1. PROBLEM STATEMENT

Career Connect

(An Application to make job search and posting easier)

The current job search process can be time-consuming and frustrating for both job seekers and recruiters. Job seekers often spend hours manually searching for openings on company career pages and job search websites, but may still receive spam messages and notifications from unauthorized sources.

Even on Linkedin, people without proper authorization can post jobs and spam the jobseekers.

Recruiters, on the other hand, may have difficulty finding qualified candidates in a timely manner, which can impact their business.

To address these issues, we proposed CareerConnect (CC). with CC.

1. Job Seekers can create accounts.
2. Select their skills, experience, and preferred location from a predetermined list and upload their resumes.
3. Job Seekers can search for jobs either using JobID or JobTitle.
4. Job Seekers can also view job openings from top companies in the world within the application using Google Programmable apis.
5. Recruiters can also create an account on CC but must use a professional email address.
6. Recruiters can then post job openings and view job seekers that match their requirements and send notifications to the job seekers.

Overall, CC aims to streamline the job search process for both job seekers and recruiters, making it easier for everyone to find their ideal job match.

2. SRS DOCUMENTATION - REQUIREMENTS ELICITATION

ID	Details	Functionalities	Priorities
R1	CC must be able to register the job seeker	Functional	Must have
R2	CC must be able to authenticate the mail of job seeker	Functional	Must have
R3	CC must be able to handle and authenticate login of job seeker	Functional	Must have
R4	CC must allow the jobseekers to select their experience, skill set, location and resume upload	Functional	Must have
R5	CC must show the basic profile of jobseekers after they logged in	Functional	Must have
R6	CC must allow the job seekers to edit their profile	Functional	Must have
R7	CC must allow the job seekers to change the password after logging	Functional	Must have
R8	CC must allow job seekers to custom search jobs in top companies	Functional	Must have
R9	CC must allow job seekers to search for jobs	Functional	Must have

	Posted in CC Application.		
R10	CC must allow jobseekers for forgot passwords	Functional	Must have
R11	CC must allow Recruiters to register	Functional	Must have
R12	CC must accept only professional mail from Recruiters	Functional	Must have
R13	CC must authenticate the emails from Recruiters	Functional	Must have
R14	CC must be able to handle the login details of Recruiters	Functional	Must have
R15	CC must allow recruiters to store their information	Functional	Must have
R16	CC must allow recruiters to post jobs	Functional	Must have
R17	CC must allow recruiters to view eligible candidates	Functional	Must have
R18	CC must allow recruiters to notify the eligible candidates	Functional	Must have
R19	CC must show the basic details of recruiters after logged in	Functional	Must have
R20	CC must allow recruiters to change their password after logging in	Functional	Must have

R21	CC must be able to logout the recruiters	Functional	Must have
R22	CC must allow recruiters for forgot password	Functional	Must have

3. SYSTEM REQUIREMENT SPECIFICATION

Software Requirements:

- Operating System: Windows11
- Language: Python
- Framework Used: Flask
- Database: Sqlite3
- IDE: Pycharm Professional
- APIs Used:
  - Google Programmable custom search API
  - MailGun SMTP API

Hardware Requirements:

- Personal Computer with keyboard and mouse maintained with uninterrupted power supply and internet.
- Processor : Intel® core™ i5
- Installed Memory (RAM) : 8.00 GB
- Hard Disc : 1 TB

4. REQUIREMENTS MODELING

The most important factor for the success of an IS project is whether the software product satisfies its users' requirements. Models constructed from an analysis perspective focus on determining these requirements. This means Requirement Model includes gathering and documenting facts and requests.

The use case model gives a perspective on many user requirements and models them in terms of what the software system can do for the user. Before the design of software which satisfies user requirements, we must analyze both the logical structure of the problem situation and also the ways that its logical elements interact with each other. We must also need to verify how different, possibly conflicting, requirements affect each other. Then we must communicate this understanding clearly and unambiguously to those who will design and build the software.

Use-case diagrams graphically represent system behavior (use cases). These diagrams present a high-level view of how the system is used viewed from an outsider’s (actor’s) perspective. A use-case diagram may contain all or some of the use cases of a system. A use-case diagram can contain

➢ ·actors ("things" outside the system)

➢ ·use cases (system boundaries identifying what the system should do) ➢ interactions or relationships between actors and use cases in the system including the associations, dependencies, and generalizations. Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the system's behavior as implemented.

5. IDENTIFICATION OF ACTORS and USE CASES

Identification of ACTORS:

Actors represent system users. They are NOT part of the system.They represent anyone or anything that interacts with the system.

An actor is someone or something that:

- Interacts with or uses the system
- Provides input to and receives information from the system
- Is external to the system and has no control over the use cases.

Actors are discovered by examining:

- Who directly uses the system.
- Who is responsible for maintaining the system
- External hardware used by the system.
- Other systems that need to interact with the system

The needs of the actor are used to develop use cases. This ensures that the system will be what the user expects.

Graphical depiction:

An actor is a stereotype of a class and is depicted as a “stickman” on a use-case diagram. For example,

The Actors identified in the system are

1. Job Seeker
2. Recruiter

1) Job Seeker: Job Seekers have to create an account & login into their account to
    - View their Profile
    - Update their profile
    - To search for jobs
    - To apply for jobs
    - To Change Passwords

And they has to logout after the desired action is completed.

2) Recruiter: Recruiters have to create an account & login into their account to
    - post a Job
    - To see eligible candidates for posted jobs
    - To send notifications to eligible candidates
    - To Update Password.
    - To view jobs posted by them.
    - To Change Password.

And they have to log out after the desired action is completed.

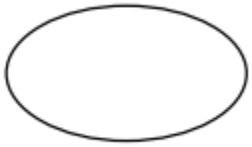
Identification of Use-Cases Or Sub-Use-Cases:

A use case can be described as a specific way of using the system from a user's perspective. A more detailed description might characterize a use case as

➢ A pattern of behavior the system exhibits

➢ A sequence of related transactions performed by an actor and the system

The UML notation for the use case is:



Purpose of use cases:

➢ Well structured use cases denote essential system or subsystem behaviors only, and are neither overly general nor too specific.

➢ A use case represents a functional requirement of the system as a whole

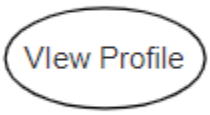
➢ Use cases represent an external view of the system

➢ A use case describes a set of sequences, in which each sequence represents the interaction of the things outside the system with the system itself

Use-cases identified for the CareerConnect are:

1. Use-Case: Login
   This is a use case which is used by an actor to log on to the system and view the available set of operations that they can perform.

2. Use-Case name: View Profile
   System allows Job Seeker to view their profile

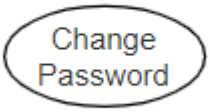


3. Use-Case name: Update Profile

System allows the Job Seeker to Update their profiles

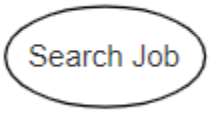
4. Use-Case name: Change Password

System allows both recruiters and Jobseekers to change their passwords.



5. Use-Case name: Search Job

System allows Job Seekers to search for Jobs either using JobID or Job Title



6. Use -Case name: Apply Job

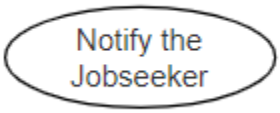
System allows Job Seeker to Apply to the Jobs.

7. Use-Case name: Post Job

System allows Recruiters to Post Jobs

8. Use-Case name: Notify the Job Seeker

System allows Recruiters to send notifications to jobseekers if their profile matches.



9. Use-Case name: Created by me

System allows the Recruiters to see the list of jobs created by them.

10. Use-Case name: Google API

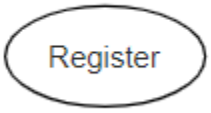
System allows Jobseekers to search for jobs using Google Programmable Custom Search API

11. Use-Case name: Applied List

System allows Job seekers to see the Jobs they applied to

12. Use-Case name: Register

System allows both the Jobseekers and Recruiters to register themselves to perform their actions.



Identification of RELATIONS:

Association Relationship:

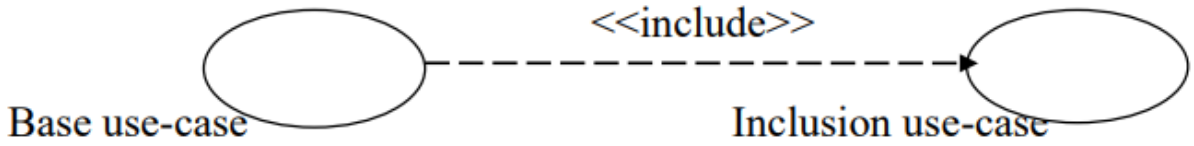
An association provides a pathway for communication. The communication can be between use cases, actors, classes, or interfaces. If two objects are usually considered independently, the relationship is an association.

Dependency Relationship:

A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning. We can provide here

1. Include relationship:

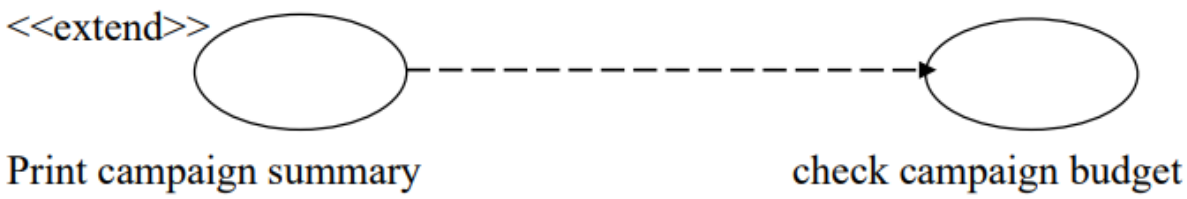
It is a stereotyped relationship that connects a base use case to an inclusion use case. An included relationship specifies how the behavior in the inclusion use case is used by the base use case.



2. Extend relationship:

It is a stereotyped relationship that specifies how the functionality of one use case can be inserted into the functionality of another use case.

<<extend>> is used when you wish to show that a use case provides additional functionality that may be required in another use case.



Print campaign summary check campaign budget

6. CONSTRUCTION OF USE CASE DIAGRAM AND FLOW OF EVENTS.

Use-case diagrams graphically represent system behavior. These diagrams present a high-level view of how the system is used and viewed from an outsider's perspective.

Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behavior of the system as implemented.

Use Case Diagram for CareerConnect

FLOW OF EVENTS

A flow of events is a sequence of transactions performed by the system. They typically contain very detailed information. A flow of events document is typically created in the elaboration phase.

Each use case is documented with the flow of events.

➢A description of events needed to accomplish the required behavior.

➢Written in terms of what the system should do, NOT how it should do it

➢Written in the domain language, not in terms of the implementation.

A flow of events should include

➢ When and how the use case starts and ends.

➢ What interaction the use case has with the actors

➢ What data is needed by the use case

➢ The description of any alternate or exceptional flows.

The flow of events for a use case is contained in a document called the use case specification. Each project should use a standard template for the creation of the use case specification. Includes the following

1. Use case name – Brief Description
2. The flow of events
    1. Basic flow
    2. Alternate flow
    3. Special requirements
    4. Preconditions
    5. Postconditions
    6. Extension points

FLOW OF EVENTS FOR CHANGING PASSWORD

1. **Use-Case:** Changing Password
   Description: This use case is started either by the Jobseekers / Recruiters. This allows them to change their passwords.

2. **Actor:** Jobseeker / Recruiter

3. **The flow of Events:**

3.1 Basic Flow:

- This use case begins when Jobseeker / Recruiter wants to change their password after they logged in.
- They will enter their old password and new password and confirmation password and then click submit.

- If the Old Password is wrong. Then alternate flow *3.2.1* is executed.

- If the new password and confirm password don't match, then alternate flow *3.2.2* is executed.

- After that Jobseekers/Recruiter would submit and save to the database.

3.2 Alternate flow:

3.2.1 Wrong password:

Wrong password is entered by admin while changing password. Jobseeker/Recruiter can re-enter the password or terminate the use case.

3.2.2 Password mismatch:

Values of "new password" and "confirm password" are mismatched. Jobseeker/Recruiter can re-enter both or either of them which is wrong or terminate the use case.

4. **Pre Conditions:** Jobseeker/Recruiter should first log in to the application
5. **Post Conditions:** Next time, Jobseeker/Recruiter can log in with the new password.

FLOW OF EVENTS FOR FORGOT PASSWORD

1. **Use-Case:** Changing Password

   Description: This use case is started either by the Jobseekers / Recruiters. This allows them to change their passwords.
2. **Actor:** Jobseeker / Recruiter
3. **The flow of Events:**

3.1 Basic Flow:

- This use case begins when Jobseeker / Recruiter wants to change their password when they forgot it.
- They will enter their mail id and otp received to mail.

- If the mail doesn't match, then alternate flow 3.2.1 is executed.
- If the otp doesn't match, then alternate flow 3.2.2 is executed.

- New password and confirmation password and then click submit.

- If the new password and confirm password don't match, then alternate flow *3.2.3* is executed.

- After that Jobseekers/Recruiter would submit and save to the database.

3.2 Alternate flow:

3.2.1 Mail Not Found:

This happens when the mail ID entered is not found. The Jobseeker/Recruiter can terminate the flow or re-enter mail.

3.2.2 OTP mismatch:

This happens when the otp doesn't match the sent one. The Jobseeker/Recruiter has to terminate the session.

3.2.3 Password mismatch:

Values of "new password" and "confirm password" are mismatched. Jobseeker/Recruiter can re-enter both or either of them which is wrong or terminate the use case.

4. **Pre Conditions:** Jobseeker/Recruiter should know their mail id
5. **Post Conditions:** Next time, Jobseeker/Recruiter can log in with the new password.

FLOW OF EVENTS FOR POSTING A JOB

1. **Use-Case:** Posting a Job

   Description: This use case is started by the Recruiters. This allows them to post jobs.

2. **Actor:** Recruiter

3. **The flow of Events:**

   3.1 Basic Flow:

   - This use case begins when Recruiters want to post jobs.
   - The Recruiters would enter all the required details and click to review and submit

- If the recruiter wants to change the details, then alternate flow 3.2.1 is executed.

   - After that Recruiter would submit and save it to the database.

   3.2 Alternate flow:

   3.2.1 Change Job Details:

   This happens when the recruiters want to change the details before confirming with the database. The Recruiters have to terminate the flow and start a new one.

4. **Pre Conditions:** Recruiter log in to their account.
5. **Post Conditions:** Next time, Recruiter can then view the eligible list and notify them.

7. BUILDING A BUSINESS PROCESS MODEL USING UML ACTIVITY DIAGRAM

An Activity diagram is a variation of a special case of a state machine, in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations. The purpose of Activity diagram is to provide a view of flows and what is going on inside a use case or among several classes. You can also use activity diagrams to model code-specific information such as a class operation.

Activity diagrams are very similar to a flowchart because you can model a workflow from activity to activity. An activity diagram is basically a special case of a state machine in which most of the states are activities and most of the transitions are implicitly triggered by completion of the actions in the source activities.

• Activity Diagrams also may be created at this stage in the life cycle. These diagrams represent the dynamics of the system. They are flow charts that are used to show the workflow of a system; that is, they show the flow of control from activity to activity in the system, what activities can be done in parallel, and any alternate paths through the flow.

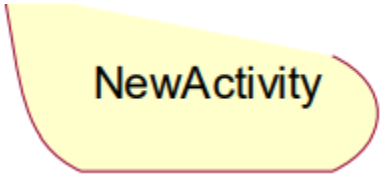
• At this point in the life cycle, activity diagrams may be created to represent the flow across use cases or they may be created to represent the flow within a particular use case.

• Later in the life cycle, activity diagrams may be created to show the workflow for an operation

The following tools are used on the activity diagram toolbox to model activity diagrams:

Activities:

An activity represents the performance of some behavior in the workflow



Transitions:

Transitions are used to show the passing of the flow of control from activity to activity. They are typically triggered by the completion of the behavior in the originating activity.

Decision Points:

When modeling the workflow of a system it is often necessary to show where the flow of control branches based on a decision point. The transitions from a decision point contain a guard condition, which is used to determine which path from the decision point is taken. Decisions along with their guard conditions allow you to show alternate paths through a work flow.

Decision Point

Start state:

A start state explicitly shows the beginning of a workflow on an activity diagram or the beginning of the execution of a state machine on a state chart diagram.

Start State

End state:

An End state represents a final or terminal state on an activity diagram or state chart diagram. Place an end state when you want to explicitly show the end of a workflow on an activity diagram or the end of a state chart diagram. Transitions can only occur into an end state; however, there can be any number of end states per context.

