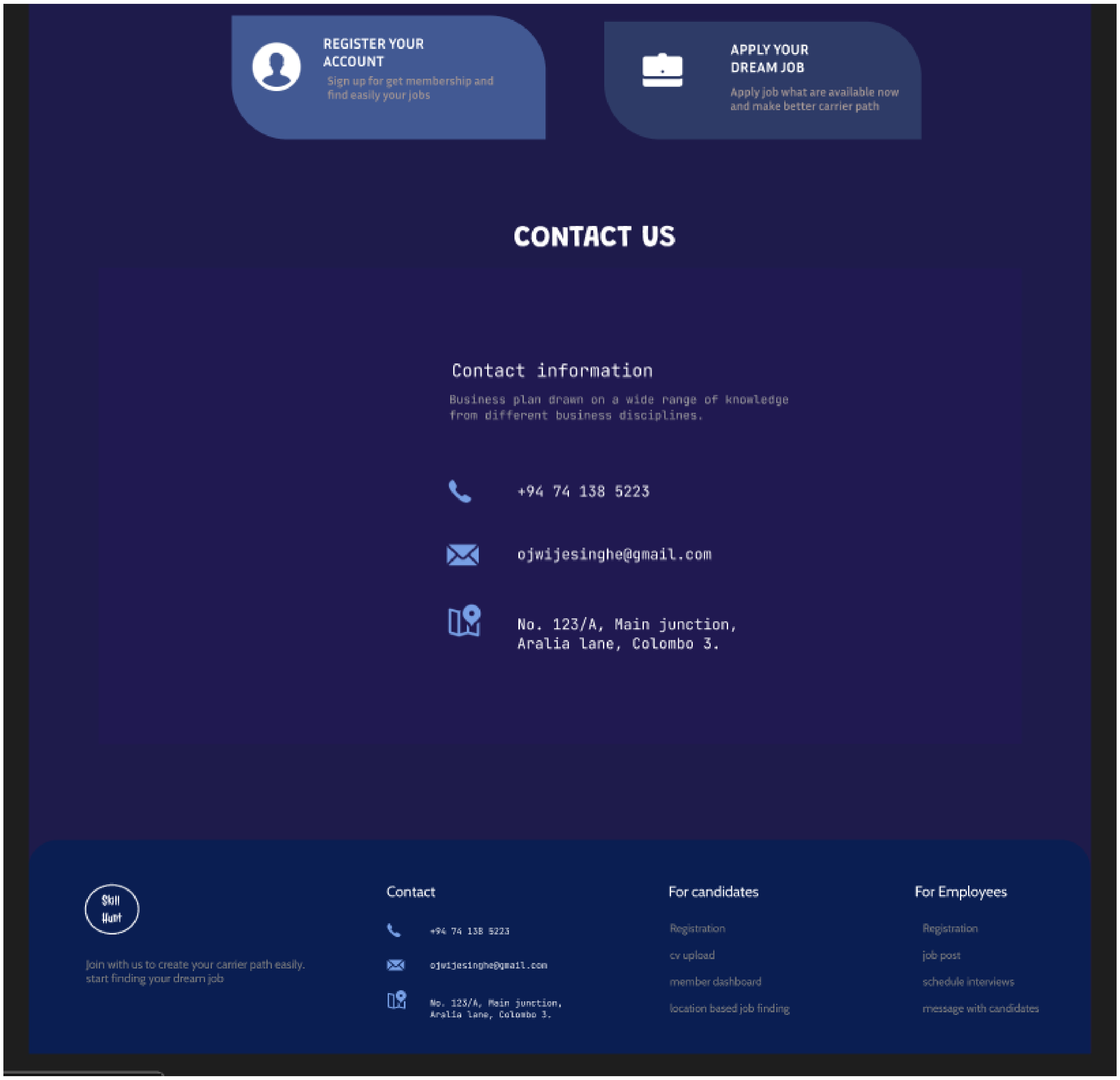
wireframes are visual representations of a web page as the architectural blueprint of your design project. These are created by using Figma.



Find Your Job
To Make a Better Life

To make your life easier, we are introducing "skill hunt" for find your dream job.
Visit and registered to our site for get many opportunities

we help to find jobs for you. students can apply part-time jobs and full-time jobs. register here!! get day to day job vacancies


REGISTER YOUR ACCOUNT

Sign up for get membership and find easily your jobs

APPLY YOUR DREAM JOB

Apply job what are available now and make better carrier path

CONTACT US

Contact information

Business plan drawn on a wide range of knowledge from different business disciplines.

+94 74 138 5223

ojwijesinghe@gmail.com

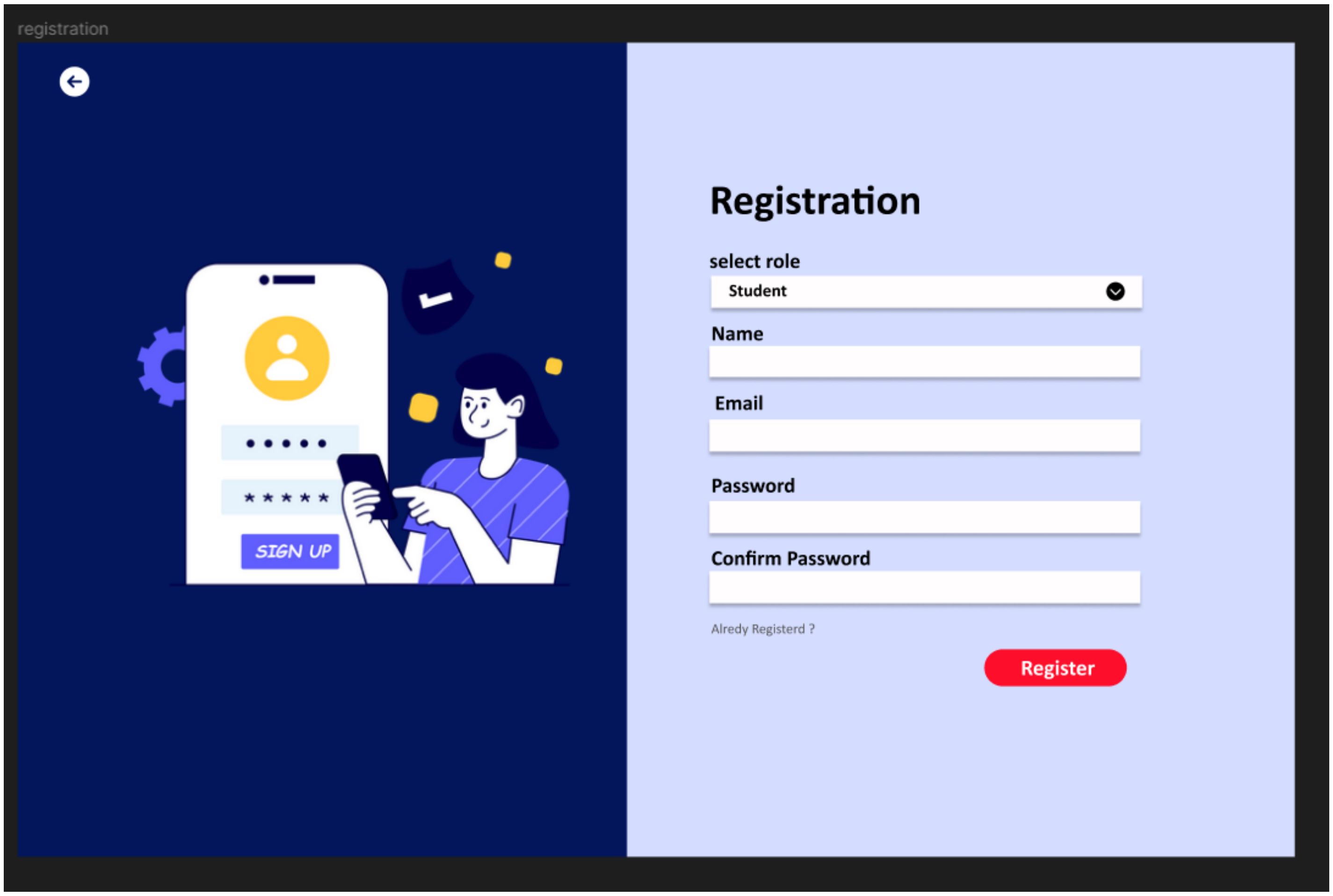
No. 123/A, Main junction,
Aralia lane, Colombo 3.

Join with us to create your carrier path easily. start finding your dream job

Contact	For candidates	For Employees
+94 74 138 5223	Registration	Registration
ojwijesinghe@gmail.com	cv upload	job post
No. 123/A, Main junction, Aralia lane, Colombo 3.	member dashboard	schedule interviews
	location based job finding	message with candidates

Figure 3.21: home page

This picture (see Figure 3.21) is a home page of the skill hunt system. In the navigation bar, you can see contact, login, and registration buttons. There are some job categories. And you can see contact details also. There is a footer including some details.



Registration

select role
Student

Name

Email

Password

Confirm Password

Alredy Registerd ?

Register

Figure 3.22: Registration page

This shows (see Figure 3.22) the registration form of the system. Users can register the system throw this form. First, want to select the correct role because the dashboard will open checking what you choose as the role. After filling in and giving the password, you can click on the registration button. If you already have an account click on already register link.

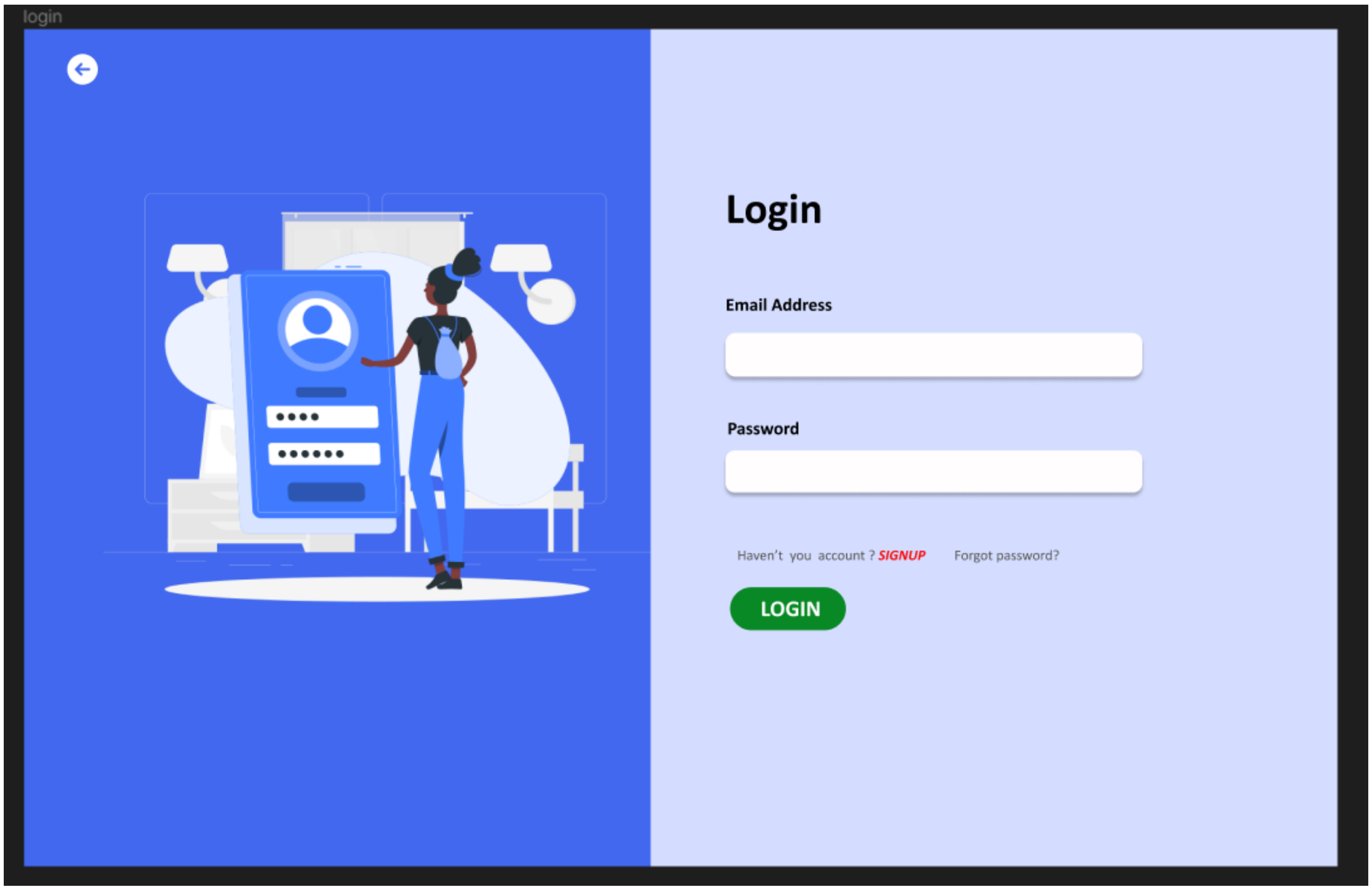
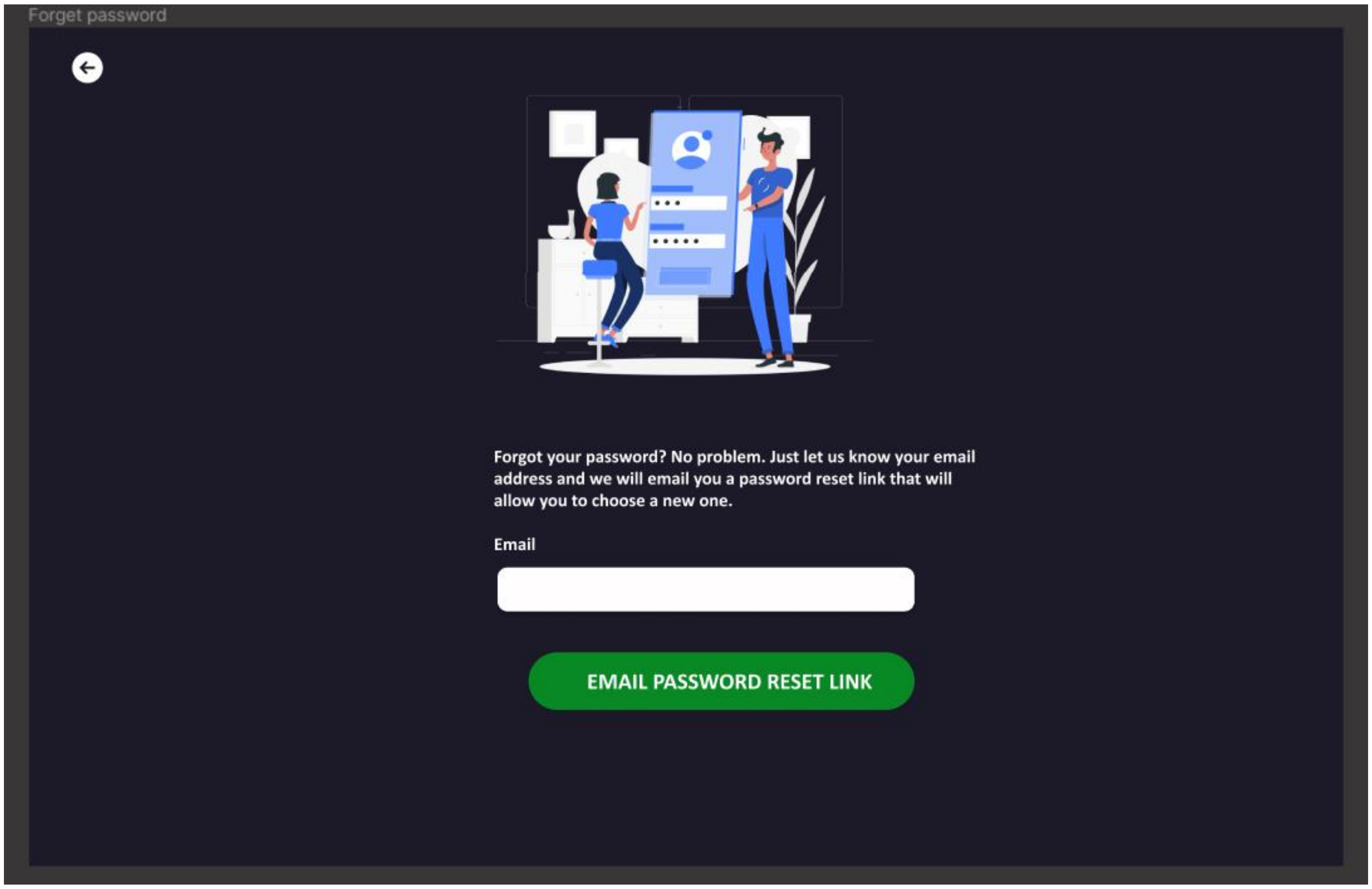


Figure 3.23: Login page

This shows the Login form of the system. (see Figure 3.23) Users can enter their email and password which are given as registration forms. After filling in and giving the password you can click on the Login button. If you already haven'tan an account click on the haven't Account link. If you forget your password then enter forget password link.



Forgot your password? No problem. Just let us know your email address and we will email you a password reset link that will allow you to choose a new one.

Email

Figure 3.24: Forgot password

This picture shows forgot password page. (see Figure 3.24) when you click on forget password on the login page then you can see this page. You want to enter the email of your account then you receive a password change link to the email.

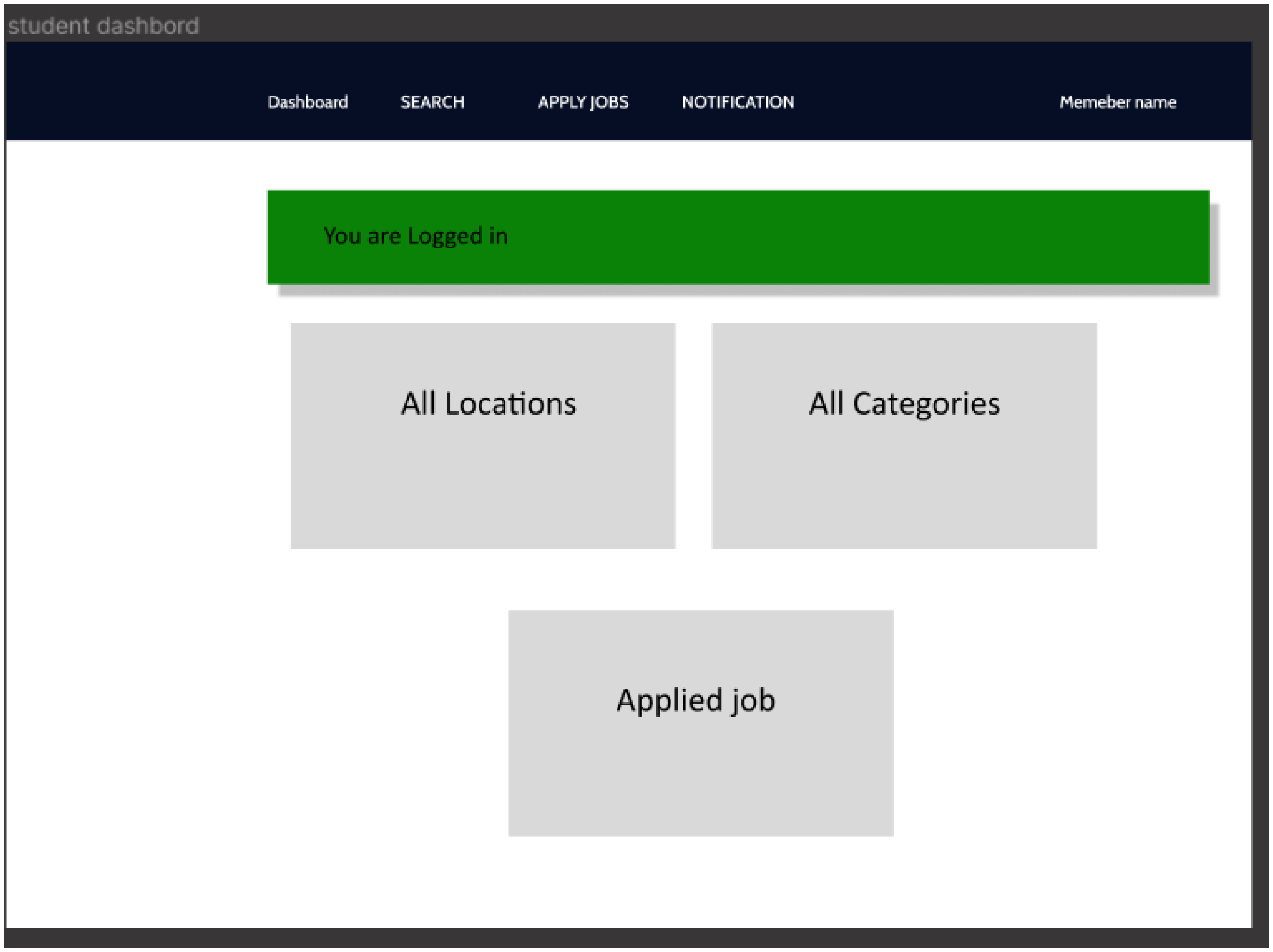
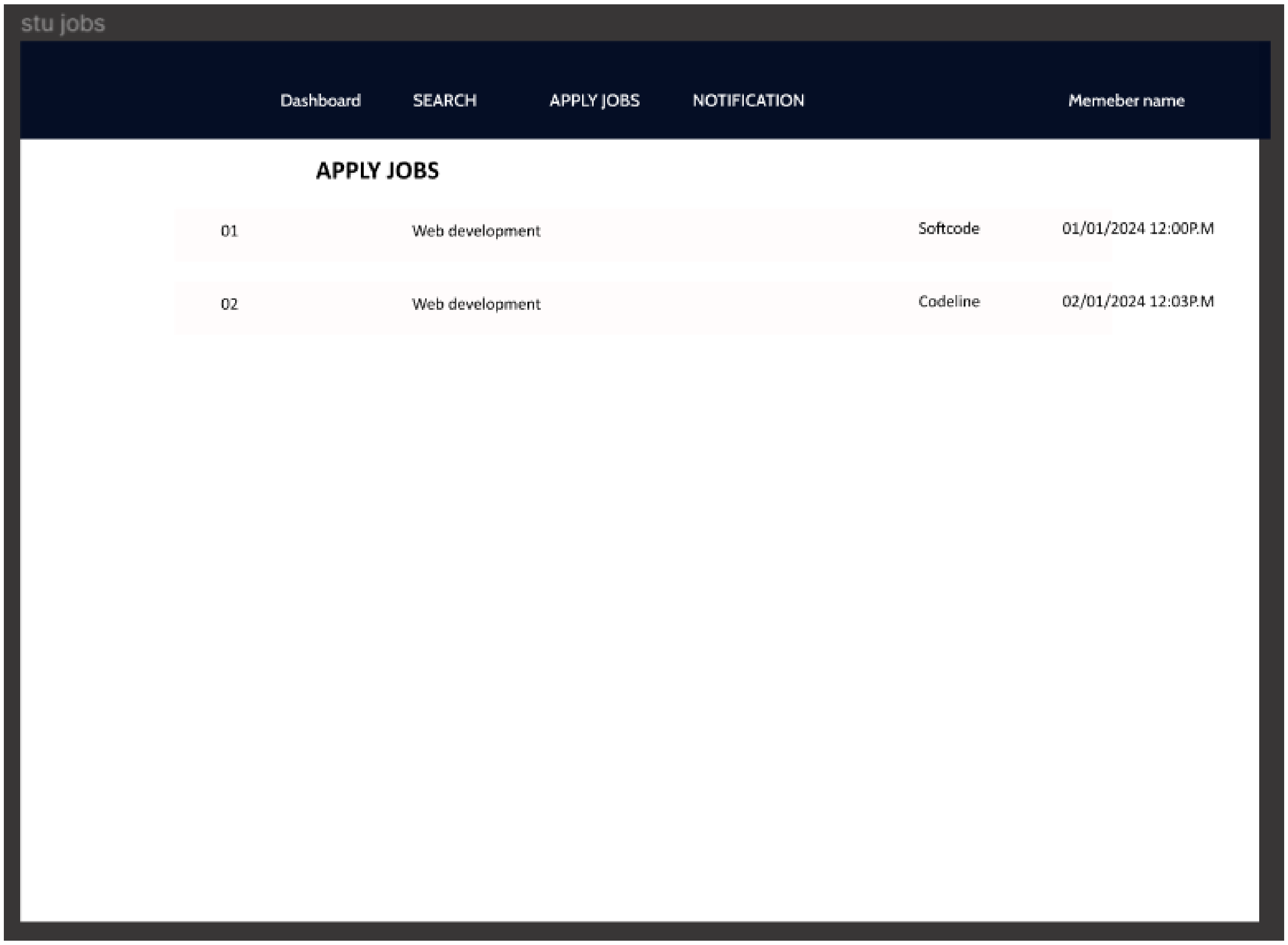


Figure 3.25: Student Dashboard

This shows the student dashboard. (see Figure 3.25) there are search, apply for jobs, and notifications in the navigation bar. Right corner there is a profile name. you can see the logged-in message in the box. The others show the count of all locations, categories, and jobs.

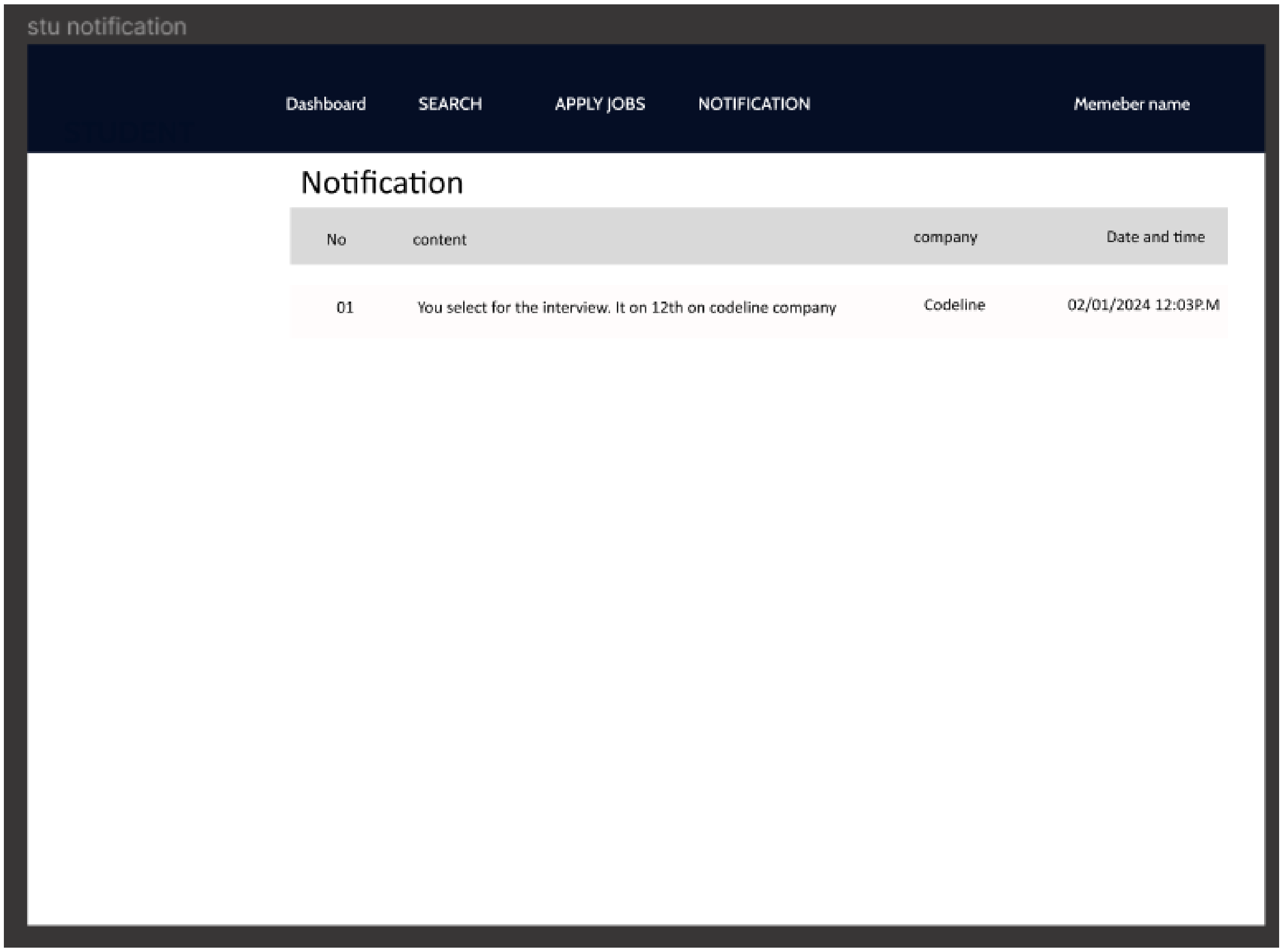


APPLY JOBS

01	Web development	Softcode	01/01/2024 12:00P.M
02	Web development	Codeline	02/01/2024 12:03P.M

Figure 3.26: Applied Jobs page

This picture is a wireframe of a job applied by members. (see Figure 3.26) this shows details about what jobs you applied for. It is easy to look at the jobs you applied for. It shows as a table with the job title and company name.