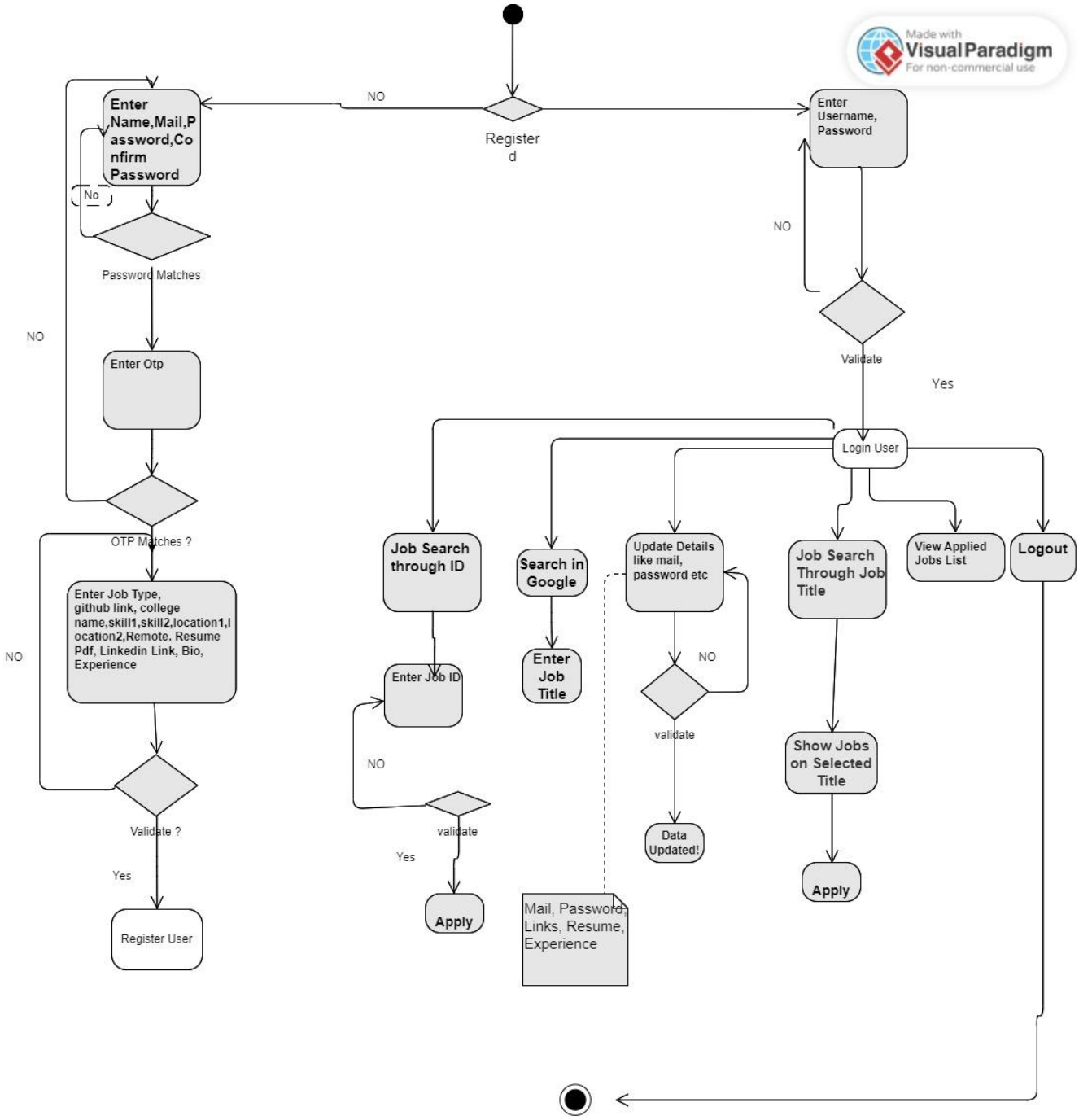
End state

Swim Lanes:

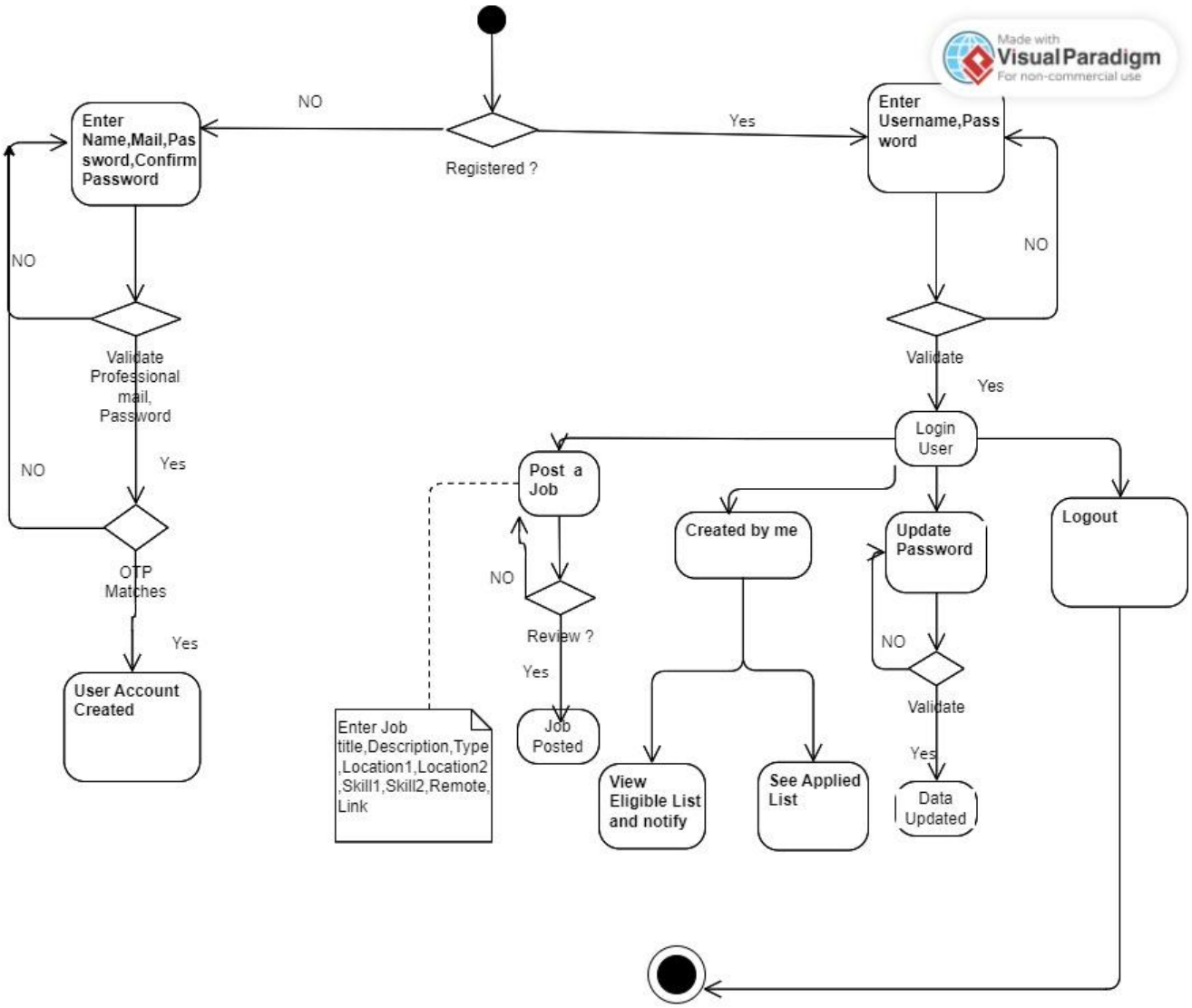
Swim lanes may be used to partition an activity diagram. This typically is done to show what person or organization is responsible for the activities contained in the swim lane.

- Horizontal synchronization.
- Vertical synchronization.

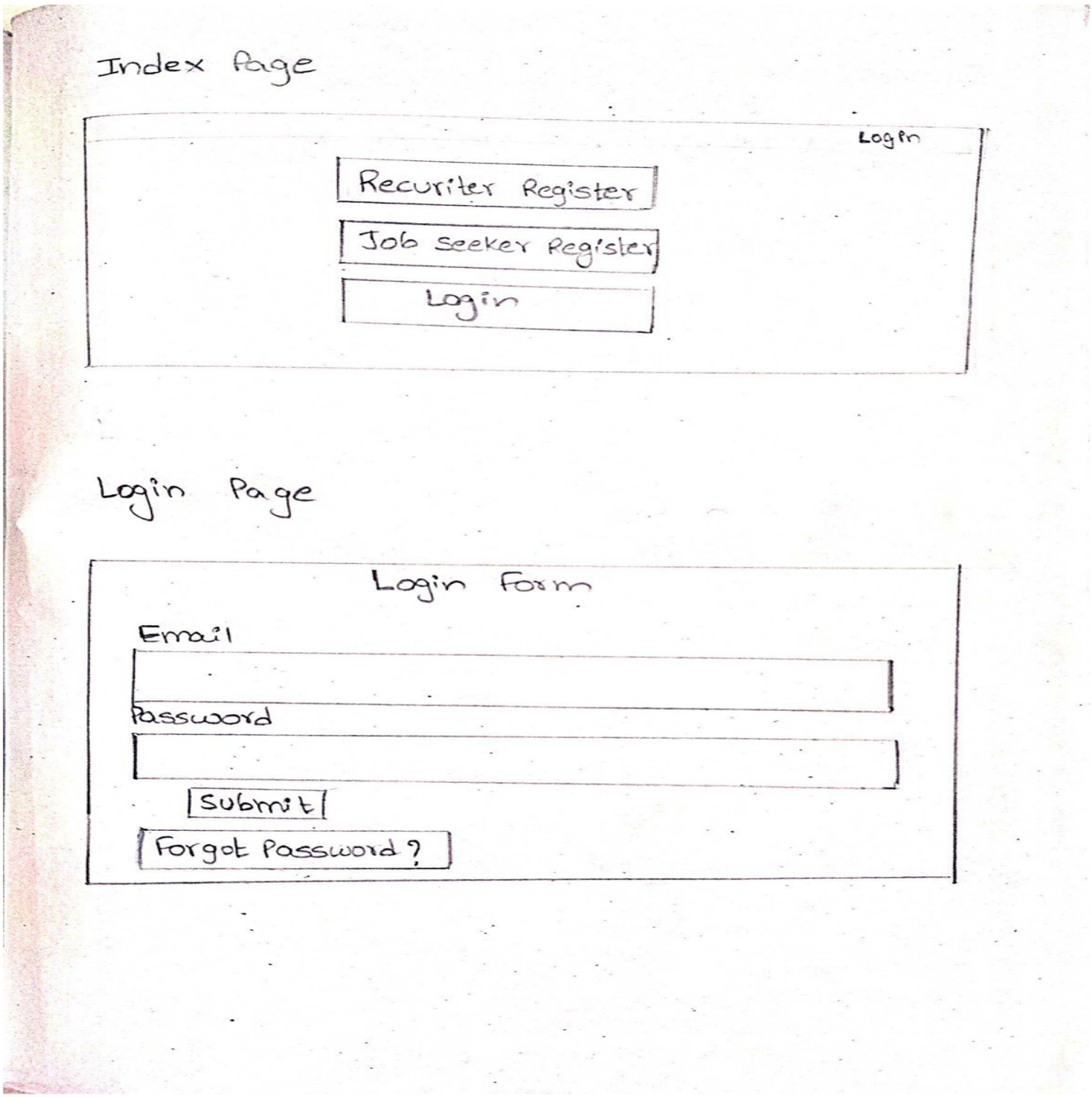
Activity Diagram for Job Seeker



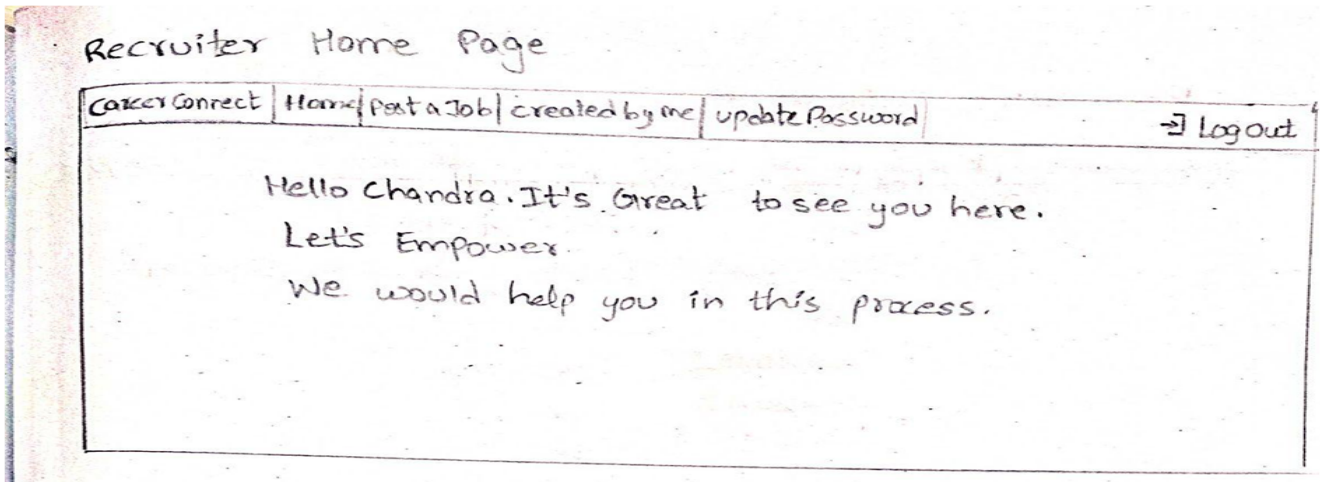
Activity Diagram for Recruiter



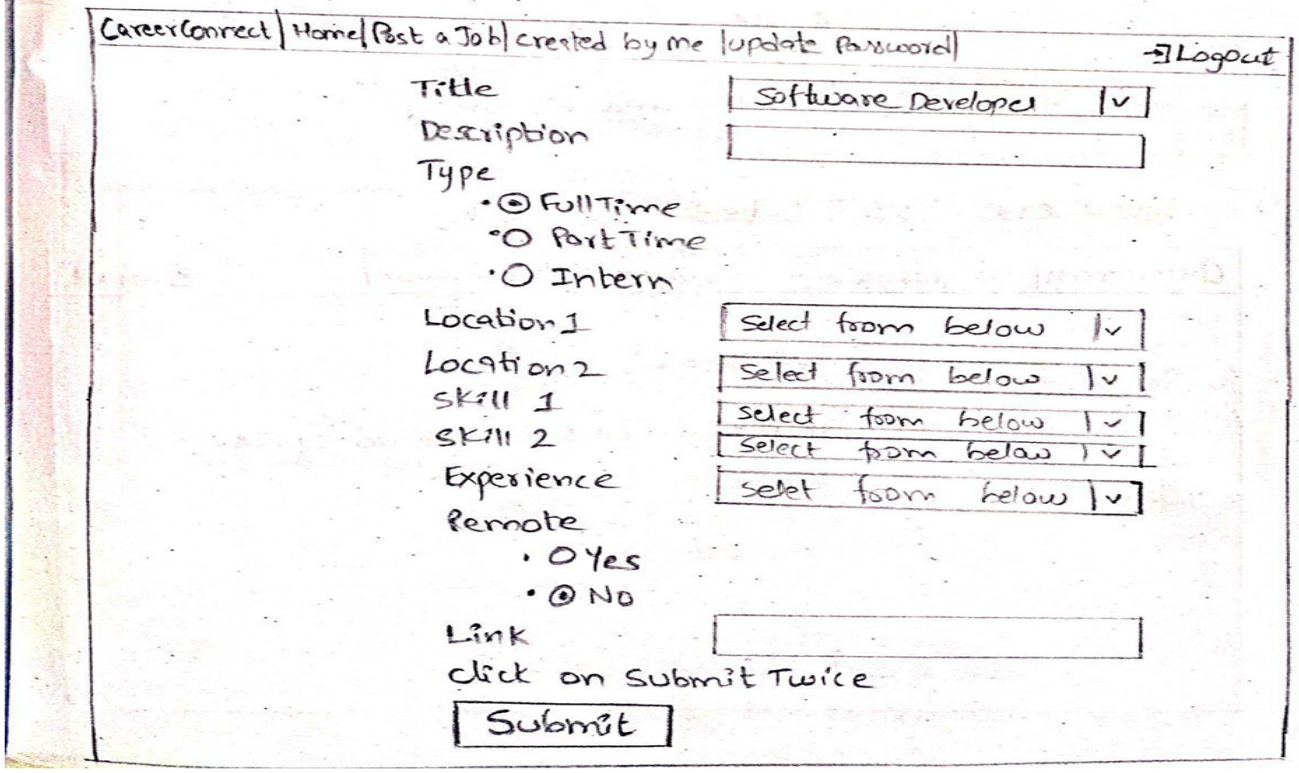
Construction of prototypes



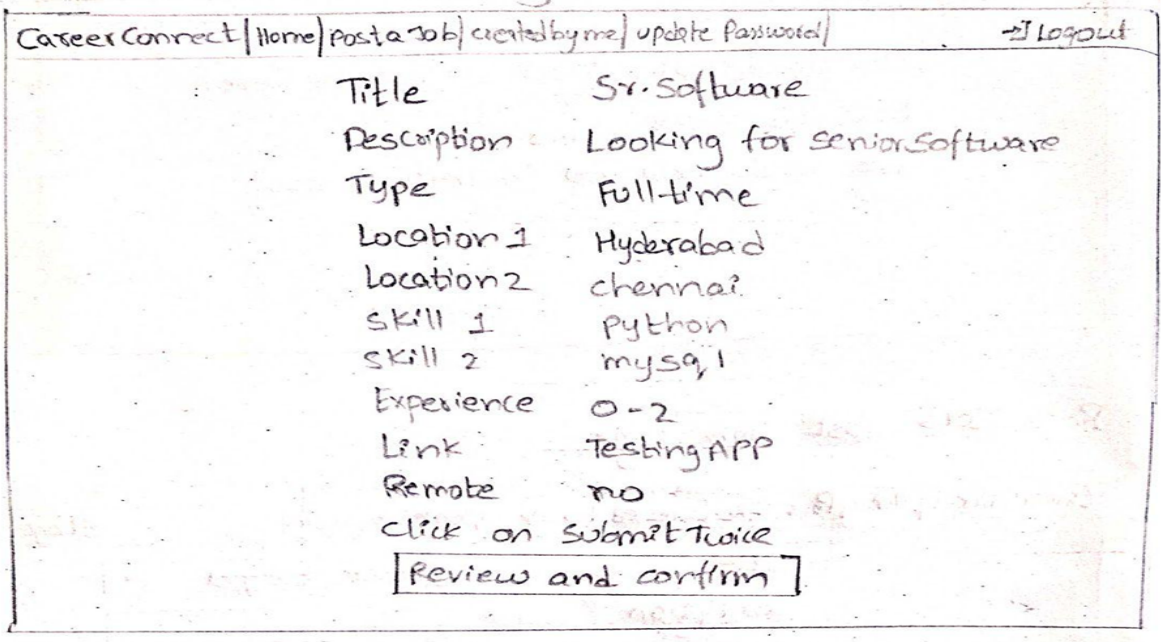
Recruiter Home Page



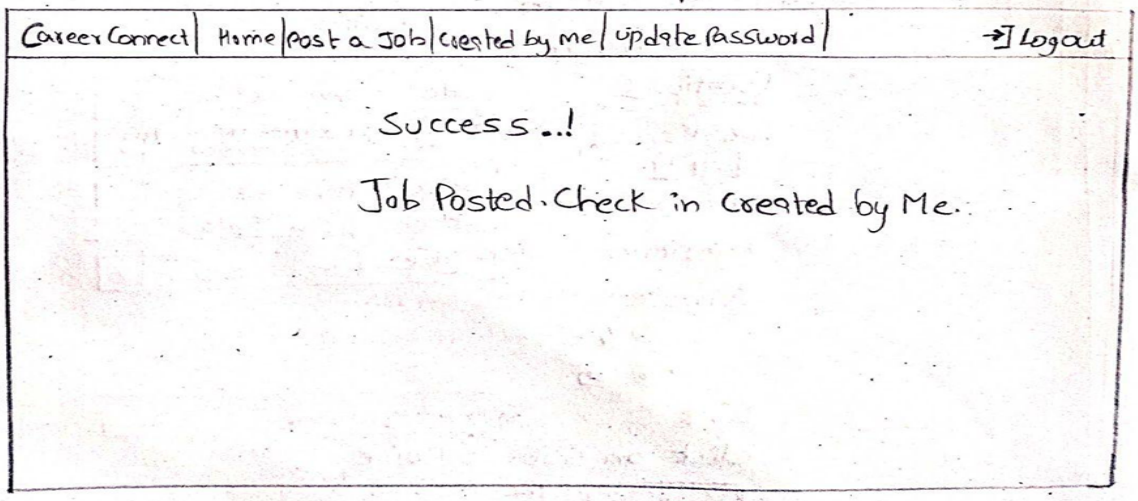
Recruiter's Post Job Page



Recruiter's Job Posting Confirm



Recruiters Job Posted Succesful



Recruiter's Created by Me Page

CareerConnect | Home | Post a Job | created by Me | update Password | → Logout

Job ID: 1
Title: Python Developer

Job ID: 2
Title: Network Engineer

JobID: 3
Title: Software Developer

JobID: 4
Title: Network Engineer

JobID: 5
Title: Software Developer

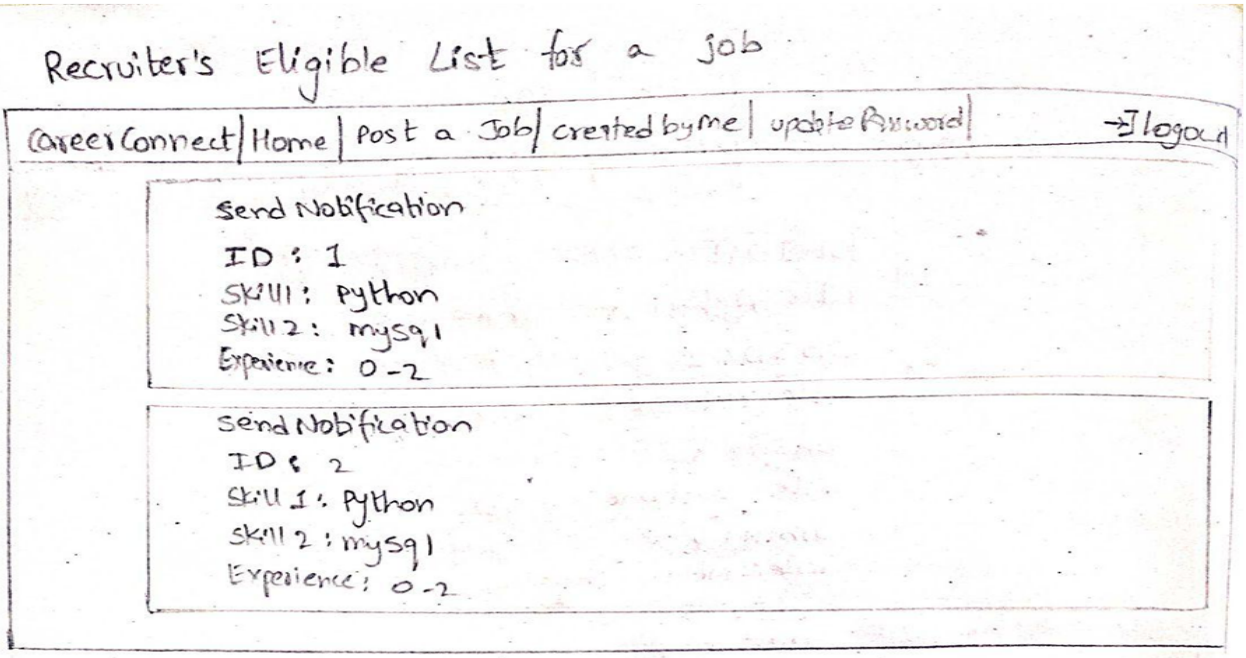
JobID: 6
Title: Sr. Software

Recruiter's Created by Me (with a jobid)

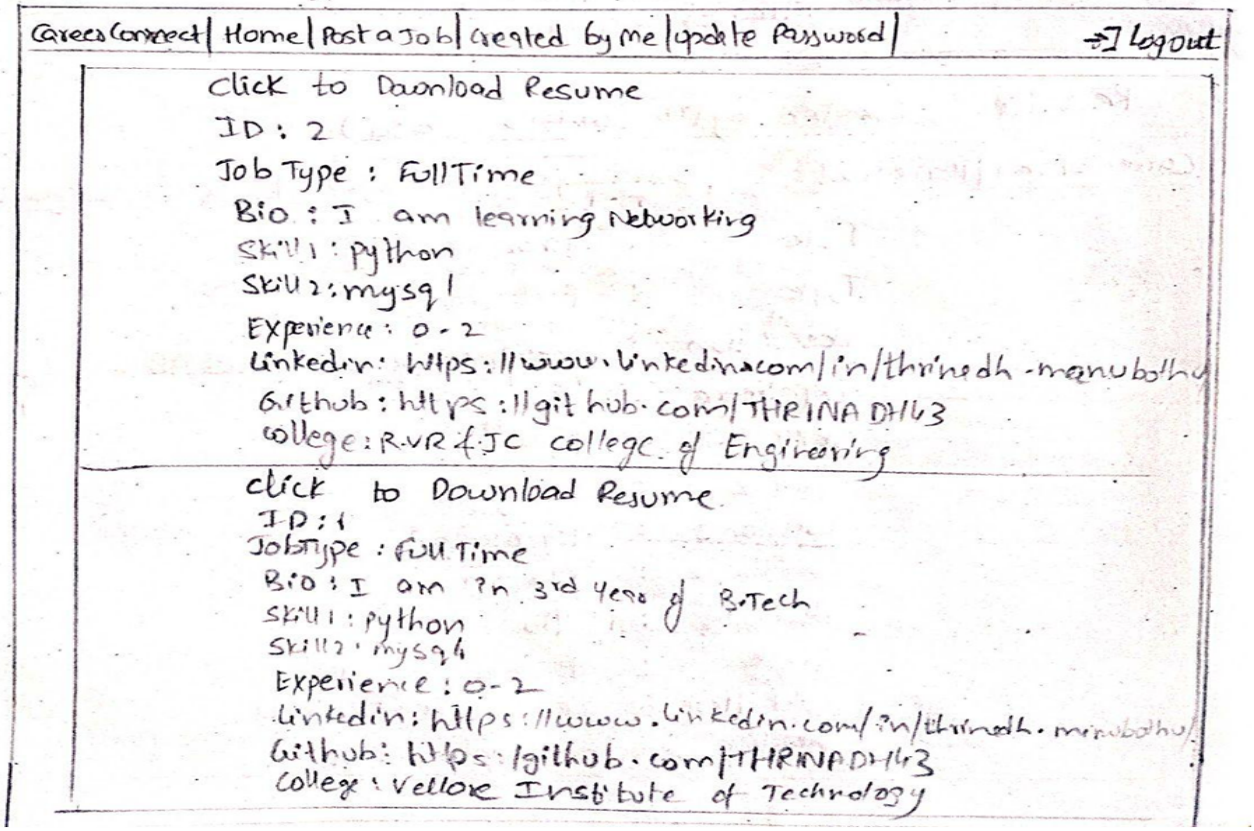
CareerConnect | Home | Post a Job | Created by me | update Password | → Logout

Title : Python Developer
Type : Full Time
Description : Looking for experienced Python Developer
Experience : 8+
Skill 1 : Python
Skill 2 : mongodb
Location 1 : Hyderabad
Location 2 : Chennai
Remote : No
Link : Thrinadh Manubothu
click Here to see eligible Candidates
Click Here to see Applied List

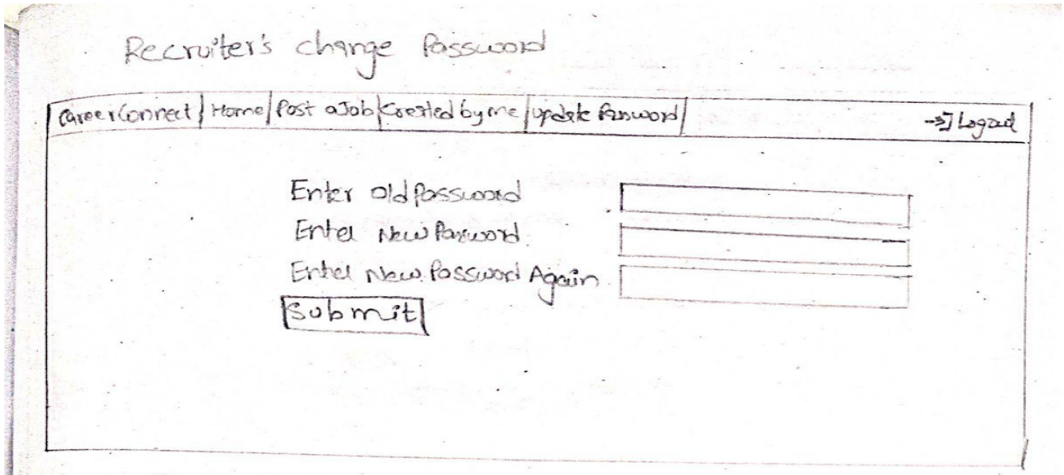
Recruiter's Eligible List for a Job



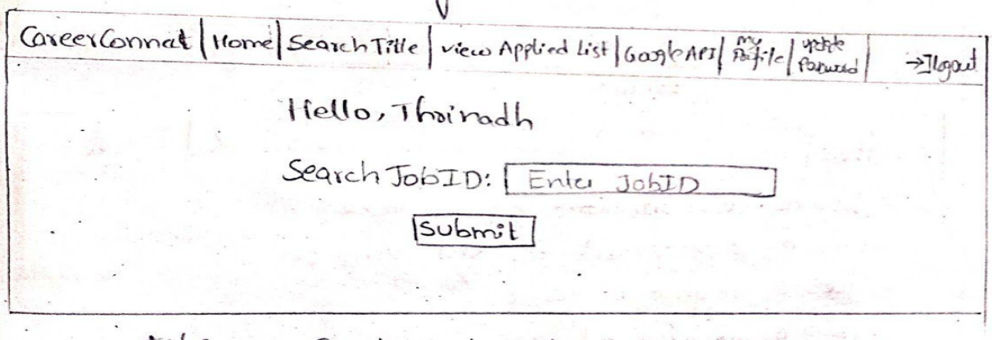
Recruiter's Applied List for a Job



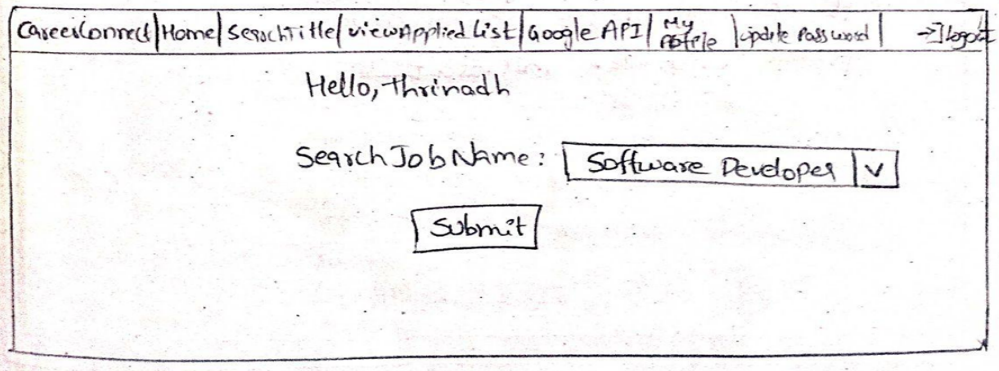
Recruiter's change password



JobSeeker Home Page



JobSeeker Search with title



JobSeeker Applied List

CareerConnect | Home | SearchTitle | viewAppliedList | Google API | MyProfile | update Password | Logout

JobId : 5
Title : Software Developer
Type : Full Time
Experience : 0 - 2
Skill : Python
Created by : chandra

JbSeeker Google API

CareerConnect | Home | SearchTitle/viewAppliedList/GoogleAPI | MyProfile | update Password | Logout

Softwa | X | a

software-development-engineer-alexa
Software-development-manager-amazon
software engineer google
software development engineer fintech
software development-engineer-aws
software engineer microsoft

JobSeeker View Profile

Careers Connect | Home | Search Title | View Applied List | Google API | my Profile | Update Password | → Logout

Hello, Thrinadh

Bio : I am learning Networking

Jobtype : Full Time

Experience : 0-2

Skill 1 : python

Skill 2 : mysql

Location 1 : Hyderabad

Location 2 : chennai

Remote : NO

Github : https://github.com/THRINADH43

Linkedin : https://www.linkedin.com/in/thrinadh-manubothu/

Click Here to download Resume click Here to update profile

Jobseeker update Profile

Careers Connect | Home | search Title | view Applied List | Google API | My Profile | update password | → Logout

Type

- ◉ Full Time
- ○ Part Time
- ○ Intern

Description

GitHub Link

Linkedin Link

College

Skill 1 — select from below

Skill 2 — select from below

Location 1 — Select from below

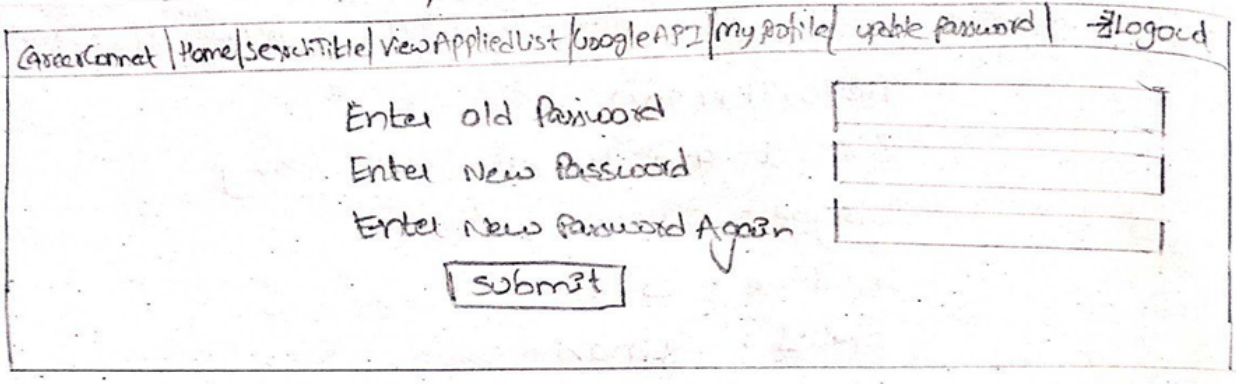
Location 2 — Select from below

I am a — select from below

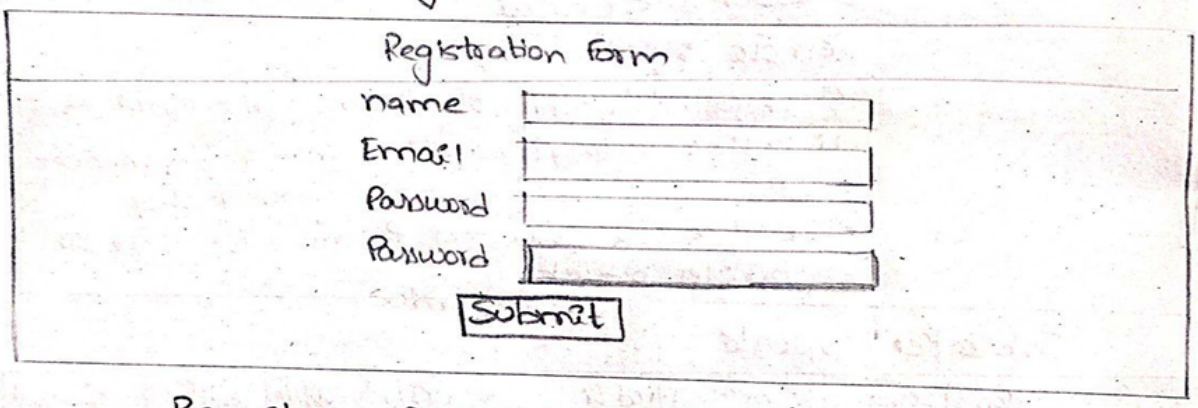
Upload resume PDF — Choose File — No file chosen

Submit

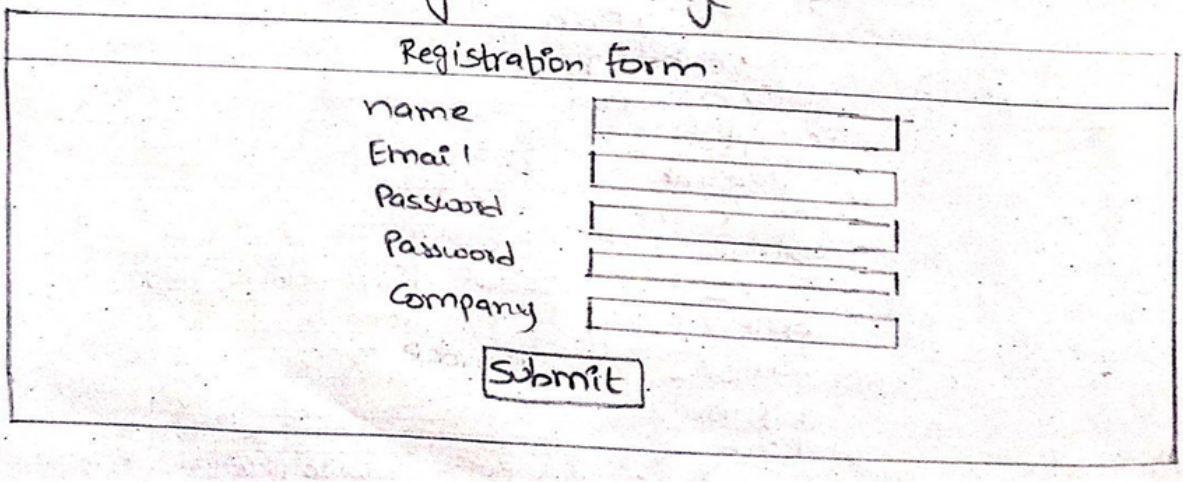
Job Seeker Update Password



Job Seeker Registration Page



Recruiter Registration Page



9. CONSTRUCTION OF SEQUENCE DIAGRAMS.

A sequence diagram is a graphical view of a scenario that shows object interaction in a time based sequence--what happens first, what happens next… Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

A sequence diagram has two dimensions: the vertical dimension represents time; the horizontal dimension represents different objects. The vertical line is called the object's lifeline. The lifeline represents the object's existence during the interaction.