Notification

No	content	company	Date and time
01	You select for the interview. It on 12th on codeline company	Codeline	02/01/2024 12:03P.M

Figure 3.27: Notification page

This picture shows notifications of students. (See Figure 3.27) When the student applies for the job the company can see who has applied for the job then the company gives an interview date for them. After giving interview dates for the students on this notification page they can see the interview dates and locations.

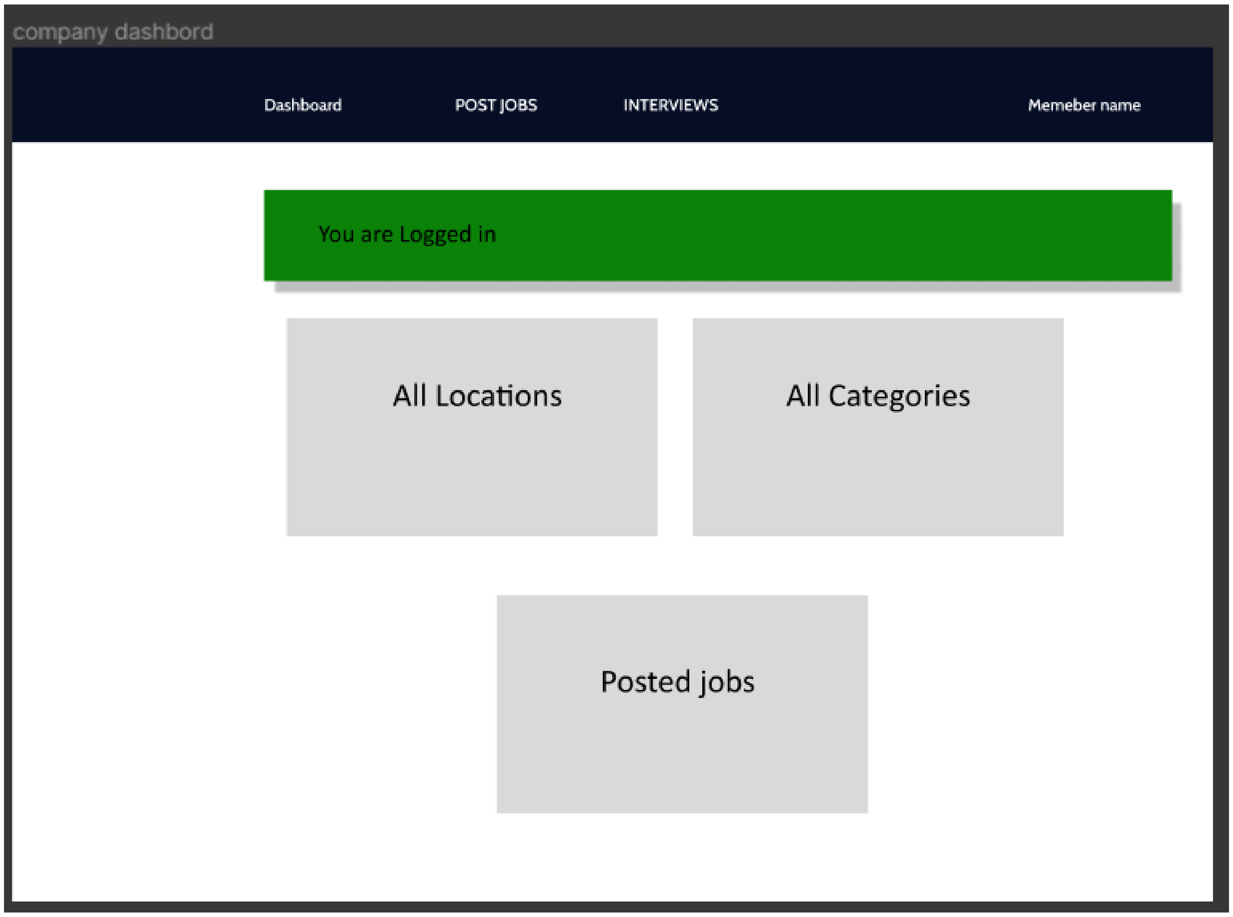


Figure 3.28: Company Dashboard

This shows the student dashboard. (see Figure 3.28) there are post jobs and interviews in the navigation bar. Right corner there is a profile name. you can see the logged-in message in the box. The others show the count of all locations, categories, and posted jobs.

3.11 Summary

In this chapter, we discussed the diagrams related to the system. Use-case diagrams, Activity diagrams, class diagrams, and state diagrams represent parts of the system and it is easy to understand how they interact with the system. We see how the system is designed in Figma and the admin part in Jframe. In the next chapter, we will discuss the implementation of the system. You can get an understanding of how we code the design part.

Chapter 4 : IMPLEMENTATION

4.1 Introduction

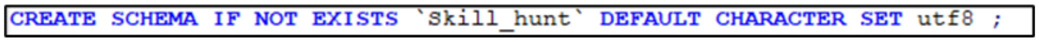
In the previous chapter, we discussed the design part of the system. We get a clear idea about how the system parts will be implemented. Before the implementation first part is to design the system in parts. In this chapter, we will implement the system. You will get an understanding of how the SQL, web, and Java code are implemented.

4.2 DDL

Data Definition Language (DDL) is a subset of SQL. It is a language for describing data and its relationships in a database. It is a group of SQL statements that you can execute to manage database objects, including tables, views, and more.

4.2.1 Listing

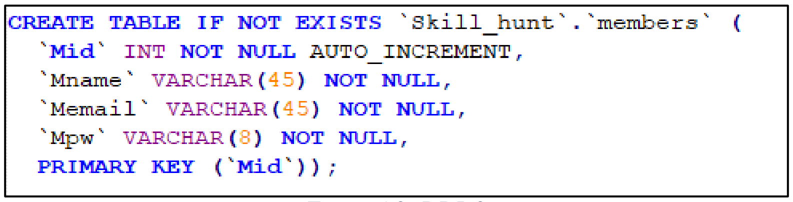
Using DDL statements, you can perform powerful commands in your database such as creating, modifying, and dropping objects.



```
CREATE SCHEMA IF NOT EXISTS `Skill_hunt` DEFAULT CHARACTER SET utf8 ;
```

Figure 4.1: DDL 1

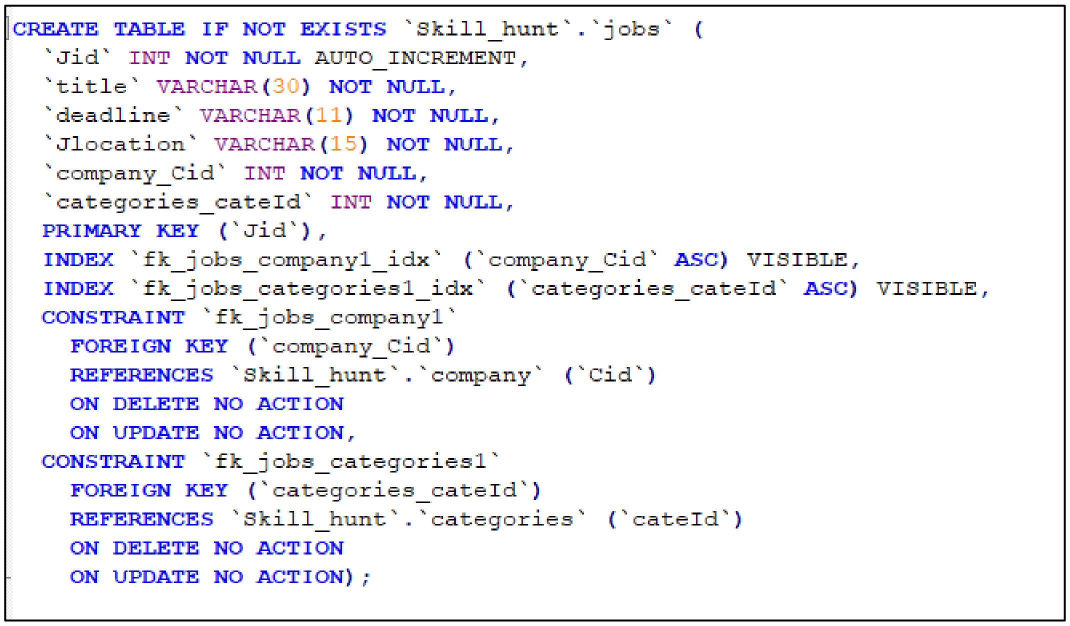
This picture represents creating a database named skill_hunt. It creates if there is no database name like that. And give the default character set for that database. (see Figure 4.1)



```sql
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`members` (
  `Mid` INT NOT NULL AUTO_INCREMENT,
  `Mname` VARCHAR(45) NOT NULL,
  `Memail` VARCHAR(45) NOT NULL,
  `Mpw` VARCHAR(8) NOT NULL,
  PRIMARY KEY (`Mid`));
```

Figure 4.2: DDL 2

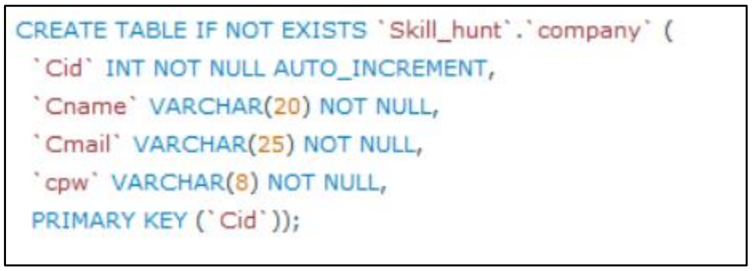
As shown in Figure 4.2 member table has four (4) attributes namely, Mid, Mname, Memail, and Mpw. Mid is the primary key. Mid is a surrogate key (domain-independent, auto-increment candidate key). Mname represents the calling name of the member. Mname, Memail, and Mpw are stored using a Varchar file for every member. Memail is collected as it is using logic.



```sql
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`jobs` (
  `Jid` INT NOT NULL AUTO_INCREMENT,
  `title` VARCHAR(30) NOT NULL,
  `deadline` VARCHAR(11) NOT NULL,
  `Jlocation` VARCHAR(15) NOT NULL,
  `company_Cid` INT NOT NULL,
  `categories_cateId` INT NOT NULL,
  PRIMARY KEY (`Jid`),
  INDEX `fk_jobs_company1_idx` (`company_Cid` ASC) VISIBLE,
  INDEX `fk_jobs_categories1_idx` (`categories_cateId` ASC) VISIBLE,
  CONSTRAINT `fk_jobs_company1`
    FOREIGN KEY (`company_Cid`)
    REFERENCES `Skill_hunt`.`company` (`Cid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_jobs_categories1`
    FOREIGN KEY (`categories_cateId`)
    REFERENCES `Skill_hunt`.`categories` (`cateId`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

Figure 4.3: DDL 3

As shown in the above picture jobs table has six (6) attributes namely, Jid, title, deadline, Jlocation, company_Cid, and categories_cateId. Jid is the primary key. The title represents the name of the job. Each job includes one category. So, this table creates a relation with the categories table using categories_cateId as a foreign key. Jlocation is store the city of company. Title, deadline, and Jlocation are stored using the Varchar field. (see Figure 4.3)



```
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`company` (
  `Cid` INT NOT NULL AUTO_INCREMENT,
  `Cname` VARCHAR(20) NOT NULL,
  `Cmail` VARCHAR(25) NOT NULL,
  `cpw` VARCHAR(8) NOT NULL,
  PRIMARY KEY (`Cid`));
```

Figure 4.4: DDL 4

As shown in Figure 4.4 picture company table has four (4) attributes namely, Cid, Cname, Cmail, and cpw. Cid is the primary key. Cname represents the name of the Company. Cname, Cmail, and Cpw are stored using the Varchar field and they get login details.

4.3 SQL Queries

4.3.1 DML

Data Manipulation Language or DML is a subset of operations used to insert, delete, and update data in a database. A DML is often a sublanguage of a more extensive language like SQL

4.3.1.1 Database Test Plan

Project Name	Skill Hunt
Test Plan ID	ID-01
Brief Introduction about the system	This software revolves around connecting students and undergraduates with job opportunities. The system specifically focuses on part-time and internship jobs, catering to the needs of undergraduates and students seeking to gain work experience and build their careers. Skill Hunt uses location-based features to help students find jobs. This system keeps members informed about job interview scheduling. Users and companies can register for this system and companies can post their jobs. There are job categories. The companies can post their jobs in a relevant job category.
Test Objectives	Select, View
Features to be tested	Search Job Count jobs posted by each company Select login details Select Interview details View applied for jobs

Test Environment	Devices – Laptop Software – workbench Database – workbench database
Test Approach	Black Box
Testing Tasks	Test Planning, Test Designing, Test Development, Test Execution, Test Evolution
Test Deliverables	Test Plan, Test Environment, Test Summary, Test Result, Test Evolution Report
Schedule	Date: 25/02/2024 Time: 7.09 P.M

Table 4.1: Database Test Plan

A Test Plan outlines the approach, scope, objectives, resources, and schedule for testing a specific project or product. A Test Strategy defines the high-level approach to testing for an organization or a project, guiding the overall testing process. Now we will see test cases for this test plan. (see Table 4.1)

4.3.1.2 Test Cases

Test Case	
Test Unit: Search	Tester: Oneli Jayodya
Test Case ID: 01	Test Type: Black Box
Test Description: Search Job in the category	Execution Date: 25/02/2024
Title: Search	Test Execution Time: 06.30 p.m.

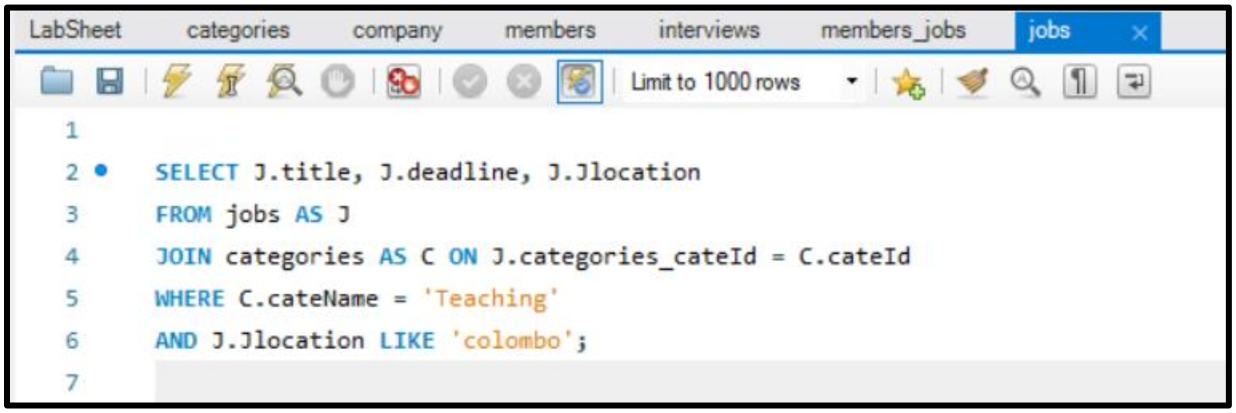
Table 4.2: search test case

This test case shows when a user searches for jobs by category and then lists each of them. Here is the result of this test case. (see Table 4.3)

Step no	Test step	Test case ID	Test Input	Expect Result	Actual Result	Test Result
01	Search	01	Search the Teaching category with the location	Show Teaching Job post with deadline and location.	Show Teaching Job post with deadline and location	Pass

Table 4.3: Search test result

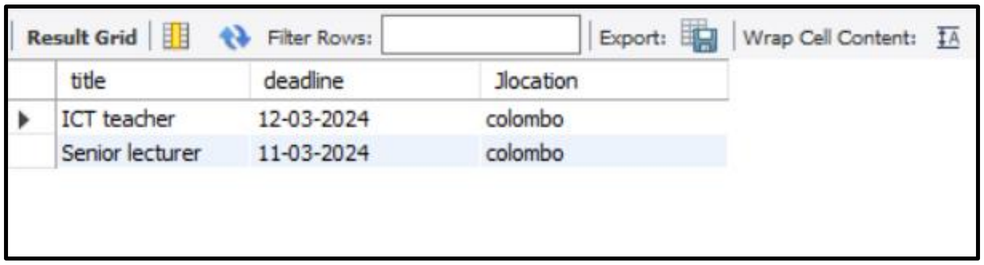
This is the test result of the test case. You can see it tested when the user searches searching category jobs and then shows only the teaching category jobs testing is pass. (see Table 4.3)



```sql

SELECT J.title, J.deadline, J.Jlocation
FROM jobs AS J
JOIN categories AS C ON J.categories_cateId = C.cateId
WHERE C.cateName = 'Teaching'
AND J.Jlocation LIKE 'colombo';

```

Figure 4.5: Search Query 1

This code shows how to find the teaching category jobs. First, select the column name to show the user. After using JOIN to get data from two tables. The code goes like about. (see Figure 4.5: Search Query 1



title	deadline	Jlocation
ICT teacher	12-03-2024	colombo
Senior lecturer	11-03-2024	colombo

Figure 4.6: Search Query 1 result

This shows the answer for the above code. (see Figure 4.6) you can see the title related to the teaching category with location.

Test Case	
Test Unit: Count	Tester: Oneli Jayodya
Test Case ID: 02	Test Type: Black Box
Test Description: Count how many jobs a posted by each company	Execution Date: 25/02/2024
Title: Count jobs	Test Execution Time: 07.00 p.m.

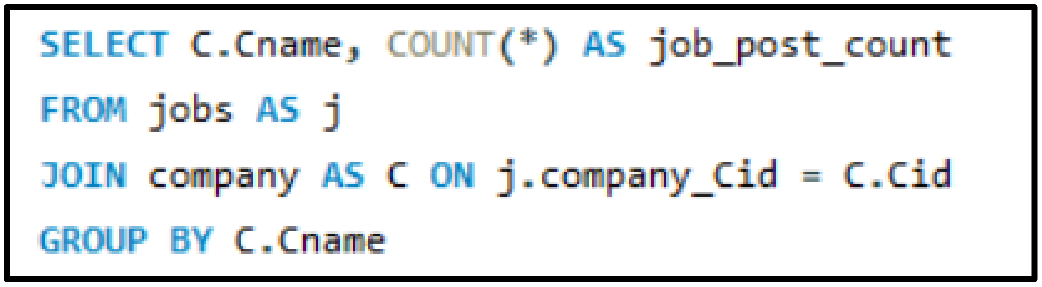
Table 4.4: Count test case

This test case shows when a company views jobs and what are they posted. then list down each of them. Here is the result of this test case. (see Table 4.4)

Step no	Test step	Test case ID	Test Input	Expect Result	Actual Result	Test Result
01	Count jobs	02	Count how many jobs posted by each company	Show the company name with a count of job posts.	Show company name with count of job posts	Pass

Table 4.5: Count test result

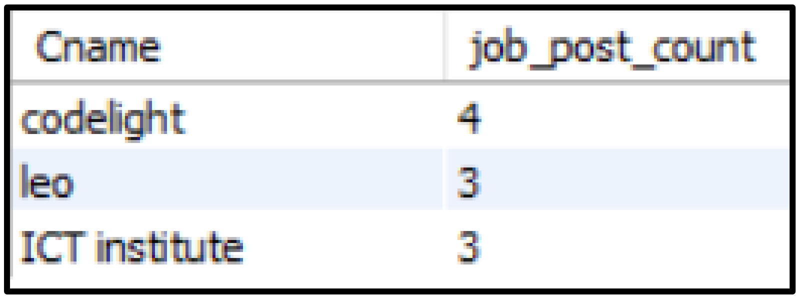
This is the test result of the test case Table 4.4. You can see it tested when the company after posting a job shows each company posted jobs. this testing is pass. (see Table 4.5)



```
SELECT C.Cname, COUNT(*) AS job_post_count
FROM jobs AS j
JOIN company AS C ON j.company_Cid = C.Cid
GROUP BY C.Cname
```

Figure 4.7: Count Query 2

This code shows how to get a count of post jobs of each company. First, select the column name and count function. After using JOIN to get data from two tables. And group by them on the company name. The code goes like this above. (see Figure 4.7)



Cname	job_post_count
codelight	4
leo	3
ICT institute	3

Figure 4.8: Count Query 2 result

This shows the answer for the above code. (see Figure 4.8) you can see the count of posted jobs in each company.

Test Case	
Test Unit: Select	Tester: Oneli Jayodya
Test Case ID: 03	Test Type: Black Box
Test Description: Select login details	Execution Date: 25/02/2024
Title: Select logs	Test Execution Time: 07.40 p.m.

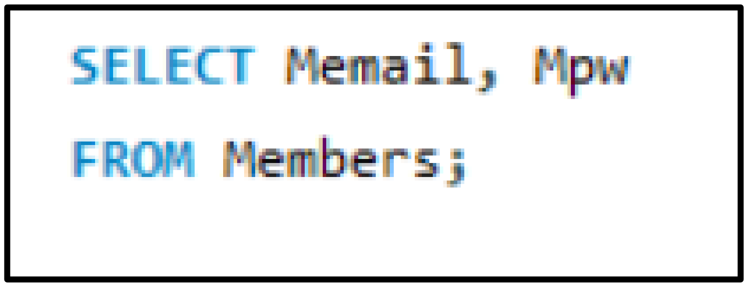
Table 4.6: Select Logs test case

This test case shows when member log in to the system check their email and password. Here is the result of this test case. (see Table 4.7)

Step no	Test step	Test case ID	Test Input	Expect Result	Actual Result	Test Result
01	Select details	03	Select Member email and password	Show member email with password for login	Show member email with a password	Pass

Table 4.7: Select Log test result

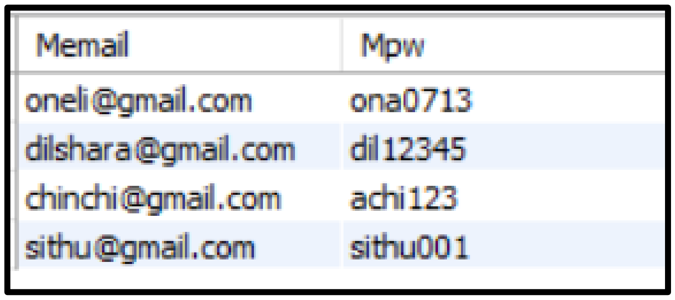
This is the test result of the test case (see Table 4.6) You can see when the member login to the system check the email and password from the member table. this testing is pass. (see Table 4.7)



```sql
SELECT Memail, Mpw
FROM Members;
```

Figure 4.9: Select query 7

This code shows how to get the member email and password from the member table for login. The code goes like this above. (see Figure 4.10)



Memail	Mpw
oneli@gmail.com	ona0713
dilshara@gmail.com	dil12345
chinchi@gmail.com	achi123
sithu@gmail.com	sithu001

Figure 4.10: Select Query 7 result

This shows the answer for the above code. (see Figure 4.9: Select query 7) you can see the email and password getting only from the table.

Test Case	
Test Unit: Select	Tester: Oneli Jayodya
Test Case ID: 04	Test Type: Black Box
Test Description: Select Interview details	Execution Date: 25/02/2024
Title: Select Interview	Test Execution Time: 08.00 p.m.

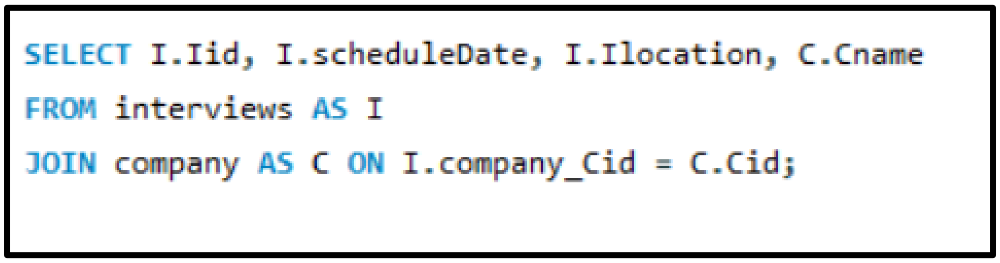
Table 4.8: Select Interview test case

This test case shows when a member gets an interview from the company. Here is the result of this test case. (see Table 4.7)

Step no	Test step	Test case ID	Test Input	Expect Result	Actual Result	Test Result
01	Select Interview details	04	Select interview details with the company name	Show interview details with the company name	Show interview details with the company name	Pass

Table 4.9: Select the Interview test result

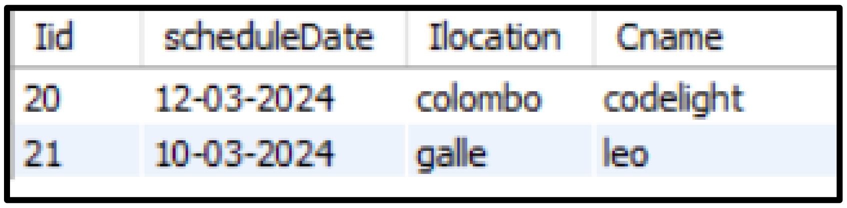
This is the test result of the test case (see Table 4.8) You can see when the member goes to the notification and then show the interviews which are given by the company. this testing is pass. (see Table 4.7)



```sql
SELECT I.Iid, I.scheduleDate, I.Ilocation, C.Cname
FROM interviews AS I
JOIN company AS C ON I.company_Cid = C.Cid;
```

Figure 4.11: Select query 8

This code shows how to get the Interview details from the Interview table with the company name. The code goes like this above. (see Figure 4.12)



Iid	scheduleDate	Ilocation	Cname
20	12-03-2024	colombo	codelight
21	10-03-2024	galle	leo

Figure 4.12: Select Query 8 result

This shows the answer for the above code. (see Figure 4.11) you can see the interview date from each company.

Test Case	
Test Unit: View	Tester: Oneli Jayodya
Test Case ID: 05	Test Type: Black Box
Test Description: View applied for jobs	Execution Date: 25/02/2024
Title: View applied jobs	Test Execution Time: 08.20 p.m.

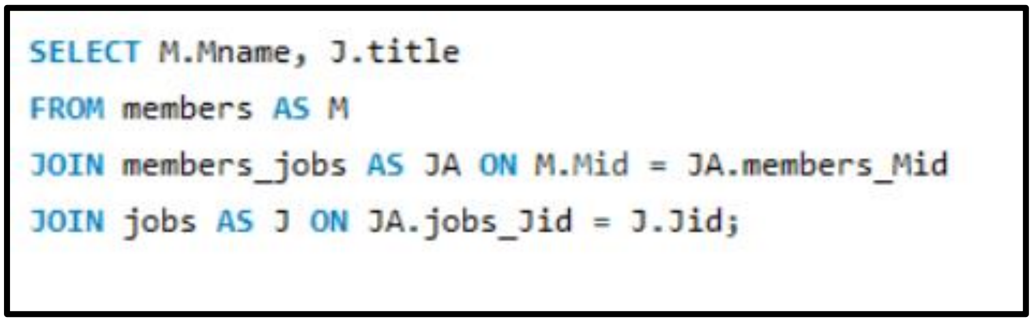
Table 4.10: View Apply Job test case

When a member applies for a job then this case shows what each member applies jobs. Here is the result of this test case. (see Table 4.11)

Step no	Test step	Test case ID	Test Input	Expect Result	Actual Result	Test Result
01	View applied jobs	05	Select the member name from the member table and the job title from the jobs table	Show member's name with what are the jobs they applied	Show member's name with what are the jobs they applied	Pass

Table 4.11: View Apply Job test result

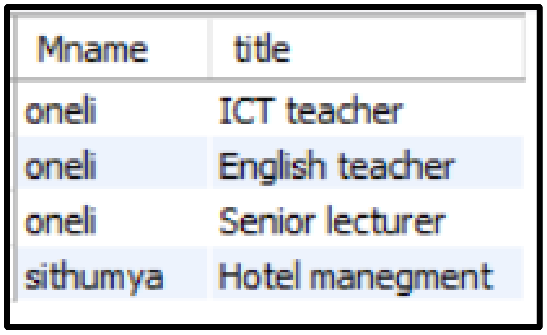
This is the test result of the test case (see Table 4.10)You can see each member's applied job title. this testing is pass. (see Table 4.11)