Steps:

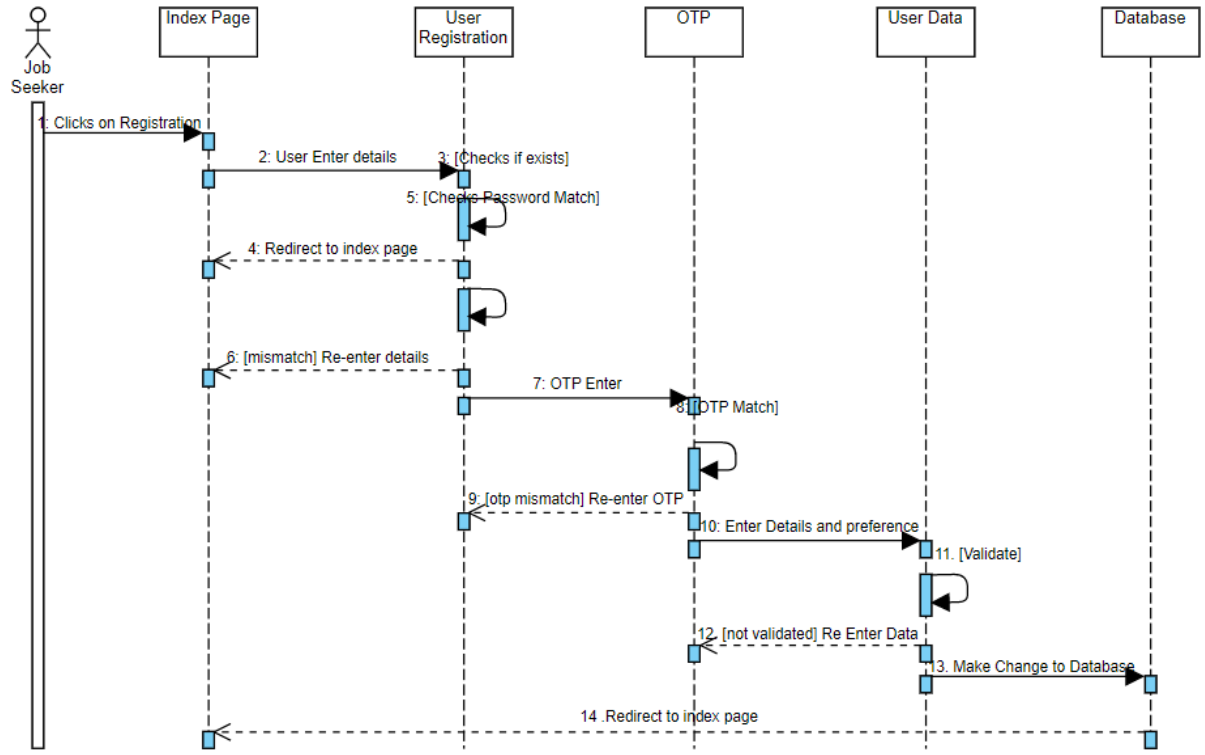
1. An object is shown as a box at the top of a dashed vertical line. Object names can be specific (e.g., Algebra 101, Section 1) or they can be general (e.g., a course offering). Often, an anonymous object (class name may be used to represent any object in the class.)
2. Each message is represented by an Arrow between the lifelines of two objects. The order in which these messages occur is shown top to bottom on the page. Each message is labeled with the message name.

There are two main differences between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other. A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects.

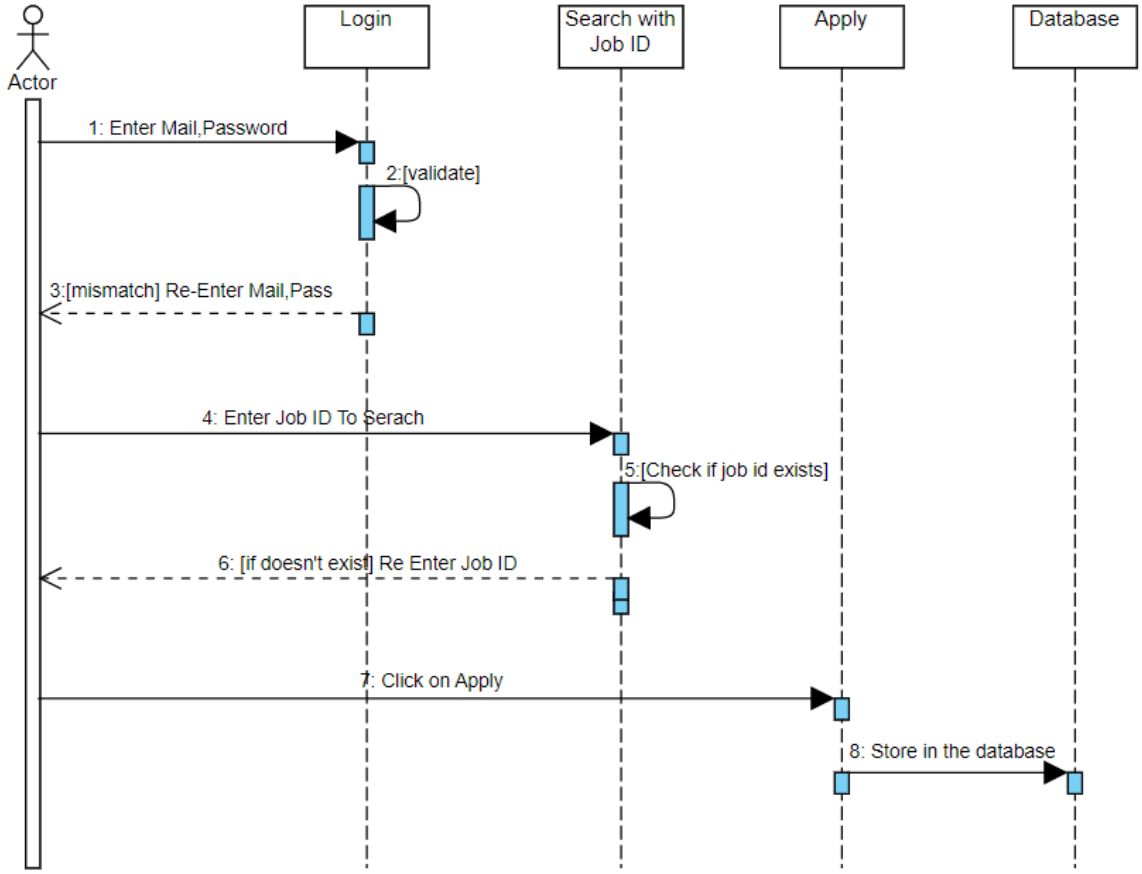
ELEMENTS OF SEQUENCE DIAGRAM:

- Objects
- Links
- Messages
- Focus of control
- Object lifeline

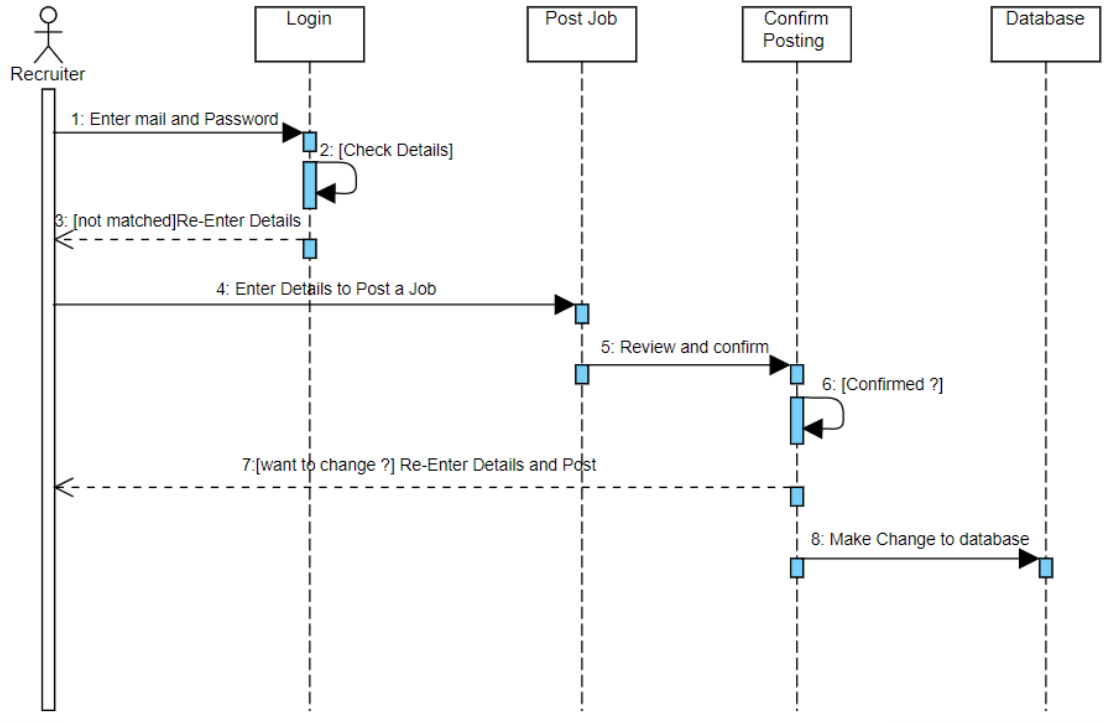
Sequence Diagram for Job Seeker Registration



Sequence for Job Seeker Applying to Job

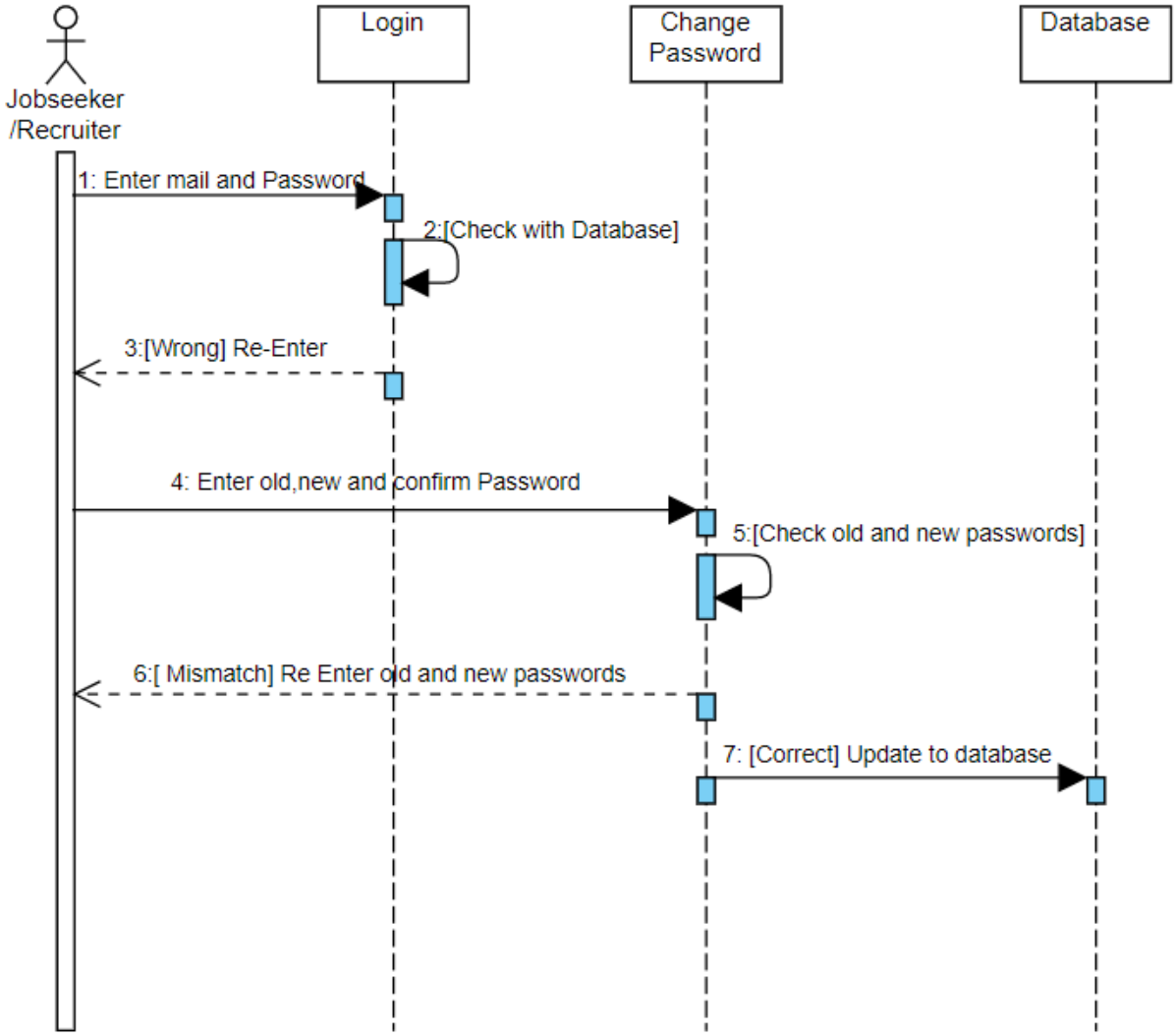


Sequence Diagram for Recruiter Posting a Job



Sequence Diagram for Changing Password

[Applies to Both JobSeeker and Recruiter]



10. CONSTRUCTION OF COLLABORATION DIAGRAMS

Collaboration diagrams are the second kind of interaction diagram in the UML diagrams. They are used to represent the collaboration that realizes a use case. The most significant difference between the two types of interaction diagram is that a collaboration diagram explicitly shows the links between the objects that participate in a collaboration , as in sequence diagrams, there is no explicit time dimension.

Message labels in collaboration diagrams:

Messages on a collaboration diagram are represented by a set of symbols that are the same as those used in a sequence diagram, but with some additional elements to show sequencing and recurrence as these cannot be inferred from the structure of the diagram. Each message label includes the message signature and also a sequence number that reflects call nesting, iteration, branching, concurrency and synchronization within the interaction.

The formal message label syntax is as follows: [predecessor] [guard-condition] sequence-expression [return-value ':='] message-name' (' [argument-list] ')'.

A *predecessor* is a list of sequence numbers of the messages that must occur before the current message can be enabled. This permits the detailed specification of branching pathways. The message with the immediately preceding sequence number is assumed to be the predecessor by default, so if an interaction has no alternative pathways the predecessor list may be omitted without any ambiguity.

The syntax for a predecessor is as follows: sequence-number

{ ',' sequence-number} 'l'.

The 'l' at the end of this expression indicates the end of the list and is only included when an explicit predecessor is shown.

Guard conditions are written in Object Constraint Language (OCL) ,and are only shown where the enabling of a message is subject to the defined condition. A guard condition may be used to represent the synchronization of different threads of control.

A sequence-expression is a list of integers separated by dots ('.') optionally followed by a name (a single letter), optionally followed by a recurrence term and terminated by a colon. A sequence-expression has the following syntax:

integer { '.' integer } [name] [recurrence] ':'

In this expression integer represents the sequential order of the message. This may be nested within a loop or a branch construct, so that, for example, message 5.1 occurs after message 5.2 and both are contained within the activation of message 5.

The name of a sequence-expression is used to differentiate two concurrent messages since these are given the same sequence number. For example, messages 3.2.1a and 3.2.1b are concurrent within the activation of message 3.2.

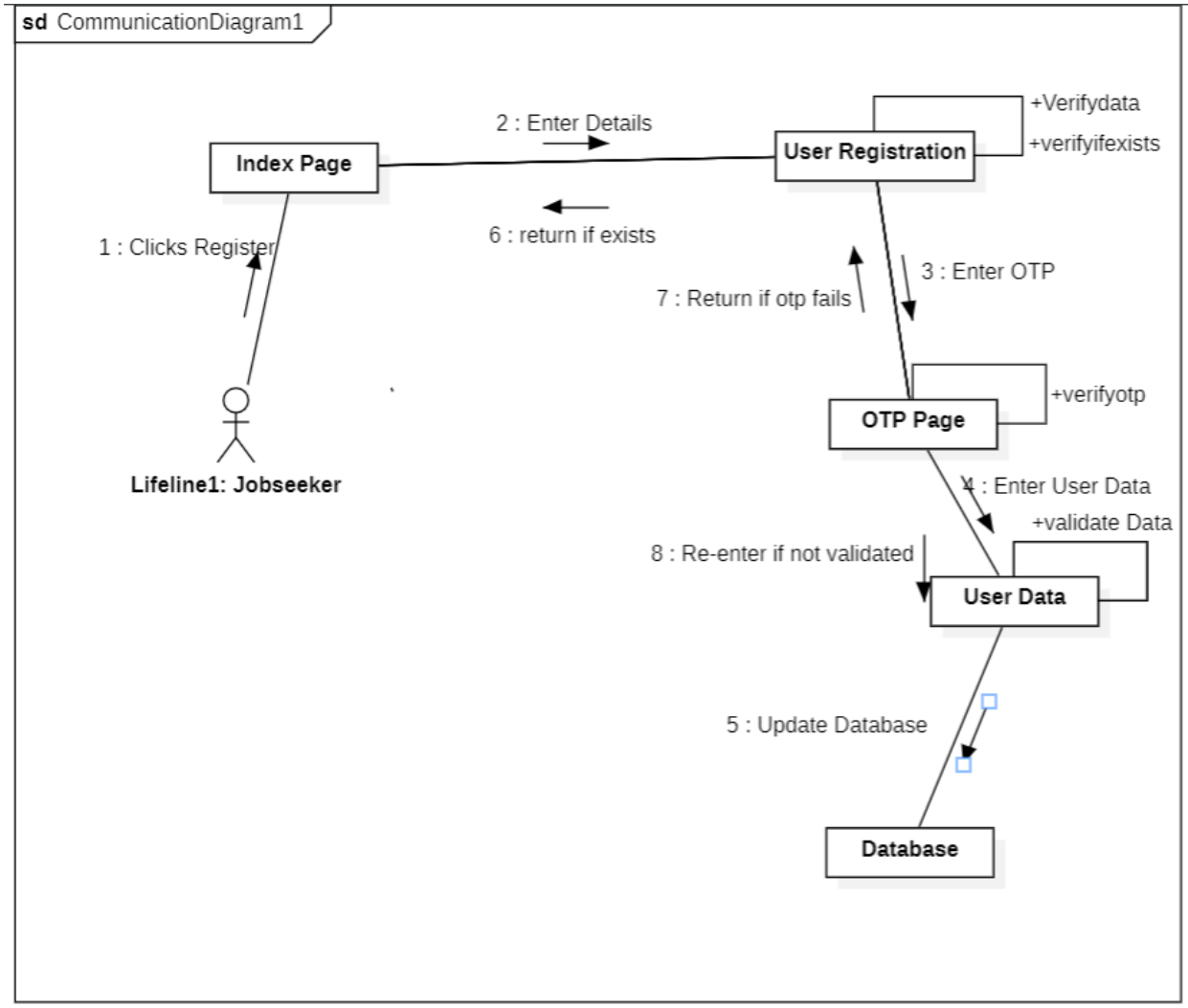
Recurrence reflects either iterative or conditional execution and its syntax is as follows:

Branching: '['condition-clause‘]

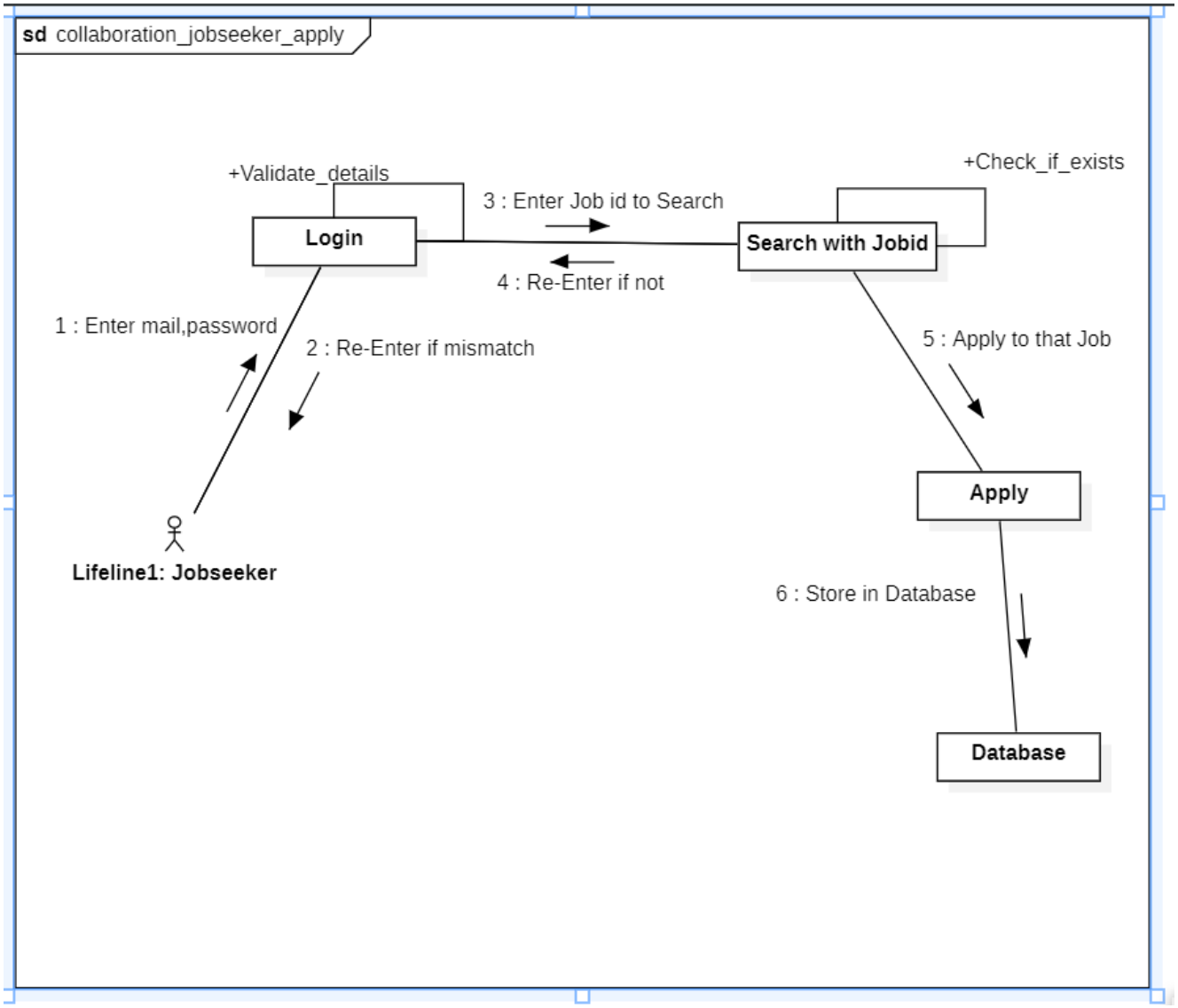
Iteration: ‘ * ‘ ‘ [‘ iteration-clause ‘] '

Elements: • Objects, Messages, Path, Sequence Numbers, Links

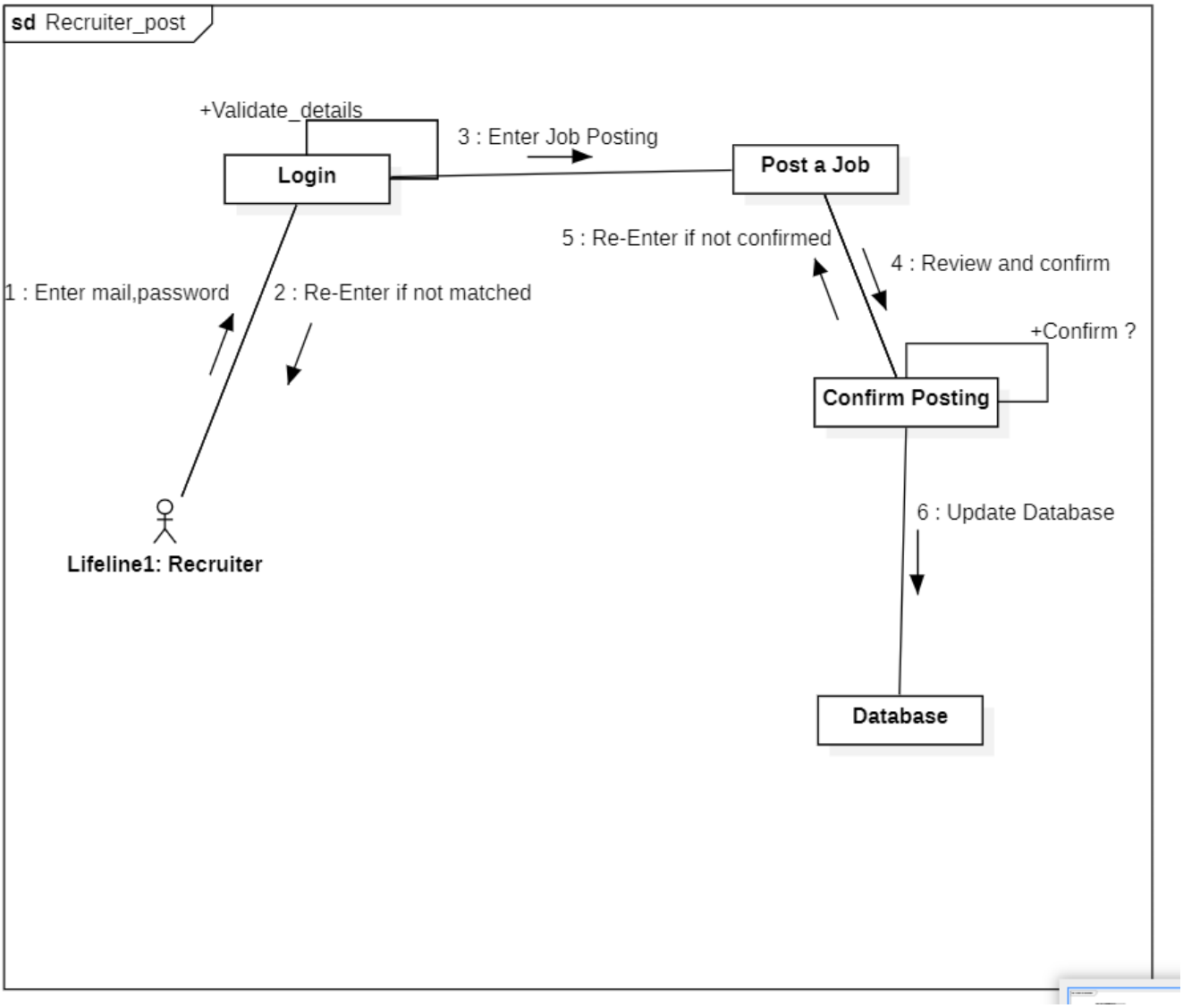
Collaboration diagram for Jobseeker Register:



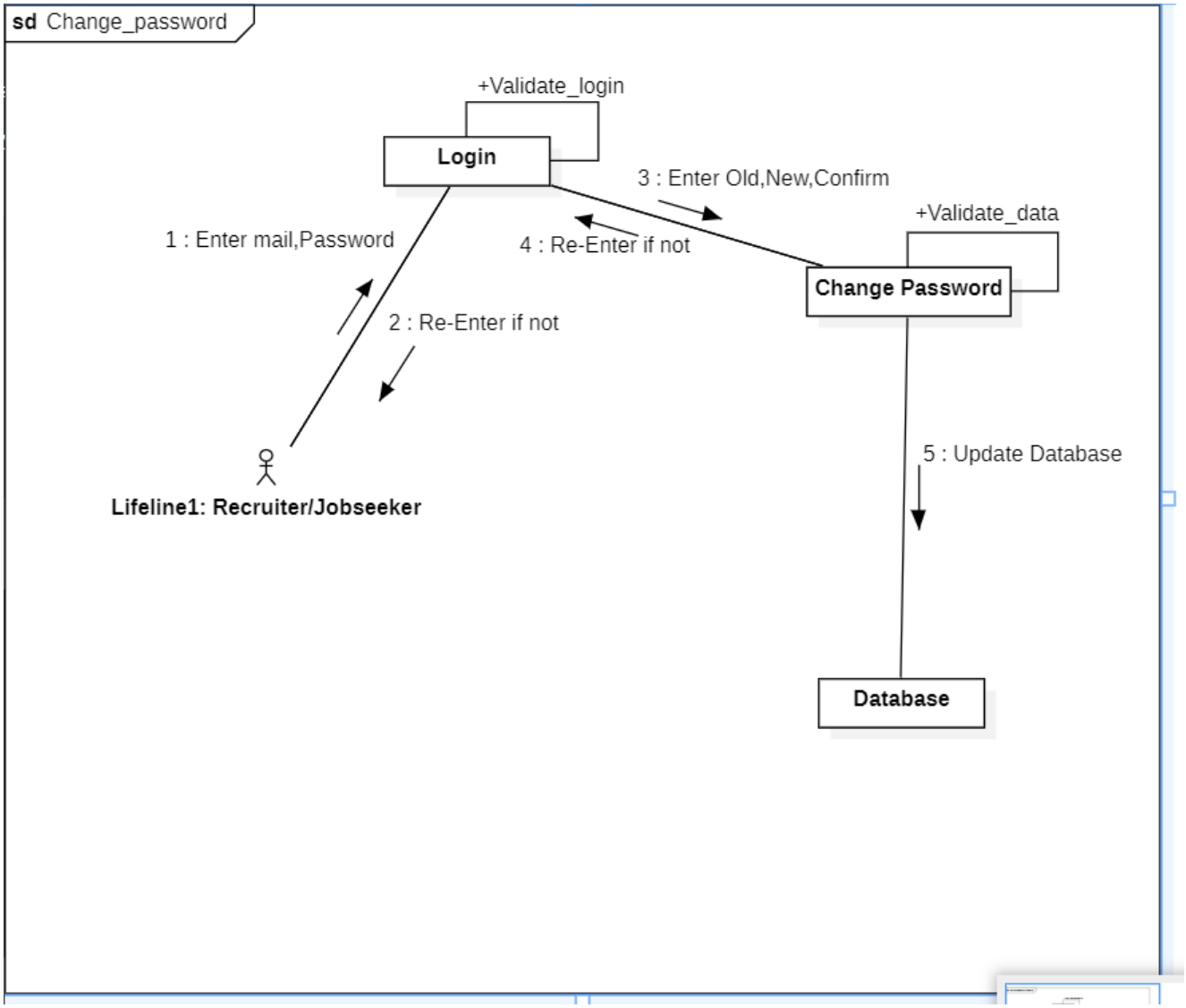
Collaboration Diagram for Job Seeker Apply.



Collaboration Diagram for Recruiter Posting a Job:



Collaboration Diagram for Change Password



11. Construction of UML static class diagram

Class diagrams contain icons representing classes, packages, interfaces, and their relationships. You can create one or more class diagrams to depict the classes at the top level of the current model; such class diagrams are themselves contained by the top level of the current model.

Class: A Class a description of a group of objects with common properties (attributes), common behavior (operations), common relationships to other objects, and common semantics.

Thus, a class is a template to create objects. Each object is an instance of some class and objects cannot be instances of more than one class. Classes should be named using the vocabulary of the domain. For example, the CourseOffering class may be defined with the following characteristics:

Attributes - location, time offered.

Operations - retrieve location, retrieve time of day, add a student to the offering.

Each object would have a value for the attributes and access to the operations specified by the CourseOffering class. In the UML, classes are represented as compartmentalized rectangles. The top compartment contains the name of the class. The middle compartment contains the structure of the class (attributes). The bottom compartment contains the behavior of the class (operations) as shown below.

OBJECT :

• AN OBJECT IS a representation of an entity, either real-world or conceptual.

• An object is a concept, abstraction, or thing with well defined boundaries and meaning for an application.

• Each object in a system has three characteristics: state, behavior, and identity.

STATE : THE STATE OF an object is one of the possible conditions in which it may exist. The state of an object typically changes over time, and is defined by a set of properties (called attributes), with the values of the properties, plus the relationships the object may have with other objects.

For example, a course offering object in the registration system may be in one of two states: open and closed. It is available in the open state if value is < 10 otherwise closed.

Behavior :

• Behavior determines how an object responds to requests from other objects .

• Behavior is implemented by the set of operations for the object.

Identity :

• Identity means that each object is unique even if its state is identical to that of another object.

Attributes :

Attributes are part of the essential description of a class. They belong to the class, unlike objects, which instantiate the class. Attributes are the common structure of what a member of the class can 'know'. Each object will have its own, possibly unique, value for each attribute.

Guidelines for identifying attributes of classes are as follows:

- ➢ Attributes usually correspond to nouns followed by prepositional phrases
- ➢ Keep the class simple; state only enough attribute to defineobject state.
- ➢ Attributes are less likely to be fully described in the problem statement.
- ➢ Omit derived attributes.
- ➢ Do not carry discovery attributes to excess.

STEREOTYPES AND CLASSES :

As like stereotypes for relationships in use case diagrams. Classes can also have stereotypes. Here a stereotype provides the capability to create a new kind of modeling element. Here, we can create new kinds of classes. Some common stereotypes for a class are entity Class, boundary Class, control class, and exception.

Entity Classes : An entity class models information and associated behavior that is generally long lived. This type of class may reflect a real-world entity or it may be needed to perform tasks internal to the system. They are typically independent of their surroundings; that is, they are not sensitive to how the surroundings communicate with the system.

Boundary Classes : Boundary classes handle the communication between the system surroundings and the inside of the system. They can provide the interface to a user or another system (i.e., the interface to an actor). They constitute the surroundings dependent part of the system.

Boundary classes are used to model the system interfaces. Boundary classes are also added to facilitate communication with other systems. During design phase, these classes are refined to take into consideration the chosen communication protocols.

Control Classes

• Control classes model sequencing behavior specific to one or more use cases.

• Control classes coordinate the events needed to realize the behavior specified in the use case.

• Control classes typically are application-dependent classes. In the early stages of the Elaboration Phase, a control class is added for each actor/use case pair. The control class is responsible for the flow of events in the use case.

In the early stages of the Elaboration Phase, a control class is added for each actor/use case pair. The control class is responsible for the flow of events in the use case.

NEED FOR RELATIONSHIPS AMONG CLASSES:

All systems are made up of many classes and objects. System behaviour is achieved through the collaborations of the objects in the system. Two types of relationships in CLASS diagram are:

1. Association Relationship
2. Aggregation Relationship

1. Association Relationship:

An association is a bidirectional semantic connection between classes. It is not a data flow as defined in structured analysis and design data may flow in either direction across the association. An association between classes means that there is a link between objects in the associated classes.

2. Aggregation Relationship:

An aggregation relationship is a specialized form of association in which a whole is related to its part(s). Aggregation is known as a “part-of” or containment relationship. The UML notation for an aggregation relationship is an association with a diamond next to the class denoting the aggregate(whole).