```sql
SELECT M.Mname, J.title
FROM members AS M
JOIN members_jobs AS JA ON M.Mid = JA.members_Mid
JOIN jobs AS J ON JA.jobs_Jid = J.Jid;
```

Figure 4.13: View query 9

This code shows what are the jobs applied by each member. The code goes like this above. (see Figure 4.14)



Mname	title
oneli	ICT teacher
oneli	English teacher
oneli	Senior lecturer
sithumya	Hotel manegment

Figure 4.14: View Query 9 result

This shows the answer for the above code. (see Figure 4.13) you can see the member's name with what are the jobs they applied for.

4.3.1.3 Java Test Plan

Project Name	Skill Hunt
Test Plan ID	ID-02
Brief Introduction about the system	This software revolves around connecting students and undergraduates with job opportunities. The system specifically focuses on part-time and internship jobs, catering to the needs of undergraduates and students seeking to gain work experience and build their careers. Skill Hunt uses location-based features to help students find jobs. This system keeps members informed about job interview scheduling. Users and companies can register for this system and companies can post their jobs. There are job categories. The companies can post their jobs in a relevant job category.
Test Objectives	Update, Delete, Insert, View
Features to be tested	Insert Category name View Category name Update Category name Delete Category name
Test Environment	Devices – Laptop Software – Eclipse

Test Approach	Black Box
Testing Tasks	Test Planning, Test Designing, Test Development, Test Execution, Test Evolution
Test Deliverables	Test Plan, Test Environment, Test Summary, Test Result, Test Evolution Report
Schedule	Date: 24/02/2024 Time: 11.00 A.M

Table 4.12: Java Test Plan

A Test Plan outlines the approach, scope, objectives, resources, and schedule for testing a specific project or product. A Test Strategy defines the high-level approach to testing for an organization or a project, guiding the overall testing process. Our Test Objectives are Insert, Delete, Update and view. Now we will see test cases for this test plan. (See Table 4.12)

4.3.1.4 Test Cases

Test Case	
Test Unit: Login	**Tester**: Oneli jayodya
Test Case ID: 01	**Test Type**: Black Box
Test Description: Enter the Dashboard	**Test Execution Date**:24/02/2024
Title: Admin Login	**Test Execution Time**: 11.30 a.m.

Table 4.13: Login test case

Step no.	Test step	Test case ID	Test Input	Expect Result	Actual Result	Test Result
01.	Login to the system	01	Username: admin Password: a23ewd	Not access to Login dashboard	Invalid username or password	Pass
02.	Login to the system	01	Username: oneli Password: a2024	Login to dashboard	Login Successful	pass

Table 4.14: Login test result

This is the test result of the test case Table 4.13 You can see the admin login to the system enters different username with password showing errors to the wrong testing and go to dashboard when enter correct. this testing is pass (see Table 4.14)

Test Case	
Test Unit: Add category	**Tester**: Oneli jayodya
Test Case ID: 02	**Test Type**: Black Box
Test Description: Add category to system	**Test Execution Date**:24/0822024
Title: Add category	**Test Execution Time**: 11.35 a.m.

Table 4.15: Add category test case

Step no.	Test step	Test case ID	Test Input	Expect Result	Actual Result	Test Result
01.	Add category to the system	02	Enter category name: Web design	Already in the database	Category already exists!	Pass
02.	Add category to the system	02	Enter category name:	Enter category	Failed to insert category. Please try again.	pass
03.	Add category to the system	02	Enter category name: shop manager	Add to database	The category was inserted successfully!	Pass

Table 4.16: Add category test result

This is the test result of the test case Table 4.15 You can see the category adding tested. It checks already added or fill data. Correct enters store in the database. (see Table 4.16)

Test Case	
Test Unit: Edit category	**Tester**: Oneli jayodya
Test Case ID: 03	**Test Type**: Black Box
Test Description: edit category to system	**Test Execution Date**:24/0822024
Title: Edit category	**Test Execution Time**: 11.50 a.m.

Table 4.17: Edit category test case

Step no.	Test step	Test case ID	Test Input	Expect Result	Actual Result	Test Result
01.	Edit category to the system	03	Enter current name: Enter new name:	Enter details	Please enter values!	Pass
02.	Edit category to the system	03	Enter current name: web design Enter new name: Web development	Edit Category	Category Updated successfully !!	pass

Table 4.18: Edit category test result

This is the test result of the test case Table 4.17 You can see the category update tested. It checks fill data or not. Correctly re-enters store in the database. (see Table 4.18)

Test Case	
Test Unit: Delete category	**Tester**: Oneli jayodya
Test Case ID: 04	**Test Type**: Black Box
Test Description: Delete category to system	**Test Execution Date**:24/0822024
Title: Delete category	**Test Execution Time**: 12.00 a.m.

Table 4.19: Delete test case

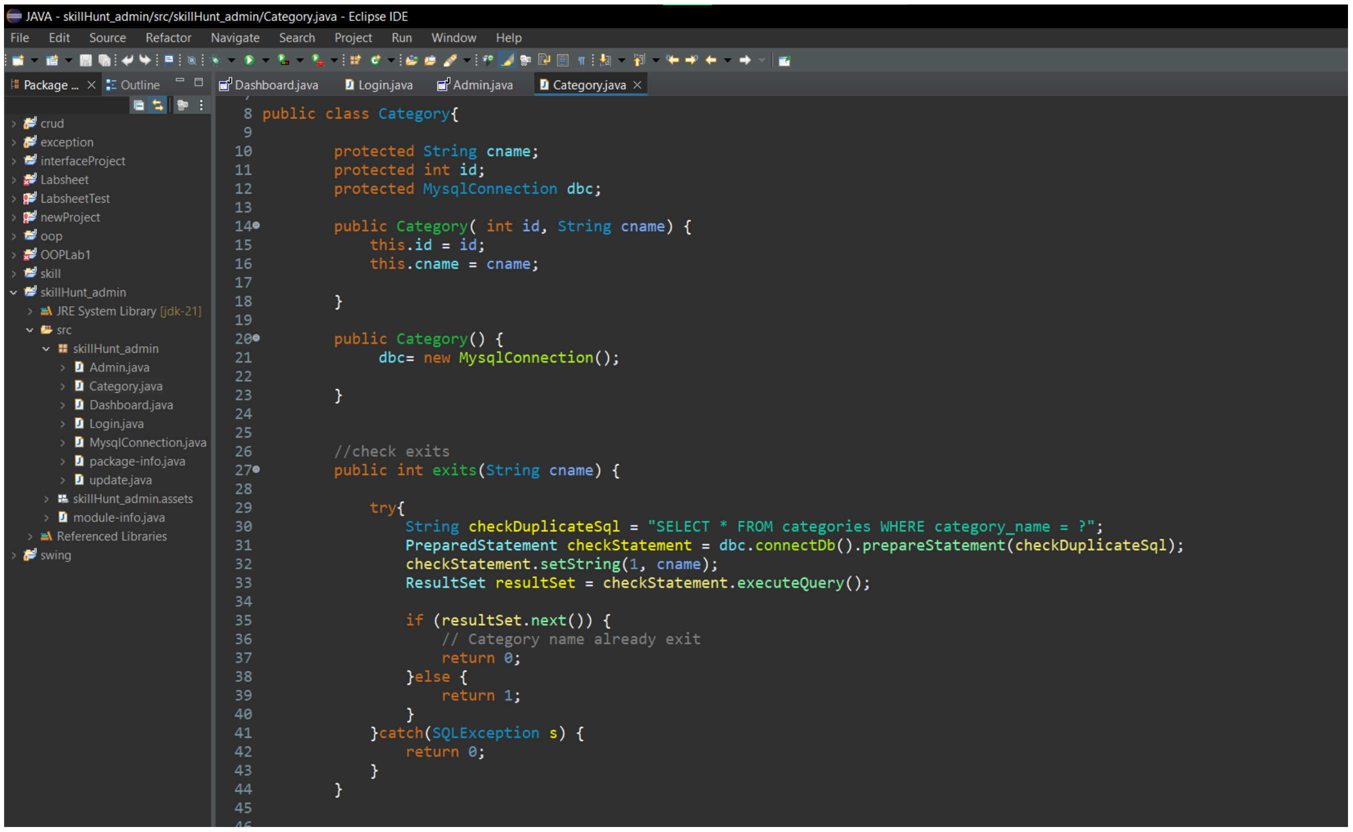
Step no.	Test step	Test case ID	Test Input	Expect Result	Actual Result	Test Result
01.	Delete the category from the system	03	Not selected a row	Select row	Please select a row to delete	Pass
02.	Delete the category from the system	03	The web design row selected	Delete Category	The category deleted successfully from the database	pass

Table 4.20: Delete test result

This is the test result of the test case Table 4.19 You can see the category delete tested. It checks select row or not. Correctly selected rows are deleted with confirmation message. (see Table 4.20)

4.4 Data validation & error handlings

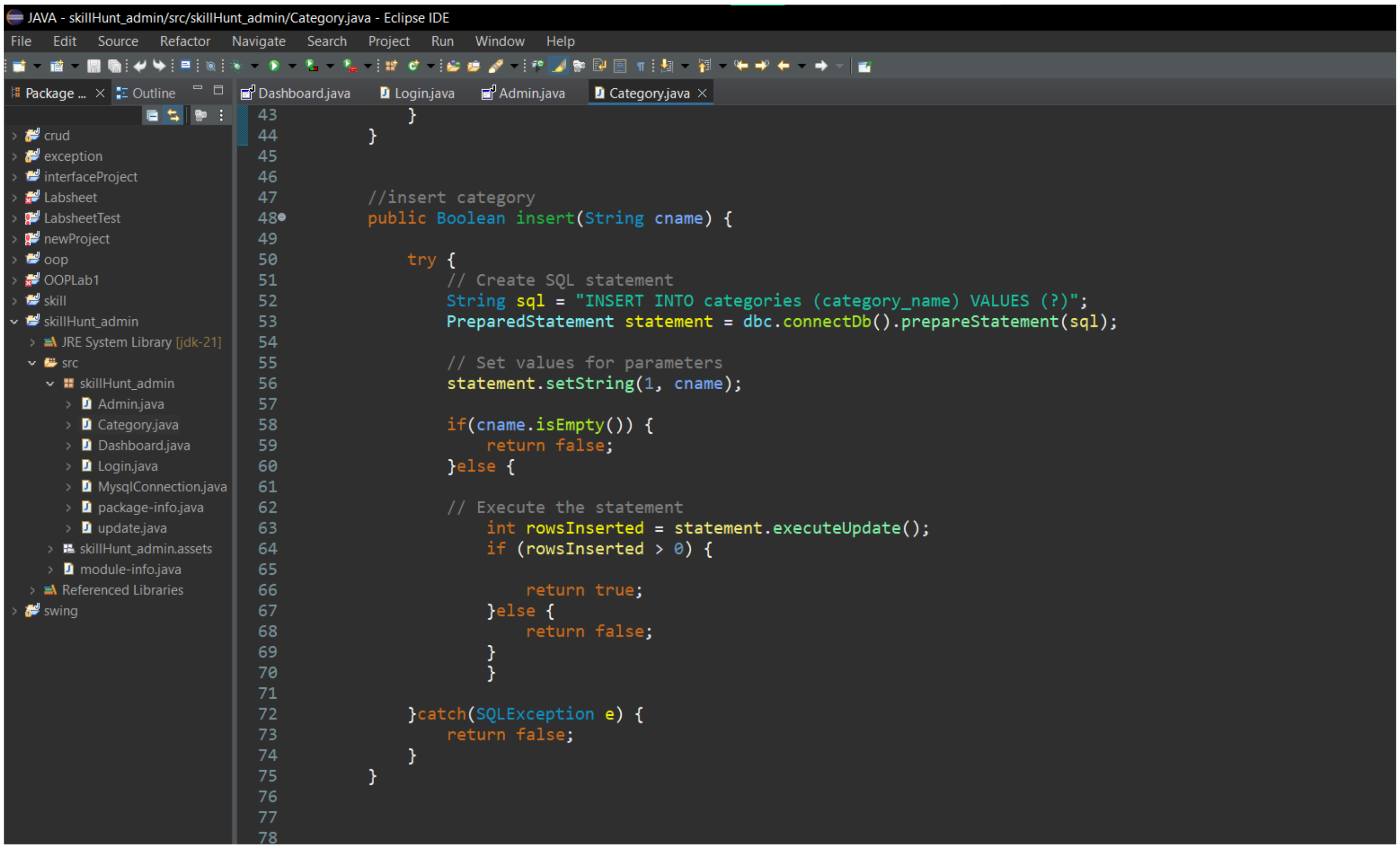
Data validation is the process of verifying and validating data that is collected before it is used. Any type of data handling task, whether it is gathering data, analyzing it, or structuring it for presentation, must include data validation to ensure accurate results.