3. Super-sub structure (Generalization Hierarchy):

These allow objects to be build from other objects. The super-sub class hierarchy is a relationship between classes, where one class is the parent class of another class.

NAMING RELATIONSHIP:

An association may be named. Usually the name is an active verb or verb phrase that communicates the meaning of the relationship. Since the verb phrase typically implies a reading direction, it is desirable to name the association so it reads correctly from left to right or top to bottom. The words may have to be changed to read the association in the other direction (e.g., Buses are allotted to Routes). It is important to note that the name of the association is optional.

ROLE NAMES:

The end of an association where it connects to a class is called an association role. Role names can be used instead of association names. A role name is a noun that denotes how one class associates with another. The role name is placed on the association near the class that it modifies, and may be placed on one or both ends of an association line.

- Associations are named or role names are used only when the names are needed for clarity.

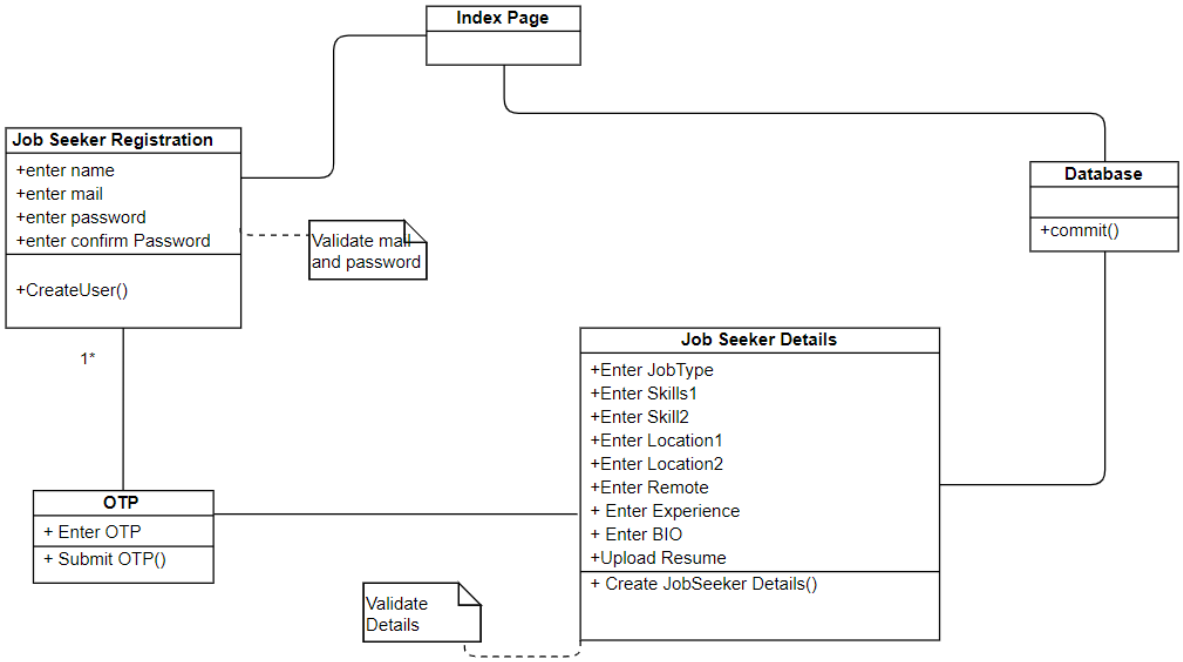
- It is not necessary to have both a role name and an association name.

MULTIPLICITY INDICATORS:

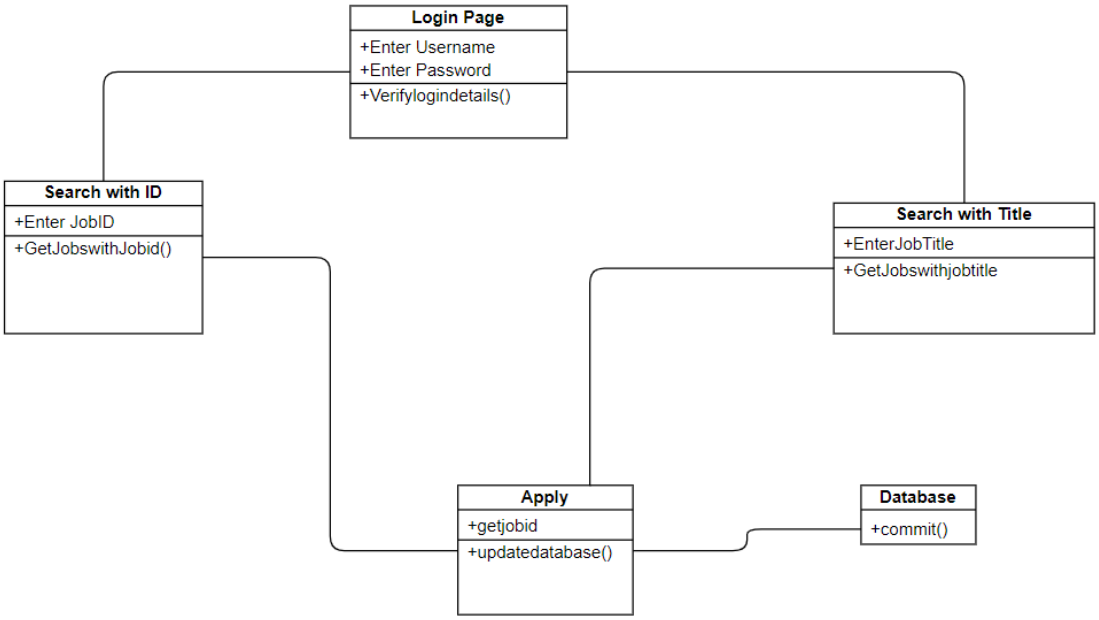
Although multiplicity is specified for classes, it defines the number of objects that participate in a relationship. Multiplicity defines the number of objects that are linked to one another. There are two multiplicity indicators for each association or aggregation one at each end of the line. Some common multiplicity indicators are

1	Exactly one
0... *	Zero or more
1... *	One or more
0... 1	Zero or one
5... 8	Specific range (5, 6, 7, or 8)
4... 7, 9	Combination (4, 5, 6, 7, or 9)

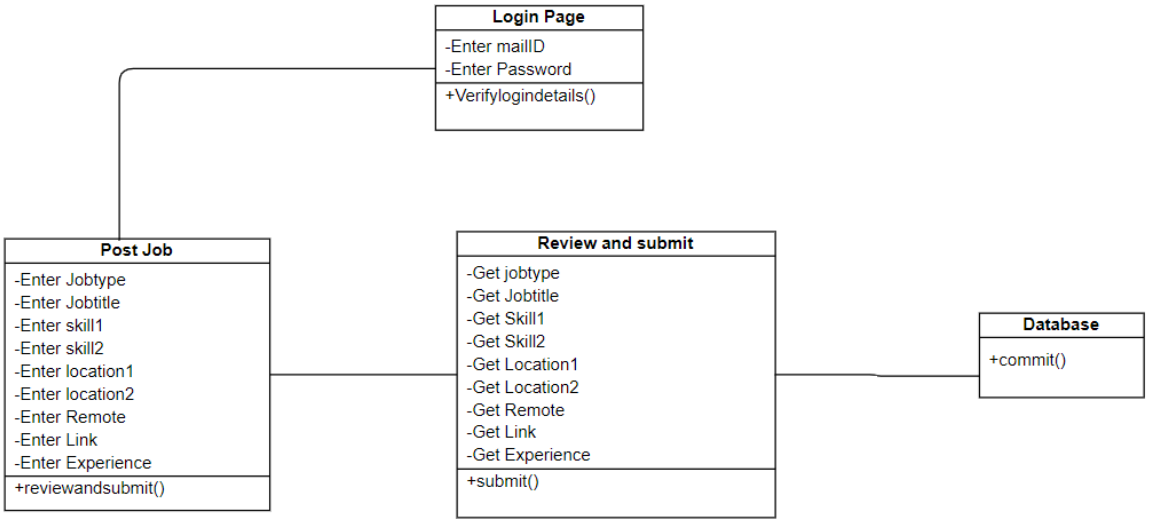
Class Diagram for Job Seeker Registration



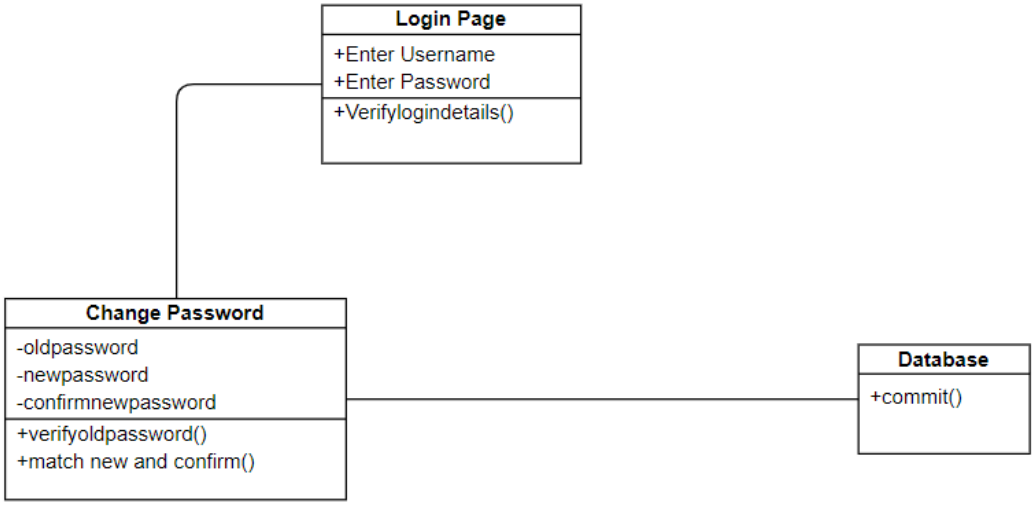
Class Diagram for Job Seeker Applying to Job



Class Diagram for Recruiter Posting Job



Class Diagram for both Recruiter and Jobseeker:



12. Analyzing the object behavior by constructing the UML State Chart diagram

Use cases and scenarios provide a way to describe system behavior; in the form of interaction between objects in the system. Sometimes it is necessary to consider inside behavior of an object.

A state chart diagram shows the **states** of a single object, the events or messages that cause a **transition** from one state to another, and the **actions** that result from a state change. As in Activity diagram , state chart diagram also contains special symbols for start state and stop state.

State chart diagram cannot be created for every class in the system , it is only for those class objects with significant behavior. State chart diagrams are closely related to activity diagrams. The main difference between the two diagrams is state chart diagrams are state centric, while activity diagrams are activity centric. A state chart diagram is typically used to model the discrete stages of an object's lifetime, whereas an activity diagram is better suited to model the sequence of activities in a process.

STATE:

A state represents a condition or situation during the life of an object during which it satisfies some condition, performs some action or waits for some event.

UML notation for STATE is

To identify the states for an object it's better to concentrate on the sequence diagram. In an ESU the object for Course Offering may have in the following states, initialization, open and closed state. These states are obtained from the attribute and links defined for the object. Each state also contains a compartment for actions.

Actions: Actions on states can occur at one of four times:

- on entry
- on exit
- do
- on event.

on entry: What type of action does that object has to perform after entering into the state.

on exit: What type of action does that object has to perform after exiting from the state.

Do: The task to be performed when the object is in this state, and must to continue until it leaves the state.

on event: An on event action is similar to a state transition label with the following syntax:

event(args)[condition]: the Action

State Transition:

A state transition indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied. A state transition is a relationship between two states, two activities, or between an activity and a state. You can show one or more state

transitions from a state as long as each transition is unique. Transitions originating from a state cannot have the same event unless there are conditions on the event.

Transitions are labeled with the following syntax:

Event (arguments) [condition] / action ^ target. send Event (arguments)

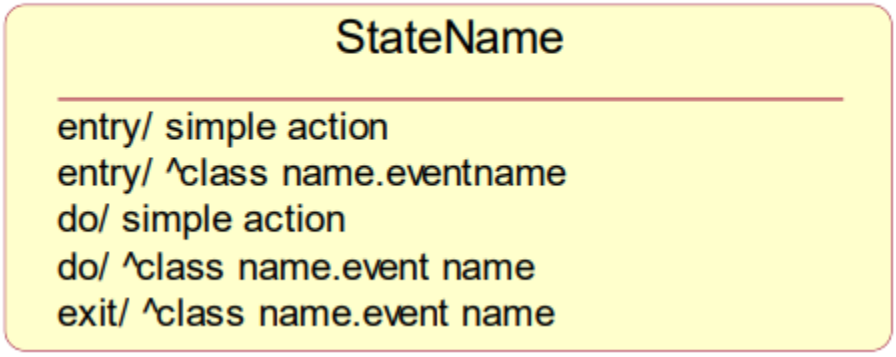
Only one event is allowed per transition, and one action per event.

State Details:

Actions that accompany all state transitions into a state may be placed as an entry action within the state. Wise that accompanies all state transitions out of a state may be placed as exit actions within the state. Behavior that occurs within the state is called an activity.

An activity starts when the state is entered and either completes or is interrupted by an outgoing state transition. The behavior may be a simple action or it may be an event sent to another object.

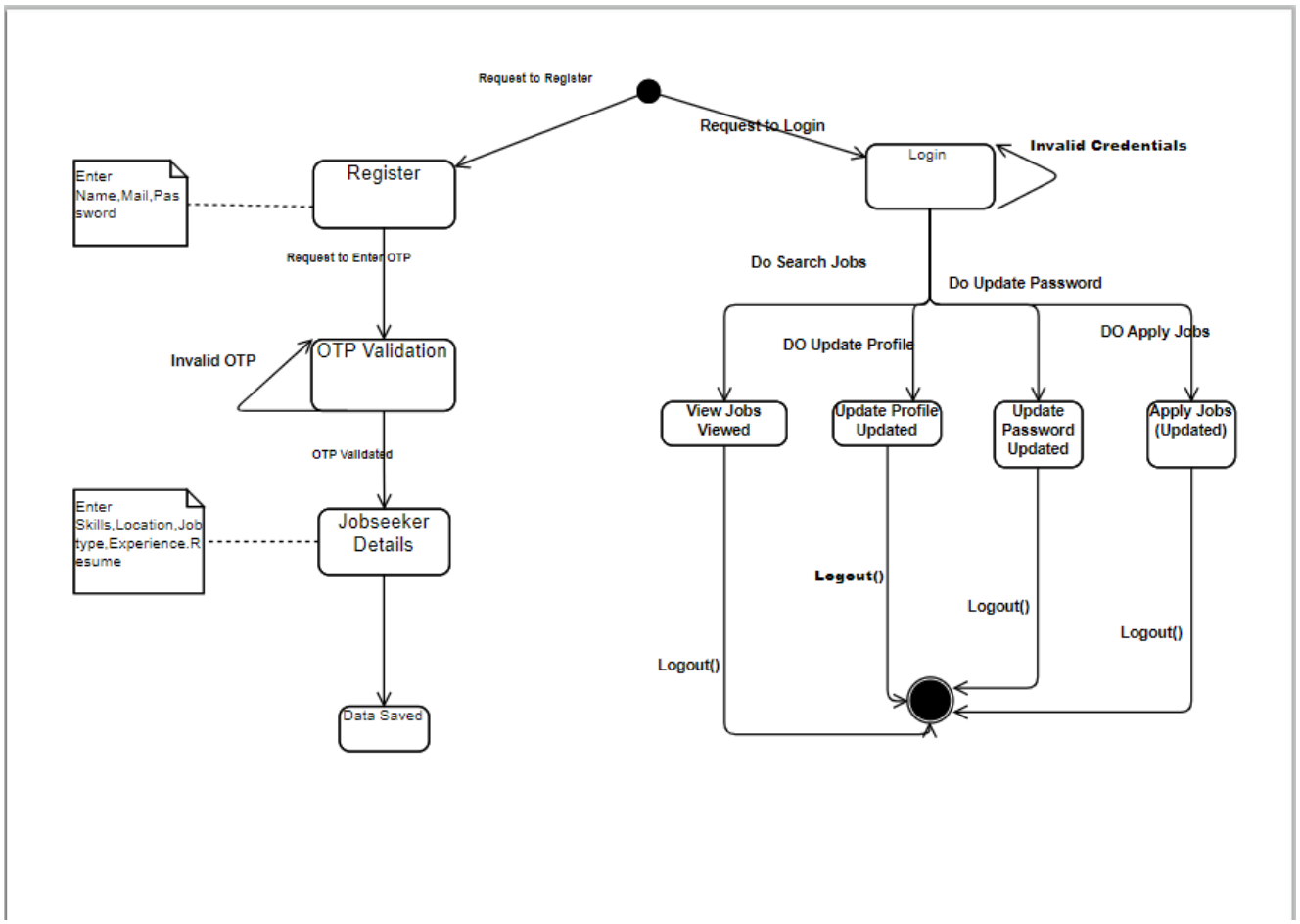
UML notation for State Details:



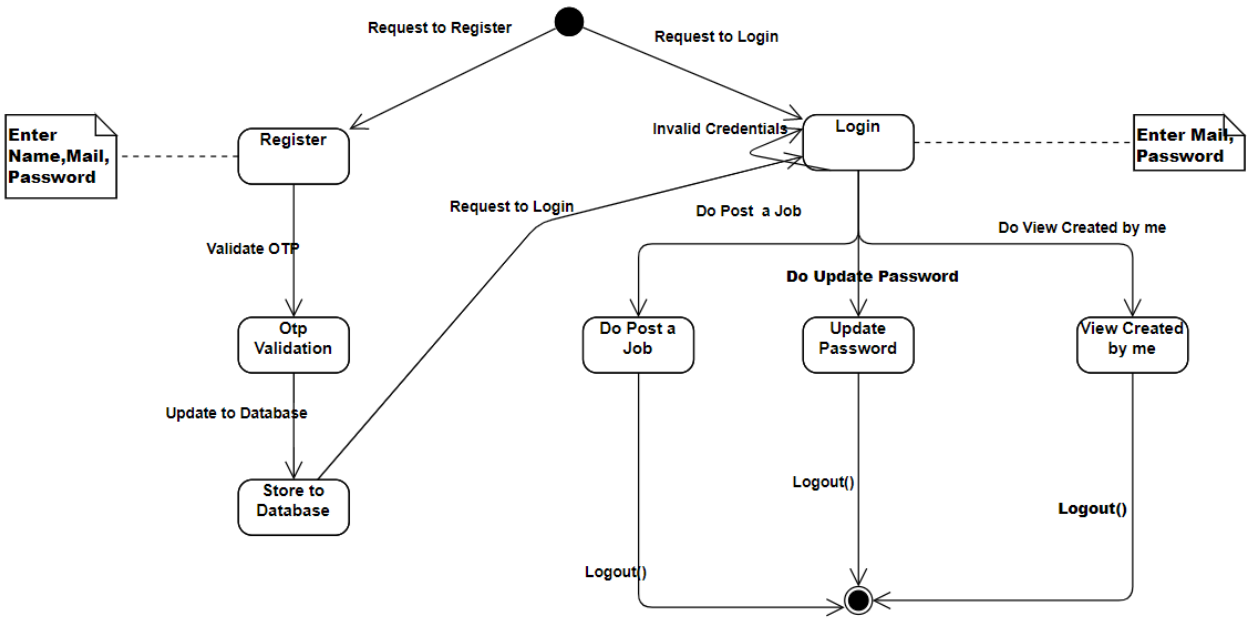
Purpose of State chart diagram:

• State chart diagrams are used to model the dynamic view of a system.
• State chart diagrams are used to model the lifetime of an object.
• State chart diagrams are used to focus on the changing state of a system driven by events.
• It will also be used when showing the behavior of a class over several use cases.