```java
public class Category{

        protected String cname;
        protected int id;
        protected MysqlConnection dbc;

        public Category( int id, String cname) {
            this.id = id;
            this.cname = cname;

        }

        public Category() {
             dbc= new MysqlConnection();

        }


        //check exits
        public int exits(String cname) {

            try{
                String checkDuplicateSql = "SELECT * FROM categories WHERE category_name = ?";
                PreparedStatement checkStatement = dbc.connectDb().prepareStatement(checkDuplicateSql);
                checkStatement.setString(1, cname);
                ResultSet resultSet = checkStatement.executeQuery();

                if (resultSet.next()) {
                    // Category name already exit
                    return 0;
                }else {
                    return 1;
                }
            }catch(SQLException s) {
                return 0;
            }
        }
```

Figure 4.15: check already exits

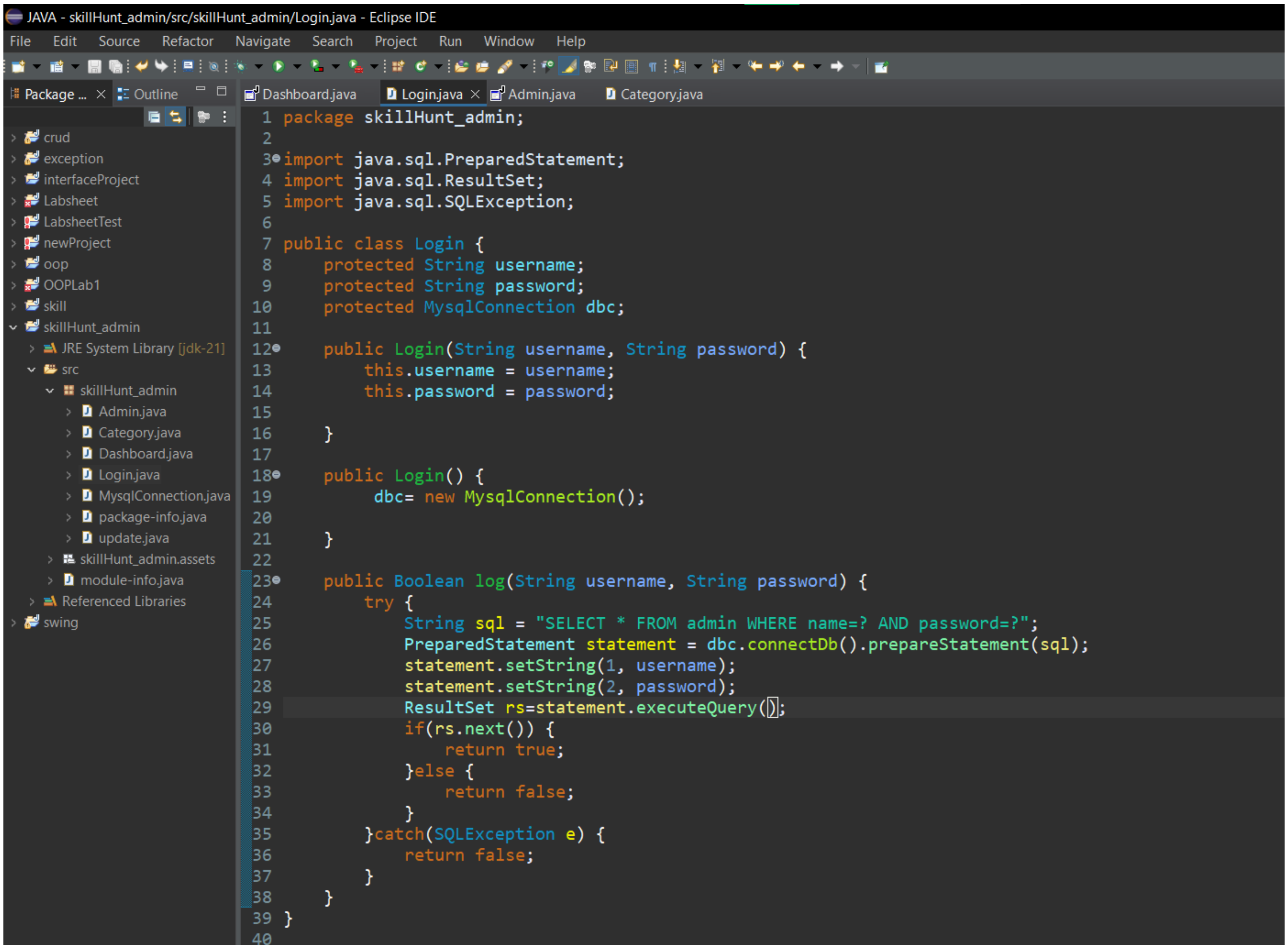
This is to check if the admin added the category to the system and whether it is already in the database then show the message it already exists if it is not then add the category and show added successfully. (see Figure 4.15)



```java
            }
        }


        //insert category
        public Boolean insert(String cname) {

            try {
                // Create SQL statement
                String sql = "INSERT INTO categories (category_name) VALUES (?)";
                PreparedStatement statement = dbc.connectDb().prepareStatement(sql);

                // Set values for parameters
                statement.setString(1, cname);

                if(cname.isEmpty()) {
                    return false;
                }else {

                // Execute the statement
                    int rowsInserted = statement.executeUpdate();
                    if (rowsInserted > 0) {

                        return true;
                    }else {
                        return false;
                    }
                    }

            }catch(SQLException e) {
                return false;
            }
        }
```

Figure 4.16: Check empty

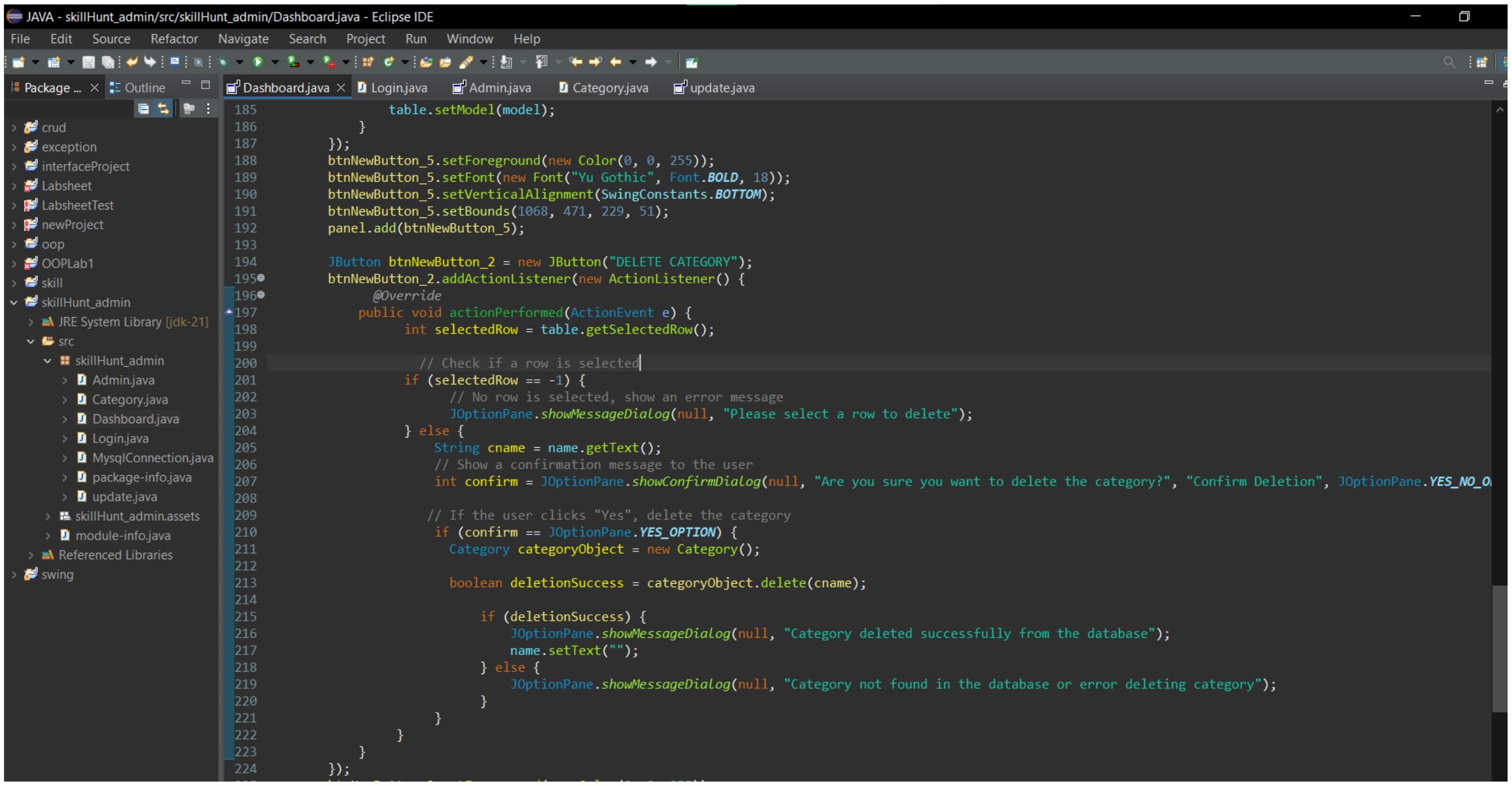
This is checked when the admin clicks the add button when the values entered are not. If the admin does not enter the value and click the add button then show an error message. (see Figure 4.16)



```java
package skillHunt_admin;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Login {
    protected String username;
    protected String password;
    protected MysqlConnection dbc;

    public Login(String username, String password) {
        this.username = username;
        this.password = password;

    }

    public Login() {
         dbc= new MysqlConnection();

    }

    public Boolean log(String username, String password) {
        try {
            String sql = "SELECT * FROM admin WHERE name=? AND password=?";
            PreparedStatement statement = dbc.connectDb().prepareStatement(sql);
            statement.setString(1, username);
            statement.setString(2, password);
            ResultSet rs=statement.executeQuery();
            if(rs.next()) {
                return true;
            }else {
                return false;
            }
        }catch(SQLException e) {
            return false;
        }
    }
}
```

Figure 4.17: Valid Login

This is to check Admin enter the correct password and username. If the admin enters an incorrect username and password then shows an error message and if it is correct the admin can enter the admin panel. (see Figure 4.17)



```java
                table.setModel(model);
            }
        });
        btnNewButton_5.setForeground(new Color(0, 0, 255));
        btnNewButton_5.setFont(new Font("Yu Gothic", Font.BOLD, 18));
        btnNewButton_5.setVerticalAlignment(SwingConstants.BOTTOM);
        btnNewButton_5.setBounds(1068, 471, 229, 51);
        panel.add(btnNewButton_5);

        JButton btnNewButton_2 = new JButton("DELETE CATEGORY");
        btnNewButton_2.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int selectedRow = table.getSelectedRow();

                // Check if a row is selected
                if (selectedRow == -1) {
                    // No row is selected, show an error message
                    JOptionPane.showMessageDialog(null, "Please select a row to delete");
                } else {
                    String cname = name.getText();
                    // Show a confirmation message to the user
                    int confirm = JOptionPane.showConfirmDialog(null, "Are you sure you want to delete the category?", "Confirm Deletion", JOptionPane.YES_NO_O

                  // If the user clicks "Yes", delete the category
                    if (confirm == JOptionPane.YES_OPTION) {
                      Category categoryObject = new Category();

                      boolean deletionSuccess = categoryObject.delete(cname);

                          if (deletionSuccess) {
                              JOptionPane.showMessageDialog(null, "Category deleted successfully from the database");
                              name.setText("");
                          } else {
                              JOptionPane.showMessageDialog(null, "Category not found in the database or error deleting category");
                          }
                    }
                }
            }
        });
```

Figure 4.18: Confirm delete

This is checked when the admin clicks on the delete button and checks if the admin selects the row first. If it is not then showing Please select the row. If the admin selects the row and clicks on the delete button then asks if you want to delete this then the admin confirms then delete if it is not then do not delete the row. (see Figure 4.18)

4.5 Evidence of code Implement

4.5.1 Java Admin panel

Connection

```
package skillHunt_admin;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class MysqlConnection {
	private String username;
	private String pwd;
	private String dbname;
	private Connection conn;

	public MysqlConnection() {
		this.dbname = "final_project";
		this.username = "root";
		this.pwd = "";
		connectDb();
	}

	public Connection connectDb() {
```

```
                try {
                        conn                                                                        =
DriverManager.getConnection("jdbc:mysql://localhost:3306/"+dbname,username,pwd);
                        return conn;
                }
                catch(SQLException e) {
                        System.out.println(e);
                        return conn;
                }


        }
}
```

Admin Login

```
package skillHunt_admin;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Login {
	protected String username;
	protected String password;
	protected MysqlConnection dbc;

	public Login(String username, String password) {
		this.username = username;
		this.password = password;

	}

	public Login() {
		 dbc= new MysqlConnection();

	}
```

```
        public int log(String username, String password) {
                try {
                        String sql = "SELECT name,password FROM users WHERE name=?
AND password=? AND role=admin";
                        PreparedStatement                    statement                    =
dbc.connectDb().prepareStatement(sql);
                        statement.setString(1, username);
               statement.setString(2, password);
                        ResultSet rs=statement.executeQuery();

                        if(username.isEmpty() | password.isEmpty() ) {
                                return 3;
                        }else {
                                if(rs.next()) {
                                        return 1;
                                }else {
                                        return 0;
                                }
                        }

                }catch(SQLException e) {
                        return 2;
                }
        }
}
```

Admin Login design

```
package skillHunt_admin;

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.border.EmptyBorder;
import java.awt.Color;
import javax.swing.JLabel;
import javax.swing.JOptionPane;

import java.awt.Font;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.ImageIcon;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JCheckBox;
```

```
public class Admin extends JFrame {

	private static final long serialVersionUID = 1L;
	private JPanel contentPane;
	private JTextField un;
	private JTextField pw;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {

		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					Admin frame = new Admin();
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}
```