State Diagram for JobSeeker:



State Diagram for Recruiter:



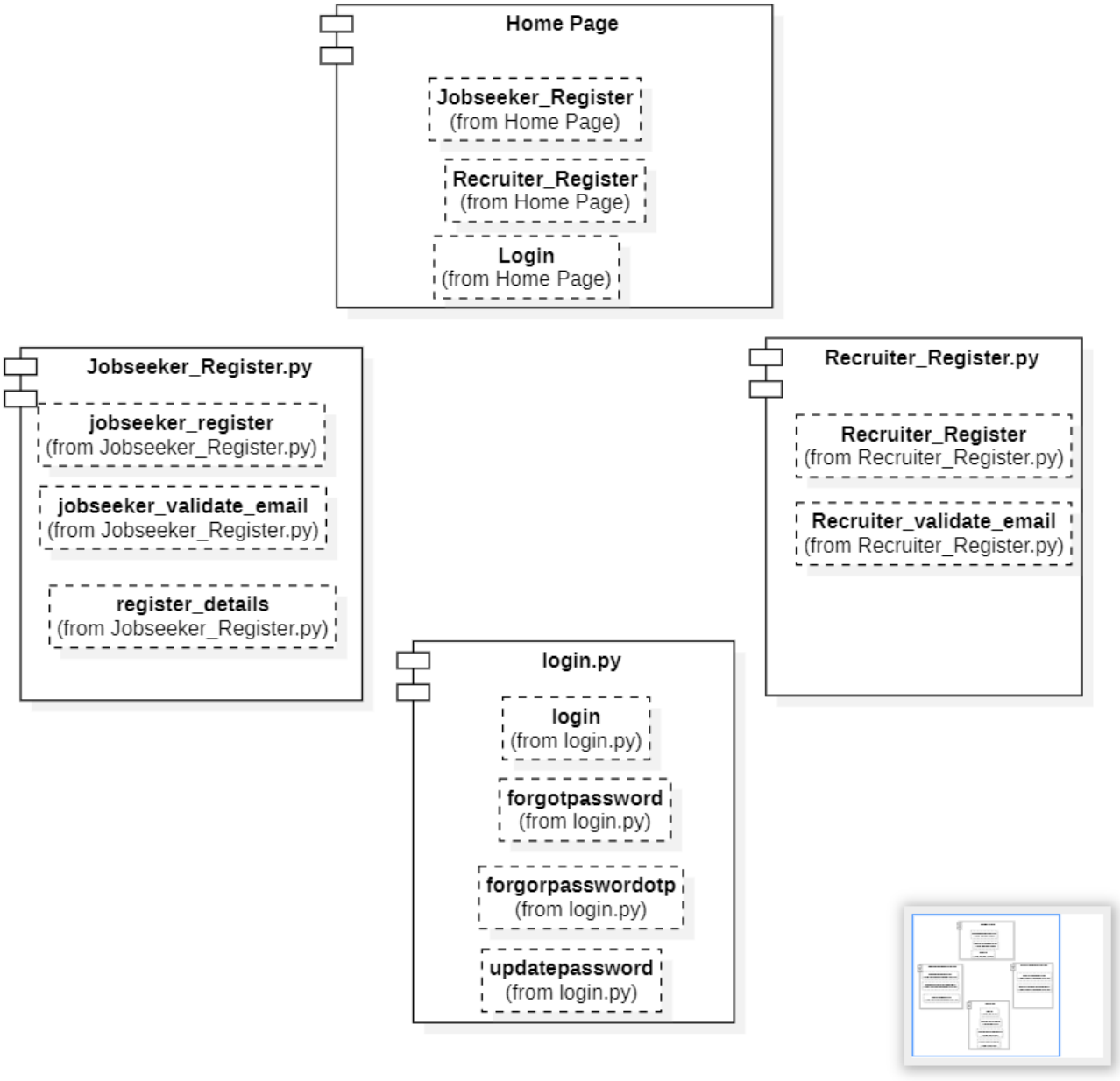
13. CONSTRUCTION OF IMPLEMENTATION DIAGRAMS

Component diagrams:

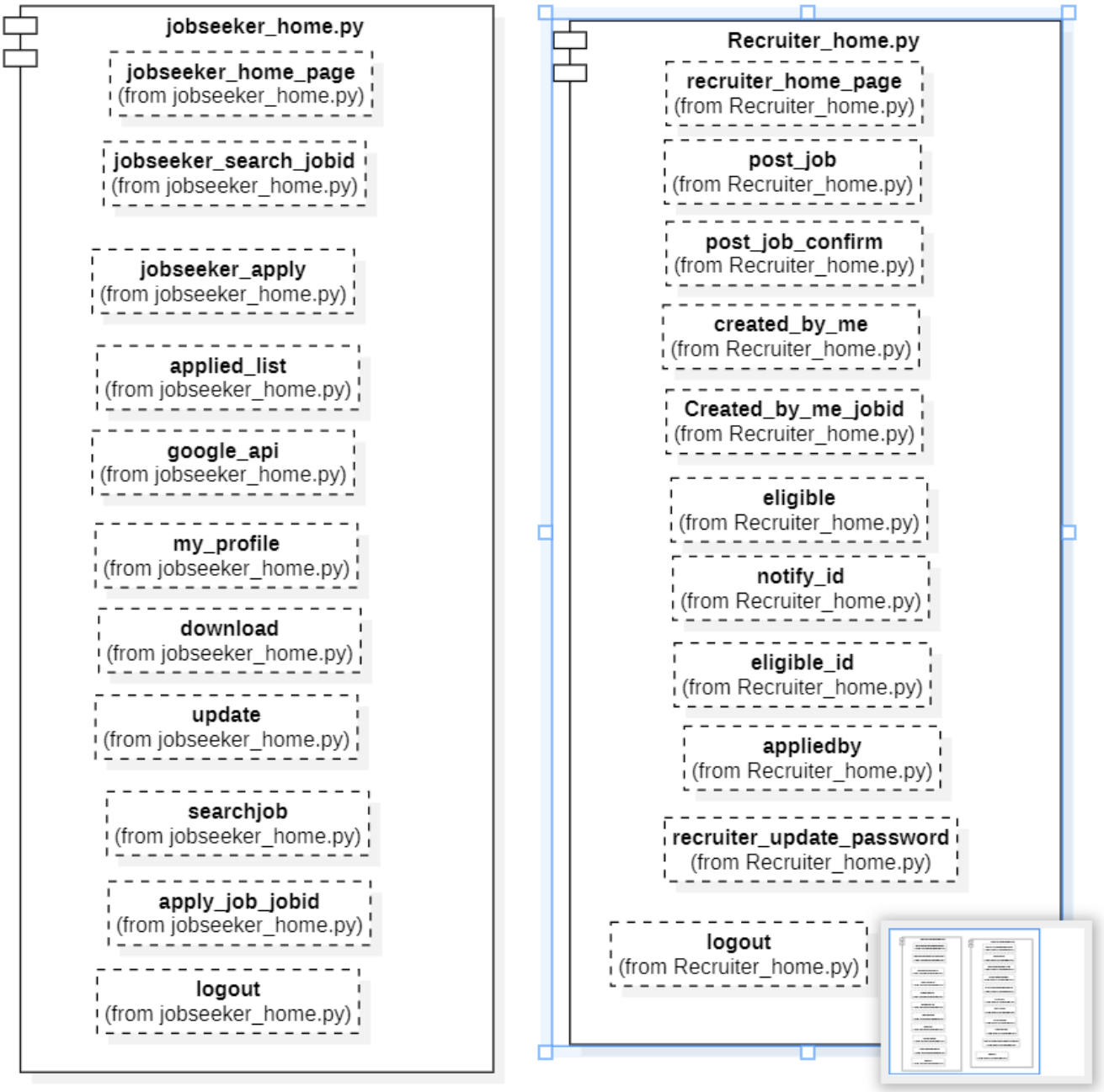
In a large project there will be many files that make up the system. These files will have dependencies on one another. The nature of these dependencies will depend on the language or languages used for the development and may exist at compile-time, at link-time or at run-time. There are also dependencies between source code files and the executable files or byte code files that are derived from them by compilation. Component diagrams are one of the two types of implementation diagram in UML.

Component diagrams show these dependencies between software components in the system. Stereotypes can be used to show dependencies that are specific to particular languages also. A component diagram shows the allocation of classes and objects to components in the physical design of a system. A component diagram may represent all or part of the component architecture of a system along with dependency relationships

Component Diagram:



Components Diagram for CareerConnect(CC)



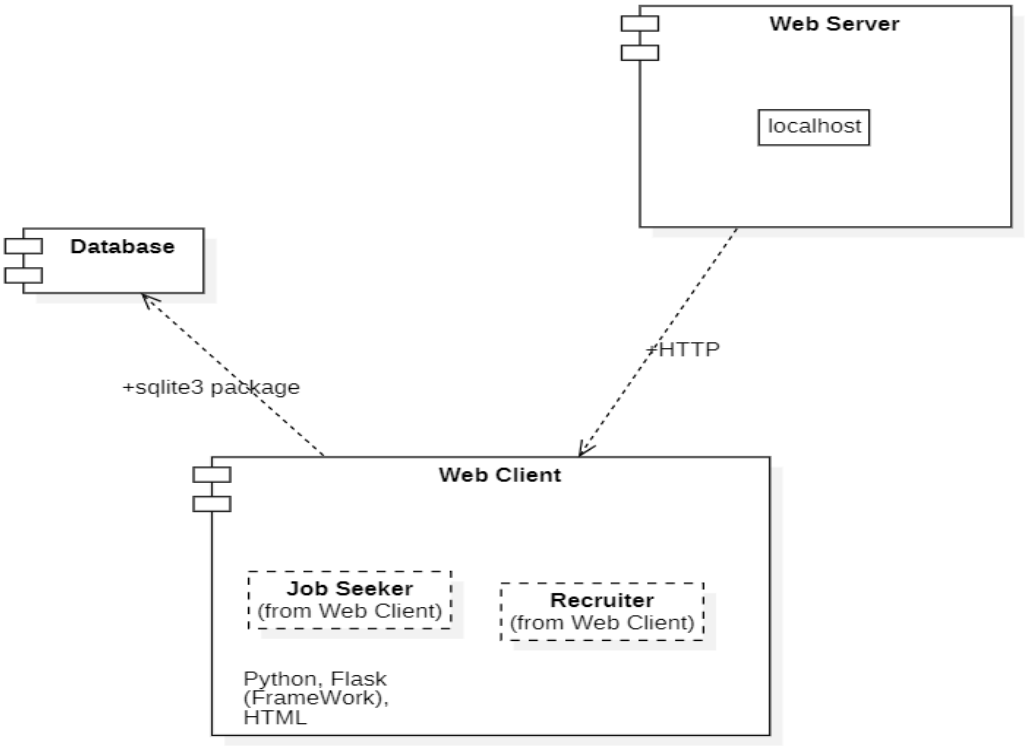
Components Diagram for CareerConnect(CC)

Deployment:

The second type of implementation diagram provided by UML is the deployment diagram. Deployment diagrams are used to show the configuration of run-time processing elements and the software components and processes that are located on them. Deployment diagrams are made up of nodes and communication associations.

Nodes are typically used to show computers and the communication associations show the network and protocols that are used to communicate between nodes. Nodes can be used to show other processing resources such as people or mechanical resources.

Nodes are drawn as 3D views of cubes or rectangular prisms, and the following figure shows a simplest deployment diagram where the nodes connected by communication associations



Deployment Diagram for CareerConnect(CC)

14. SAMPLE APPLICATION CODE AND DATABASE

Various modules in the system are:

1. Home Page
2. Jobseeker_Registration
3. Recruiter_Registration
4. Login
5. Jobseeker_homepage
6. Jobseeker_searchjobid
7. Jobseeker_serachjob
8. Jobseeker_googleapi
9. Jobseeker_myprofile
10. Jobseeker_apply
11. Jobseeker_applied_list
12. Jobseeker_update
13. Jobseeker_download
14. Jobseeker_logout
15. Recruiter_homepage
16. Recruiter_postjob
17. Recruiter_post_job_confirm
18. Recruiter_created_by_me
19. Recruiter_created_by_me/<jobid>
20. Recruiter_eligible
21. Recruiter_notify/<id>
22. Recrutier_eligible/<id>
23. Recruiter_updatepassword
24. Recruiter_appliedby
25. Recruiter_logout
26. Forgot password

Code for Index Page

(app.py)

```python
from flask import Flask, render_template
from flask_mail import Mail
from login import login_blueprint
from recruiter_register import recruiter_blueprint
from jobseeker_register import jobseeker_blueprint

app = Flask(__name__)


app.config["MAIL_USERNAME"] = "postmaster@connectcareer.tech"
app.config["MAIL_PORT"] = 587
app.config["MAIL_SERVER"] = "smtp.mailgun.org"
app.config["MAIL_PASSWORD"] = "#########" #Use Your Password
app.config['EMAIL_USE_TLS'] = True
app.secret_key = "Thrinadh"
mail = Mail(app)

app.register_blueprint(recruiter_blueprint)
app.register_blueprint(jobseeker_blueprint)
app.register_blueprint(login_blueprint)


THRINADH43
@app.route('/', methods=["POST", "GET"])
def home():
    return render_template('index.html')


if __name__ == "__main__":
    app.run(debug=True)
```

(index.html)

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Index Page</title>
</head>
<body>
        <form action="{{url_for('recruiter_blueprint.recruiter_register')}}" method="POST">
            <input type="submit" value="Recuriter Register">
        </form>
        <form action="{{url_for('jobseeker_blueprint.jobseeker_register')}}" method="POST">
            <input type="submit" value="Job Seeker Register">
        </form>
        <form action="{{url_for('login_blueprint.login')}}" method="POST">
            <input type="submit" value="Login">
        </form>
</body>
</html>
```

Code for Jobseeker_Registration (Jobseeker_register.py)

```python
from mailbox import Message
from flask import Flask, render_template, request, flash, url_for, Blueprint, current_app,session
from werkzeug.utils import secure_filename
from wtforms import Form, StringField, validators, PasswordField, SubmitField, EmailField, FileField, SelectField, \
    RadioField
from wtforms.validators import DataRequired
from flask_mail import Mail, Message
from flask_wtf import FlaskForm
from database import get_db
import os

upload_path = './resumes'

if not os.path.exists(upload_path):
    os.mkdir(upload_path)

jobseeker_blueprint = Blueprint('jobseeker_blueprint', __name__)

sent_otp = 652013
login_email = []
user_details = []
userid = []

allowed_extensions = {'pdf'}


THRINADH43
class CreateData(FlaskForm):
    name = StringField("name", [validators.InputRequired()])
    mail = EmailField("Email", [validators.InputRequired()])
    password = PasswordField("password", [validators.InputRequired()])
    conform_password = PasswordField("password", [validators.InputRequired()])
    submit = SubmitField("Submit")


THRINADH43
class Submit_otp(Form):
    otp = StringField("OTP", [validators.InputRequired()])
    submit = SubmitField("submit")
```

```
THRINADH43                                                                    11  2
class Userdetails(FlaskForm):
    jobtype = RadioField("Type", choices=[('Full Time', 'Full Time'), ('Part Time', 'Part Time'), ('Intern', 'Intern')]
                         , default='Full Time')
    bio = StringField("description", [validators.InputRequired()])
    skill1 = SelectField('skill 1',
                         choices=[('None', 'select from the below'), ('python', 'python'), ('java', 'java'),
                                  ('c++', 'c++'), ])
    skill2 = SelectField('skill 2',
                         choices=[('None', 'select from the below'), ('mysql', 'mysql'), ('mongodb', 'mondogb')])
    location1 = SelectField('Location 1',
                            choices=[('None', 'Select from below'), ('Hyderabad', 'Hyderabad'),
                                     ('Banglore', 'Banglore'), ('Chennai', 'Chennai'), ('Mumbai', 'Mumbai')])
    location2 = SelectField('Location 2',
                            choices=[('None', 'Select from below'), ('Hyderabad', 'Hyderabad'), ('Chennai', 'Chennai'),
                                     ('Banglore', 'Banglore'), ('Mumbai', 'Mumbai')])
    remote = RadioField("Remote", choices=[('yes', 'yes'), ('no', 'no')], default='no')
    occupation = SelectField("I am a", choices=[("None", "Select from below"), ("Student", "Student"),
                                                ("Professional", "Professional")])
    graduation_year = SelectField("Graduation Year",
                                  choices=[("None", "None"), ("2024", "2024"), ("2025", "2025"), ("2026", "2026")])
    experience = SelectField("Years of Experience",
                             choices=[('None', 'Select from below'), ('0-2', '0-2'), ('2-5', '2-5'),
                                      ('5-8', '5-8'), ('8+', '8+')])
    previous_experience = StringField("PreviousExperience", [validators.Length(min=10, max=100)])
    file = FileField("Upload a PDF file")
    githublink = StringField("GitHub Link", [validators.length(min=10, max=2500)])
    linkedinlink = StringField("Linkedin Link", [validators.Length(min=10, max=2500)])
    college = StringField("College", [validators.Length(min=10, max=50)])
    submit = SubmitField('submit')
```