```
/**
 * Create the frame.
 */
public Admin() {



        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(500, 200, 600, 400);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        setContentPane(contentPane);
        contentPane.setLayout(null);

        JPanel panel = new JPanel();
        panel.setBackground(new Color(0, 0, 106));
        panel.setBounds(0, 0, 586, 363);
        contentPane.add(panel);
        panel.setLayout(null);

        JLabel log = new JLabel("ADMIN LOG IN");
        log.setFont(new Font("Gill Sans MT", Font.BOLD, 25));
        log.setForeground(new Color(0, 255, 255));
        log.setBounds(217, 49, 210, 51);
        panel.add(log);
```

```
JLabel user = new JLabel("User Name");
user.setFont(new Font("Tahoma", Font.PLAIN, 16));
user.setForeground(new Color(255, 255, 255));
user.setBounds(86, 140, 119, 36);
panel.add(user);

un = new JTextField();
un.setFont(new Font("Yu Gothic UI", Font.BOLD, 15));
un.setBounds(205, 140, 222, 36);
panel.add(un);
un.setColumns(10);

JLabel word = new JLabel("Password");
word.setFont(new Font("Tahoma", Font.PLAIN, 16));
word.setForeground(new Color(255, 255, 255));
word.setBounds(86, 202, 113, 36);
panel.add(word);

pw = new JPasswordField();
pw.setFont(new Font("Yu Gothic UI", Font.BOLD, 15));
pw.setBounds(207, 205, 220, 36);
panel.add(pw);
pw.setColumns(10);
```

```
                JButton exit = new JButton("Exit");
                exit.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                JFrame frmLogin = new JFrame("Exit");
                                if (JOptionPane.showConfirmDialog(frmLogin, "Confirm if you
want to exit","Login Systems",

        JOptionPane.YES_NO_OPTION)==JOptionPane.YES_NO_OPTION) {
                                        System.exit(0);
                                }
                        }
                });
                exit.setForeground(new Color(255, 255, 255));
                exit.setBackground(new Color(255, 0, 0));
                exit.setFont(new Font("Times New Roman", Font.BOLD, 14));
                exit.setBounds(77, 275, 119, 44);
                panel.add(exit);

                JButton reset = new JButton("Reset");
                reset.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                un.setText(null);
                                pw.setText(null);
                        }
                });
        reset.setFont(new Font("Times New Roman", Font.BOLD, 14));
        reset.setBounds(246, 275, 113, 44);
        panel.add(reset);

        JButton login = new JButton("Login");
        login.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String password= pw.getText();
                String username= un.getText();

                Login lobj = new Login();
                int mzg = lobj.log(username, password);

                    if(mzg ==1) {
                        JOptionPane.showMessageDialog(null,        "Login
Successful");
                        dispose();
                        Dashboard db = new Dashboard();
                        db.setVisible(true);
                    }else if(mzg==3) {
                        JOptionPane.showMessageDialog(null,   "Fill   the
details");
                    }else {
                        JOptionPane.showMessageDialog(null,      "Invalid
name or password");
                    }
            }
        });
```

```
            login.setForeground(new Color(255, 255, 255));
            login.setBackground(new Color(0, 128, 64));
            login.setFont(new Font("Times New Roman", Font.BOLD, 14));
            login.setBounds(402, 275, 124, 44);
            panel.add(login);

            JLabel lblNewLabel = new JLabel("New label");
            lblNewLabel.setIcon(new
ImageIcon(Admin.class.getResource("/skillHunt_admin/assets/user-login-icon-29
(1).png")));
            lblNewLabel.setBounds(50, 10, 120, 120);
            panel.add(lblNewLabel);

            JCheckBox show = new JCheckBox("Show password");
            show.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        if(show.isSelected()) {
                              ((JPasswordField) pw).setEchoChar((char)0);
                        }else {
                              ((JPasswordField) pw).setEchoChar('*');
                        }
                  }
            });
            show.setForeground(new Color(255, 255, 255));
            show.setBackground(new Color(0, 0, 128));
            show.setBounds(433, 212, 128, 21);
            panel.add(show);
      } }
```

Category management

```
package skillHunt_admin;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.table.DefaultTableModel;

public class Category{

        protected String cname;
        protected int id;
        protected MysqlConnection dbc;

        public Category( int id, String cname) {
            this.id = id;
            this.cname = cname;

        }

        public Category() {
            dbc= new MysqlConnection();

        }
```

```
//check exits
public int exits(String cname) {


     try{
          String checkDuplicateSql = "SELECT * FROM categories
WHERE category_name = ?";
        PreparedStatement checkStatement =
dbc.connectDb().prepareStatement(checkDuplicateSql);
        checkStatement.setString(1, cname);
        ResultSet resultSet = checkStatement.executeQuery();


        if (resultSet.next()) {
           // Category name already exit
           return 0;
        }else {
          return 1;
        }
      }catch(SQLException s) {
             return 0;
      }
}



//insert category
public Boolean insert(String cname) {
        try {
                // Create SQL statement
  String sql = "INSERT INTO categories (category_name) VALUES (?)";
  PreparedStatement statement = dbc.connectDb().prepareStatement(sql);

  // Set values for parameters
  statement.setString(1, cname);

  if(cname.isEmpty()) {
        return false;
                }else {

  // Execute the statement
                        int rowsInserted = statement.executeUpdate();
                        if (rowsInserted > 0) {

                                return true;
                        }else {
                                return false;
                        }
                }

        }catch(SQLException e) {
                return false;
        }
}
```

```
//view category
public DefaultTableModel view() {
    DefaultTableModel model = new DefaultTableModel();

    try {
        // Retrieve data from the database
        String sql = "SELECT * FROM categories";
        PreparedStatement statement = dbc.connectDb().prepareStatement(sql);

        ResultSet resultSet = statement.executeQuery();

        // Get column names
        int columnCount = resultSet.getMetaData().getColumnCount();
        for (int i = 1; i <= columnCount; i++) {
            model.addColumn(resultSet.getMetaData().getColumnName(i));
        }

        // Get data rows
        while (resultSet.next()) {
            Object[] rowData = new Object[columnCount];
            for (int i = 0; i < columnCount; i++) {
                rowData[i] = resultSet.getObject(i + 1);
            }
            model.addRow(rowData);
        }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return model;
    }



    //edit category
    public int update(String oldName, String newName) {
        try {
            if (newName.isEmpty()) {
                return 0;
            } else {
                String updateSql = "UPDATE categories SET category_name = ?
WHERE category_name = ?";
                PreparedStatement statement =
dbc.connectDb().prepareStatement(updateSql);
                statement.setString(1, newName);
                statement.setString(2, oldName);

                int rowsUpdated = statement.executeUpdate();
                if (rowsUpdated > 0) {
                    return 1;
                } else {
                    return 2;
                }
            }
        } catch (SQLException e) {
            return 2;
        }
    }


    //delete category
    public boolean delete(String cname) {


            try {
     // Delete category from the database
     String deleteQuery = "DELETE FROM categories WHERE category_name
= ?";
     PreparedStatement                    preparedStatement                    =
dbc.connectDb().prepareStatement(deleteQuery);
     preparedStatement.setString(1, cname);
     int rowsAffected = preparedStatement.executeUpdate();


     if (rowsAffected > 0) {
         return true;
     } else {
        return false;
     }
   } catch (SQLException e) {


     return false;
   }
    }
}
```

Admin Dashboard

```
package skillHunt_admin;

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import java.awt.Color;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.SwingConstants;
import java.awt.Font;
import javax.swing.JTextField;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
import javax.swing.JScrollPane;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.print.PrinterException;
```

```
public class Dashboard extends JFrame {

	private static final long serialVersionUID = 1L;
	private JPanel contentPane;
	private JTextField name;
	DefaultTableModel model;
	private JTable table;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					Dashboard frame = new Dashboard();
					frame.setVisible(true);

				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}

	/**
	 * Create the frame.
	 */
	public Dashboard() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(0, 0, 1450, 850);
		contentPane = new JPanel();
		contentPane.setBackground(new Color(0, 0, 64));
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

		setContentPane(contentPane);
		contentPane.setLayout(null);

		JPanel panel = new JPanel();
		panel.setBackground(new Color(0, 0, 64));
		panel.setBounds(10, 10, 1400, 776);
		contentPane.add(panel);
		panel.setLayout(null);

		JLabel lblNewLabel = new JLabel("---------------Welcome Admin Panel------
--------");
		lblNewLabel.setBounds(189, 39, 1070, 51);
		lblNewLabel.setForeground(new Color(255, 255, 255));
		lblNewLabel.setFont(new Font("Tahoma", Font.BOLD, 42));
		lblNewLabel.setHorizontalAlignment(SwingConstants.CENTER);
		panel.add(lblNewLabel);

		JLabel lblNewLabel_1 = new JLabel("Enter Category Name ");
		lblNewLabel_1.setFont(new Font("Trebuchet MS", Font.PLAIN, 24));
		lblNewLabel_1.setForeground(new Color(177, 241, 243));
		lblNewLabel_1.setBounds(82, 174, 318, 57);
		panel.add(lblNewLabel_1);

		name = new JTextField();
		name.setFont(new Font("Tahoma", Font.BOLD, 18));
		name.setBounds(355, 186, 368, 43);
		panel.add(name);
		name.setColumns(10);

		JLabel lblNewLabel_2 = new JLabel("New label");
		lblNewLabel_2.setIcon(new
ImageIcon(Dashboard.class.getResource("/skillHunt_admin/assets/add (1).png")));
		lblNewLabel_2.setBounds(954, 146, 81, 85);
		panel.add(lblNewLabel_2);

		JLabel label = new JLabel("New label");
		label.setIcon(new
ImageIcon(Dashboard.class.getResource("/skillHunt_admin/assets/edit (1).png")));
		label.setBounds(951, 248, 84, 85);
		panel.add(label);

		JLabel lblNewLabel_3 = new JLabel("New label");
		lblNewLabel_3.setFont(new Font("Yu Gothic", Font.BOLD, 18));
		lblNewLabel_3.setIcon(new
ImageIcon(Dashboard.class.getResource("/skillHunt_admin/assets/delete (1).png")));
		lblNewLabel_3.setBounds(954, 348, 81, 85);
		panel.add(lblNewLabel_3);
		JButton btnNewButton_4 = new JButton("EXIT");

		btnNewButton_4.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				JFrame frame=new JFrame("Exit");
				if(JOptionPane.showConfirmDialog(frame,  "Confirm  if  you
want to exit","MYSQL connector",

	JOptionPane.YES_NO_OPTION)==JOptionPane.YES_NO_OPTION)
				{
					System.exit(0);
				}
			}
		});

		btnNewButton_4.setBackground(new Color(255, 0, 0));
		btnNewButton_4.setVerticalAlignment(SwingConstants.BOTTOM);
		btnNewButton_4.setFont(new Font("Yu Gothic", Font.BOLD, 20));
		btnNewButton_4.setForeground(new Color(255, 255, 255));
		btnNewButton_4.setBounds(1169, 600, 160, 60);
		panel.add(btnNewButton_4);

		JScrollPane scrollPane = new JScrollPane();
		scrollPane.addMouseListener(new MouseAdapter() {
			@Override
			public void mouseClicked(MouseEvent e) {
				DefaultTableModel			tblmodel			=
(DefaultTableModel)table.getModel();
				String tblname = tblmodel.getValueAt(table.getSelectedRow(),
2).toString();

				name.setText(tblname);

			}
		});
		scrollPane.setBounds(202, 283, 500, 500);
		panel.add(scrollPane);

		table = new JTable(model);
		scrollPane.setViewportView(table);
		table.setModel(new DefaultTableModel(
			new Object[][] {
			},
			new String[] {
				"Id", "Category_name"
			}
		));
```

```
          JButton add = new JButton("ADD CATEGORY");
          add.addActionListener(new ActionListener() {
               public void actionPerformed(ActionEvent e) {
                 String cname = name.getText();
                 Category categoryObject = new Category();
                 int suc = categoryObject.exits(cname);



                 if (suc == 0) {
                   JOptionPane.showMessageDialog(null,     "Category     already
exists!");
                 } else {
                     boolean success = categoryObject.insert(cname);


                   if (success) {
                     JOptionPane.showMessageDialog(null,    "Category    inserted
successfully!");
                   } else {
                     JOptionPane.showMessageDialog(null,    "Failed    to    insert
category. Please try again.");
                   }
                 }
               }


          });
```

```
        add.setVerticalAlignment(SwingConstants.BOTTOM);
        add.setForeground(new Color(0, 0, 255));
        add.setFont(new Font("Yu Gothic", Font.BOLD, 18));
        add.setBounds(1067, 158, 229, 51);
        panel.add(add);

        JButton btnNewButton_5 = new JButton("VIEW CATEGORY");
        btnNewButton_5.addActionListener(new ActionListener() {
                @Override
public void actionPerformed(ActionEvent e) {
                        Category cobj = new Category();
   model = cobj.view();
   table.setModel(model);
}
        });
        btnNewButton_5.setForeground(new Color(0, 0, 255));
        btnNewButton_5.setFont(new Font("Yu Gothic", Font.BOLD, 18));
        btnNewButton_5.setVerticalAlignment(SwingConstants.BOTTOM);
        btnNewButton_5.setBounds(1068, 471, 229, 51);
        panel.add(btnNewButton_5);

        JButton btnNewButton_2 = new JButton("DELETE CATEGORY");
        btnNewButton_2.addActionListener(new ActionListener() {
                 @Override
public void actionPerformed(ActionEvent e) {
                         int selectedRow = table.getSelectedRow();
                        // Check if a row is selected
                      if (selectedRow == -1) {
                          // No row is selected, show an error message
                          JOptionPane.showMessageDialog(null, "Please select a row to
delete");
                      } else {
                                  String cname = name.getText();
                                  // Show a confirmation message to the user
                              int confirm = JOptionPane.showConfirmDialog(null, "Are
you sure you want to delete the category?", "Confirm Deletion",
JOptionPane.YES_NO_OPTION);


                             // If the user clicks "Yes", delete the category
                              if (confirm == JOptionPane.YES_OPTION) {
                        Category categoryObject = new Category();


                        boolean deletionSuccess = categoryObject.delete(cname);
                        if (deletionSuccess) {
                            JOptionPane.showMessageDialog(null, "Category deleted
successfully from the database");
                            name.setText("");
                        } else {
                            JOptionPane.showMessageDialog(null, "Category not found
in the database or error deleting category");
                        }
                              }
                      }
        }
            });
```

```
btnNewButton_2.setForeground(new Color(0, 0, 255));
btnNewButton_2.setVerticalAlignment(SwingConstants.BOTTOM);
btnNewButton_2.setFont(new Font("Yu Gothic", Font.BOLD, 18));
btnNewButton_2.setBounds(1067, 368, 229, 51);
panel.add(btnNewButton_2);

JButton btnNewButton_1 = new JButton("EDIT CATEGORY");
btnNewButton_1.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {



                dispose();
                update edit = new update();
                edit.setVisible(true);
        }
});
btnNewButton_1.setVerticalAlignment(SwingConstants.BOTTOM);
btnNewButton_1.setForeground(new Color(0, 0, 255));
btnNewButton_1.setFont(new Font("Yu Gothic", Font.BOLD, 18));
btnNewButton_1.setBounds(1067, 260, 229, 51);
panel.add(btnNewButton_1);

JButton btnNewButton = new JButton("PRINT");
btnNewButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                try {
                        table.print();
```

```
                    }catch(PrinterException p) {
                            System.out.format("No printer found");
                    }
            }
        });
        btnNewButton.setBackground(new Color(222, 189, 35));
        btnNewButton.setFont(new Font("Yu Gothic", Font.BOLD, 20));
        btnNewButton.setVerticalAlignment(SwingConstants.BOTTOM);
        btnNewButton.setForeground(new Color(255, 255, 255));
        btnNewButton.setBounds(954, 600, 160, 60);
        panel.add(btnNewButton);

        JLabel lblNewLabel_4 = new JLabel("");
        lblNewLabel_4.setIcon(new
ImageIcon(Dashboard.class.getResource("/skillHunt_admin/assets/view (1).png")));
        lblNewLabel_4.setBounds(954, 459, 81, 85);
        panel.add(lblNewLabel_4);

        JButton btnNewButton_3 = new JButton("RESET");
        btnNewButton_3.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                    name.setText("");
            }
        });
```

```
		btnNewButton_3.setBackground(new Color(0, 64, 64));
		btnNewButton_3.setForeground(new Color(255, 255, 255));
		btnNewButton_3.setVerticalAlignment(SwingConstants.BOTTOM);
		btnNewButton_3.setFont(new Font("Yu Gothic", Font.BOLD, 20));
		btnNewButton_3.setBounds(1067, 686, 160, 60);
		panel.add(btnNewButton_3);

	}
}
```

Update Design

```
package skillHunt_admin;

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import java.awt.Color;
import javax.swing.JLabel;
import javax.swing.JOptionPane;

import java.awt.Font;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.SwingConstants;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class update extends JFrame {

	private static final long serialVersionUID = 1L;
	private JPanel contentPane;
	private JTextField oldtext;
	private JTextField newtext;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					update frame = new update();
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}
	/**
	 * Create the frame.
	 */
	public update() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(500, 200, 550, 500);
		contentPane = new JPanel();
		contentPane.setBackground(new Color(0, 0, 64));
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

		setContentPane(contentPane);
		contentPane.setLayout(null);

		JLabel lblNewLabel = new JLabel("========UPDATE
CATEGORY========");
		lblNewLabel.setFont(new Font("Tahoma", Font.BOLD, 20));
		lblNewLabel.setForeground(new Color(255, 255, 255));
		lblNewLabel.setBounds(22, 25, 464, 48);
		contentPane.add(lblNewLabel);

		JLabel lblNewLabel_1 = new JLabel("Enter current name :");
		lblNewLabel_1.setFont(new Font("Yu Gothic Medium", Font.PLAIN, 16));
		lblNewLabel_1.setForeground(new Color(128, 255, 255));
		lblNewLabel_1.setBounds(57, 118, 238, 27);
		contentPane.add(lblNewLabel_1);
```

```
oldtext = new JTextField();
oldtext.setFont(new Font("Tahoma", Font.BOLD, 17));
oldtext.setBounds(67, 155, 298, 34);
contentPane.add(oldtext);
oldtext.setColumns(10);

JLabel lblNewLabel_2 = new JLabel("Enter new name :");
lblNewLabel_2.setFont(new Font("Yu Gothic Medium", Font.PLAIN, 16));
lblNewLabel_2.setForeground(new Color(128, 255, 255));
lblNewLabel_2.setBounds(57, 234, 238, 27);
contentPane.add(lblNewLabel_2);

newtext = new JTextField();
newtext.setFont(new Font("Tahoma", Font.BOLD, 17));
newtext.setBounds(67, 271, 298, 34);
contentPane.add(newtext);
newtext.setColumns(10);

JButton btnNewButton = new JButton("UPDATE");
btnNewButton.addActionListener(new ActionListener() {
        @Override
 public void actionPerformed(ActionEvent e) {


                String oldName = oldtext.getText();
        String newName = newtext.getText();
        Category categoryObject = new Category();
```

```
                    int edit = categoryObject.update(oldName,newName);


                    if (edit==0) {
                JOptionPane.showMessageDialog(null, "Please enter values!");
                    }else {
                int confirm = JOptionPane.showConfirmDialog(null, "Are you sure you want
to edit the category?", "Confirm Update", JOptionPane.YES_NO_OPTION);


                        if (confirm == JOptionPane.YES_OPTION) {
                        if (edit==1) {
                        JOptionPane.showMessageDialog(null, "Please enter values!");
                          JOptionPane.showMessageDialog(null, "Category Updated
successfully !!");
                          oldtext.setText("");
                          newtext.setText("");
                        } else {
                          JOptionPane.showMessageDialog(null, "Category not found
in the database or error updating category");
                        }
                        }
                 }
                    }
                });
                btnNewButton.setFont(new Font("Yu Gothic", Font.BOLD, 18));
                btnNewButton.setVerticalAlignment(SwingConstants.BOTTOM);
                btnNewButton.setBounds(60, 367, 136, 48);
                contentPane.add(btnNewButton);
                JButton btnNewButton_1 = new JButton("GO TO MAIN");
                btnNewButton_1.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                dispose();
                                Dashboard dash = new Dashboard();
                                dash.setVisible(true);
                        }
                });
                btnNewButton_1.setBackground(new Color(30, 145, 75));
                btnNewButton_1.setForeground(new Color(255, 255, 255));
                btnNewButton_1.setFont(new Font("Yu Gothic", Font.BOLD, 18));
                btnNewButton_1.setVerticalAlignment(SwingConstants.BOTTOM);
                btnNewButton_1.setBounds(259, 364, 182, 51);
                contentPane.add(btnNewButton_1);
        }
}
```

4.5.2 SQL Database Design

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_
DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBST
ITUTION';


-- -----------------------------------------------------
-- Schema Skill_hunt
-- -----------------------------------------------------


-- -----------------------------------------------------
-- Schema Skill_hunt
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `Skill_hunt` DEFAULT CHARACTER SET utf8 ;
USE `Skill_hunt` ;


-- -----------------------------------------------------
-- Table `Skill_hunt`.`members`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`members` (
  `Mid` INT NOT NULL AUTO_INCREMENT,
  `Mname` VARCHAR(45) NOT NULL,
  `Memail` VARCHAR(45) NOT NULL,
  `Mpw` VARCHAR(8) NOT NULL,
  PRIMARY KEY (`Mid`));
```

```
-- -----------------------------------------------------
-- Table `Skill_hunt`.`users`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`users` (
  `Uid` INT NOT NULL AUTO_INCREMENT,
  `Uname` VARCHAR(30) NOT NULL,
  `Ugamil` VARCHAR(45) NOT NULL,
  `members_Mid` INT NOT NULL,
  PRIMARY KEY (`Uid`),
  INDEX `fk_users_members_idx` (`members_Mid` ASC) VISIBLE,
  CONSTRAINT `fk_users_members`
    FOREIGN KEY (`members_Mid`)
    REFERENCES `Skill_hunt`.`members` (`Mid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `Skill_hunt`.`company`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`company` (
  `Cid` INT NOT NULL AUTO_INCREMENT,
  `Cname` VARCHAR(20) NOT NULL,
  `Cmail` VARCHAR(25) NOT NULL,
  `cpw` VARCHAR(8) NOT NULL,
  PRIMARY KEY (`Cid`));
```

```
-- -----------------------------------------------------
-- Table `Skill_hunt`.`categories`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`categories` (
  `cateId` INT NOT NULL AUTO_INCREMENT,
  `cateName` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`cateId`));


-- -----------------------------------------------------
-- Table `Skill_hunt`.`jobs`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`jobs` (
  `Jid` INT NOT NULL AUTO_INCREMENT,
  `title` VARCHAR(30) NOT NULL,
  `deadline` VARCHAR(11) NOT NULL,
  `Jlocation` VARCHAR(15) NOT NULL,
  `company_Cid` INT NOT NULL,
  `categories_cateId` INT NOT NULL,
  PRIMARY KEY (`Jid`),
  INDEX `fk_jobs_company1_idx` (`company_Cid` ASC) VISIBLE,
  INDEX `fk_jobs_categories1_idx` (`categories_cateId` ASC) VISIBLE,
  CONSTRAINT `fk_jobs_company1`
    FOREIGN KEY (`company_Cid`)
    REFERENCES `Skill_hunt`.`company` (`Cid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_jobs_categories1`
    FOREIGN KEY (`categories_cateId`)
    REFERENCES `Skill_hunt`.`categories` (`cateId`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `Skill_hunt`.`interviews`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`interviews` (
  `Iid` INT NOT NULL AUTO_INCREMENT,
  `scheduleDate` VARCHAR(11) NOT NULL,
  `Ilocation` VARCHAR(15) NOT NULL,
  `company_Cid` INT NOT NULL,
  `members_Mid` INT NOT NULL,
  PRIMARY KEY (`Iid`),
  INDEX `fk_interviews_company1_idx` (`company_Cid` ASC) VISIBLE,
  INDEX `fk_interviews_members1_idx` (`members_Mid` ASC) VISIBLE,
  CONSTRAINT `fk_interviews_company1`
    FOREIGN KEY (`company_Cid`)
    REFERENCES `Skill_hunt`.`company` (`Cid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_interviews_members1`
    FOREIGN KEY (`members_Mid`)
    REFERENCES `Skill_hunt`.`members` (`Mid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `Skill_hunt`.`messages`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`messages` (
  `Nid` INT NOT NULL AUTO_INCREMENT,
  `Ncontent` VARCHAR(40) NOT NULL,
  `Ndate` DATETIME NOT NULL,
  PRIMARY KEY (`Nid`));

-- -----------------------------------------------------
-- Table `Skill_hunt`.`members_jobs`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`members_jobs` (
  `MJid` INT NOT NULL AUTO_INCREMENT,
  `members_Mid` INT NOT NULL,
  `jobs_Jid` INT NOT NULL,
  INDEX `fk_members_has_jobs_jobs1_idx` (`jobs_Jid` ASC) VISIBLE,
  INDEX `fk_members_has_jobs_members1_idx` (`members_Mid` ASC) VISIBLE,
  PRIMARY KEY (`MJid`),
  CONSTRAINT `fk_members_has_jobs_members1`
    FOREIGN KEY (`members_Mid`)
    REFERENCES `Skill_hunt`.`members` (`Mid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_members_has_jobs_jobs1`
    FOREIGN KEY (`jobs_Jid`)
    REFERENCES `Skill_hunt`.`jobs` (`Jid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `Skill_hunt`.`admin`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`admin` (
  `aId` INT NOT NULL AUTO_INCREMENT,
  `Aname` VARCHAR(20) NOT NULL,
  `Apassword` VARCHAR(8) NOT NULL,
  PRIMARY KEY (`aId`));
```

```
-- -----------------------------------------------------
-- Table `Skill_hunt`.`members_messages`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Skill_hunt`.`members_messages` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `members_Mid` INT NOT NULL,
  `messages_Nid` INT NOT NULL,
  `jobs_Jid` INT NOT NULL,
  INDEX `fk_members_has_messages_messages1_idx` (`messages_Nid` ASC) VISIBLE,
  INDEX `fk_members_has_messages_members1_idx` (`members_Mid` ASC) VISIBLE,
  PRIMARY KEY (`id`),
  INDEX `fk_members_messages_jobs1_idx` (`jobs_Jid` ASC) VISIBLE,
  CONSTRAINT `fk_members_has_messages_members1`
    FOREIGN KEY (`members_Mid`)
    REFERENCES `Skill_hunt`.`members` (`Mid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_members_has_messages_messages1`
    FOREIGN KEY (`messages_Nid`)
    REFERENCES `Skill_hunt`.`messages` (`Nid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
 CONSTRAINT `fk_members_messages_jobs1`
  FOREIGN KEY (`jobs_Jid`)
  REFERENCES `Skill_hunt`.`jobs` (`Jid`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION);



SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

4.6 Summary

In this chapter we discussed about the implement part of the system. You can see the code for the designing chapter. We discussed test planes and test cases also. We can get better idea about how they work correctly or not. In next chapter we will discuss all the summary of this documentation.

Chapter 5 : Conclusion

In the entire document we discussed about Skill Hunt job port system. First chapter, we started with introducing the system and why I chose this system to build, including the motivation behind the project, the Aims, and Objectives of the Project. The scope of the Skill Hunt project is the development of a web and mobile platform and intern job opportunities. In next second chapter, we discussed about analysis part of the system. We used the waterfall method for this system. We identified what are the functions and non-functions of this system. We know about what are similar systems like this system and the mistakes they have. In third chapter, we discussed the diagrams related to the system. Use-case diagrams, Activity diagrams, class diagrams, and state diagrams represent parts of the system and it is easy to understand how they interact with the system. We see how the system is designed in Figma and the admin part in Java. After Next chapter, we discussed implement the system and how the SQL, web, and Java code are implemented. We tested our system using test plan and test cases. According to these steps we build our system.

References

www.youtube.com

www.w3school.com

https://chat.openai.com/

https://www.javatpoint.com

www.google.com