```
70                                                                            11  2
    THRINADH43
71  @jobseeker_blueprint.route("/jobseeker_register", methods=["POST", "GET"])
72  def jobseeker_register():
73      form = CreateData()  # request.form is required
74      if form.validate_on_submit():
75          print(form.mail.data)
76          login_email.append(form.mail.data)
77          user_details.append(form.name.data)
78          user_details.append(form.mail.data)
79          user_details.append(form.password.data)
80          return render_template('jobseeker_authentication.html')
81      return render_template('jobseeker_register.html', form=form)
82
83
    THRINADH43 *
84  @jobseeker_blueprint.route('/jobseeker_validate-email/', methods=["POST", "GET"])
85  def validate():
86      dmail = login_email[0]
87      login_email.pop()
88      msg = Message("OTP To Authenticate", sender="connectcareer78@gmail.com", recipients=[dmail])
89      msg.body = f"Your OTP to Authenticate: {sent_otp}"
90      with current_app.app_context():
91          current_app.extensions['mail'].send(msg)
92      received_otp = request.args.get("otp")
93      if received_otp == str(sent_otp):
94          with current_app.app_context():
95              db = get_db()
96              name = user_details[0]
97              mail = user_details[1]
```

```
            mail = user_details[1]
            password = user_details[2]
            db.execute("insert into user (name,mail,password,type) values (?,?,?,?)",
                       [name, mail, password, 0])
            db.commit()

            user_id = db.execute('select user.id as id from user where name = ? and password = ?', [user_details[0],
                                                                                                    user_details[2]])
            user_result = user_id.fetchone()
            print(user_id)
            user_details.clear()
            print(f"userid: {user_result['id']}")
            session['id'] = user_result['id']
            print(f"userid from session: {session['id']}")
        return render_template('jobseeker_authentication_successful.html')
    else:
        return render_template('auth_not.html')
    # return render_template('authentication.html')


 THRINADH43
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in allowed_extensions


 THRINADH43 *
@jobseeker_blueprint.route('/registerdetails', methods=["POST", "GET"])
def registerdetails():
    userdata = Userdetails()
```

app.py | index.html | jobseeker_register.py

```python
        jobtype = userdata.jobtype.data
        skill1 = userdata.skill1.data
        skill2 = userdata.skill2.data
        location1 = userdata.location1.data
        location2 = userdata.location2.data
        remote = userdata.remote.data
        occupation = userdata.occupation.data
        graduation = userdata.graduation_year.data
        if graduation:
            experience = "0-2"
        else:
            experience = userdata.experience.data
        previousexperience = userdata.previous_experience.data
        resume = filename
        githublink = userdata.githublink.data
        linkedinlink = userdata.linkedinlink.data
        college = userdata.college.data
        db = get_db()
        db.execute(
            'insert into jobseeker (id, bio, jobtype, skill1, skill2, location1, location2, remote, occupation, '
            'graduation, experience, previousexperience, resume, githublink, linkedinlink, college) values (?,?,?,?,'
            '?,?,?,?,?,?,?,?,?,?,?)',
            [user, bio, jobtype, skill1, skill2, location1, location2, remote, occupation, graduation, experience,
             previousexperience, resume, githublink, linkedinlink, college])
        db.commit()
        session.clear()
        return render_template('jobseeker_user_register.html')
    return render_template('jobseeker_user_details.html', form=userdata)
```

(jobseeker_register.html)

app.py | index.html | jobseeker_register.py | jobseeker_register.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home Page</title>
</head>
<body>

    <form action="/jobseeker_register" method="post">
        {{form.csrf_token}}
        <p>{{form.name.label}} <br> {{form.name}}</p>
        <p>{{form.mail.label}} <br> {{form.mail}}</p>
        <p>{{form.password.label}} <br> {{form.password}}</p>
        <p>{{form.conform_password.label}} <br> {{form.conform_password}}</p>

        <p>{{form.submit.label}} <br> {{form.submit}} </br></p>
    </form>
</body>
</html>
```

(jobseeker_authentication.html)

app.py | index.html | jobseeker_register.py | jobseeker_authentication.html | jobseeker_register.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Authentication</title>
</head>
<body>
    <form action="/jobseeker_validate-email" action = "POST">
      <label for="otp" id="otp" name="otp">Enter OTP Sent to mail: </label>
      <input type="number" for="otp" name="otp">
      <input type="submit">
    </form>
</body>
</html>
```

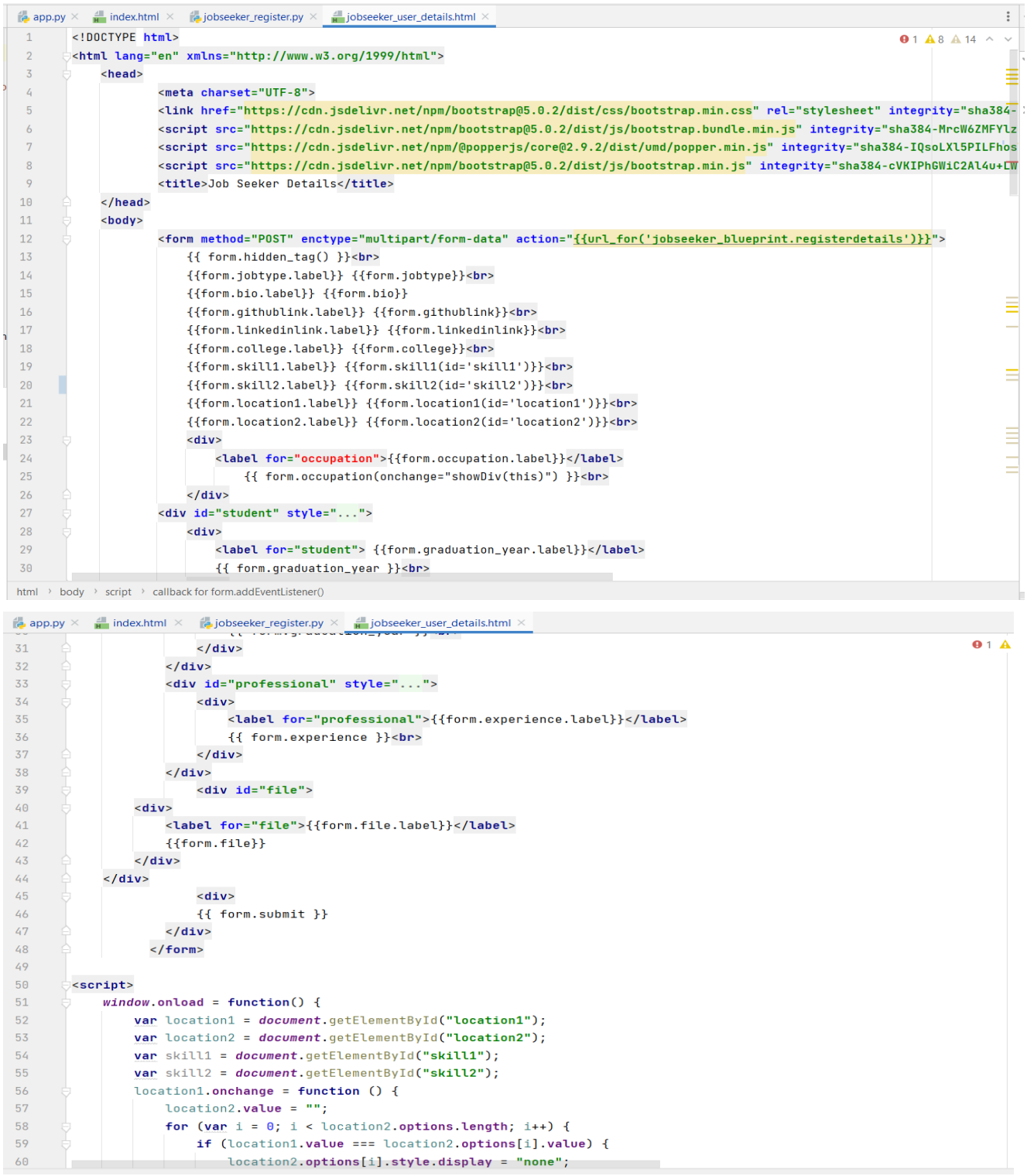
(jobseeker_authentication_successful.html)

app.py | index.html | jobseeker_register.py | jobseeker_authentication_successful.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>OTP Successful</title>
</head>
<body>
    <h3 align="center">Authentication Successful</h3>
    <h3 align="center">You would be Redirected to User Data Entry Page in 3 seconds.</h3>
    <h4 align="center">Kindly fill all the details</h4>
    <h3 align="center"><a href="{{url_for('jobseeker_blueprint.registerdetails')}}">If not click Here</a></h3>
    <script>
    setTimeout(() => {
        location.href = "{{ url_for('jobseeker_blueprint.registerdetails') }}";
    }, 3000);
</script>

</body>
</html>
```

(jobseeker_user_register.html)



```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/html">
    <head>
            <meta charset="UTF-8">
            <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
            <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-MrcW6ZMFYlz
            <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js" integrity="sha384-IQsoLXl5PILFhos
            <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js" integrity="sha384-cVKIPhGWiC2Al4u+LW
            <title>Job Seeker Details</title>
    </head>
    <body>
            <form method="POST" enctype="multipart/form-data" action="{{url_for('jobseeker_blueprint.registerdetails')}}">
                {{ form.hidden_tag() }}<br>
                {{form.jobtype.label}} {{form.jobtype}}<br>
                {{form.bio.label}} {{form.bio}}
                {{form.githublink.label}} {{form.githublink}}<br>
                {{form.linkedinlink.label}} {{form.linkedinlink}}<br>
                {{form.college.label}} {{form.college}}<br>
                {{form.skill1.label}} {{form.skill1(id='skill1')}}<br>
                {{form.skill2.label}} {{form.skill2(id='skill2')}}<br>
                {{form.location1.label}} {{form.location1(id='location1')}}<br>
                {{form.location2.label}} {{form.location2(id='location2')}}<br>
                <div>
                    <label for="occupation">{{form.occupation.label}}</label>
                        {{ form.occupation(onchange="showDiv(this)") }}<br>
                </div>
            <div id="student" style="...">
                <div>
                    <label for="student"> {{form.graduation_year.label}}</label>
                    {{ form.graduation_year }}<br>
                </div>
            </div>
            <div id="professional" style="...">
                <div>
                    <label for="professional">{{form.experience.label}}</label>
                    {{ form.experience }}<br>
                </div>
            </div>
                <div id="file">
        <div>
            <label for="file">{{form.file.label}}</label>
            {{form.file}}
        </div>
    </div>
                <div>
                {{ form.submit }}
            </div>
          </form>

<script>
    window.onload = function() {
        var location1 = document.getElementById("location1");
        var location2 = document.getElementById("location2");
        var skill1 = document.getElementById("skill1");
        var skill2 = document.getElementById("skill2");
        location1.onchange = function () {
            location2.value = "";
            for (var i = 0; i < location2.options.length; i++) {
                if (location1.value === location2.options[i].value) {
                    location2.options[i].style.display = "none";
```


```
    var file = input.files[0];
    if (file) {
      var fileSize = file.size;
      var fileExtension = file.name.split(".")[1].toLowerCase();
      if (fileSize > maxSize) {
        alert("File size should be less than 3MB");
        e.preventDefault();
      } else if (allowedExtensions.indexOf(fileExtension) === -1) {
        alert("Only PDF files are allowed");
        e.preventDefault();
      } else {
        // submit the form
      }
    }
  });

</script>

    </body>
</html>
```

Code for Recruiter_Register

(recruiter_register.py)

recruiter_register.py

```python
from mailbox import Message

from flask import Flask, render_template, request, flash, url_for, Blueprint, current_app
from wtforms import Form, StringField, validators, PasswordField, SubmitField, EmailField
from wtforms.validators import InputRequired
from flask_mail import Mail, Message
from flask_wtf import FlaskForm
from database import get_db

recruiter_blueprint = Blueprint('recruiter_blueprint', __name__)

sent_otp = 652013
login_email = []
user_details = []


THRINADH43
def email_validators(form, field):
    domains_not_allowed = ['yahoo.com', 'gmail.com']
    newdata = field.data.split('@')[-1]
    print(newdata)
    if newdata in domains_not_allowed:
        print("Error Generated")
        field.errors.append("Please Enter your professional mail")
        return False
    return True


```

```python
THRINADH43
class CreateData(FlaskForm):
    name = StringField("name", validators=[InputRequired()])
    mail = EmailField("Email", validators=[InputRequired(), email_validators])
    password = PasswordField("password", validators=[InputRequired()])
    conform_password = PasswordField("password", validators=[InputRequired()])
    company = StringField("Company", validators=[InputRequired()])
    submit = SubmitField("Submit")


THRINADH43
class Submit_otp(Form):
    otp = StringField("OTP", validators=[InputRequired()])
    submit = SubmitField("submit")


THRINADH43 *
@recruiter_blueprint.route("/recruiter_register", methods=["POST", "GET"])
def recruiter_register():
    form = CreateData()  # request.form is required
    domain_not_allowed = ['yahoo.com', 'gmail.com']
    if form.validate_on_submit():
        print(form.mail.data)
        login_email.append(form.mail.data)
        user_details.append(form.name.data)
        user_details.append(form.mail.data)
        user_details.append(form.password.data)
        user_details.append(form.company.data)
        return render_template('recruiter_authentication.html')
    return render_template('recruiter_register.html', form=form)
```

recruiter_register() › if form.validate_on_submit()

recruiter_register.py

```python
THRINADH43
@recruiter_blueprint.route('/recruiter_validate-email/', methods=["POST", "GET"])
def validate():
    dmail = login_email[0]
    login_email.pop()
    msg = Message("OTP To Authenticate", sender="thrinadh.manubothu@gmail.com", recipients=[dmail])
    msg.body = f"Your OTP to Authenticate: {sent_otp}"
    with current_app.app_context():
        current_app.extensions['mail'].send(msg)
    received_otp = request.args.get("otp")
    if received_otp == str(sent_otp):
        print("OTP Successful")
        with current_app.app_context():
            db = get_db()
            name = user_details[0]
            print(name)
            mail = user_details[1]
            password = user_details[2]
            company = user_details[3]
            db.execute("insert into user (name,mail,password,type,company) values (?,?,?,?,?)",
                       [name, mail, password, 1, company])
            db.commit()
            user_details.clear()
            print(user_details)
        return render_template('auth_successful.html')
    else:
        return render_template('auth_not.html')
```

(recruiter_register.html)

recruiter_register.py | recruiter_register.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home Page</title>
</head>
<body>
      {% with messages = get_flashed_messages() %}
            {% if messages %}
                {% for msg in messages %}
                    <h4 align="center">{{msg}}</h4>
                {% endfor %}
             {% endif %}
      {% endwith %}
      {% for error in form.mail.errors %}
        <span style="color: red">[{{ error }}]</span>
    {% endfor %}
      <form action="/recruiter_register" method="post">
          {{form.csrf_token}}
          <p>{{form.name.label}} <br> {{form.name}}</p>
          <p>{{form.mail.label}} <br> {{form.mail}}</p>
          <p>{{form.password.label}} <br> {{form.password}}</p>
          <p>{{form.conform_password.label}} <br> {{form.conform_password}}</p>
          <p>{{form.company.label}} <br> {{form.company}}</p>
          <p>{{form.submit.label}} <br> {{form.submit}} </br></p>
      </form>
</body>
</html>
```

(recruiter_authentication.html)

recruiter_register.py | recruiter_authentication.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Authentication</title>
</head>
<body>
      <form action="/recruiter_validate-email" action = "POST">
        <label for="otp" id="otp" name="otp">Enter OTP Sent to mail: </label>
        <input type="number" for="otp" name="otp">
        <input type="submit">
      </form>
</body>
</html>
```

Code for Login

(login.py)

login.py

```
1   import ...
10
11  login_blueprint = Blueprint('login_blueprint', __name__)
12  login_blueprint.register_blueprint(jobseeker_home, url_prefix='/login')
13  login_blueprint.register_blueprint(recruiter_home, url_prefix='/login')
14
15  sent_otp = 652013
16
17

    THRINADH43
18  class CreateData(FlaskForm):
19      mail = EmailField("Email", validators=[InputRequired()])
20      password = StringField("Password", validators=[InputRequired()])
21      submit = SubmitField("Submit")
22
23

    THRINADH43
24  @login_blueprint.route('/login', methods=["GET", "POST"])
25  def login():
26      form = CreateData()
27      mail = request.form.get('mail')
28      if form.validate_on_submit():
29          # mail = form.mail.data
30          password = form.password.data
31          with current_app.app_context():
32              db = get_db()
33              cur = db.execute('select id,password,type from user where  mail = ?', [mail])
34              user_result = cur.fetchone()
35              print(user_result)
36              print(type(user_result))
```

login.py

```
            if user_result is not None:
                if user_result['password'] == password:
                    print("Password Validated")
                    # session['logged_in'] = True
                    session['id'] = user_result['id']
                    session['type'] = user_result['type']
                    user_id = session['id']
                    print(user_id)
                    if user_result['type'] == 1:
                        return redirect(url_for('login_blueprint.recruiter_home.recruiter_home_page'))
                    else:
                        return redirect(url_for('login_blueprint.jobseeker_home.jobseeker_home_page'))
                else:
                    return "Wrong Password/User Not Found"
            print(session.get('id'))
    return render_template('login.html', form=form)


new *
@login_blueprint.route('/forgotpassword', methods=["POST", "GET"])
def forgotpassword():
    change = True
    if request.method == "POST":
        dmail = request.form.get('email')
        print(request.form.get('email'))
        db = get_db()
        email_found = db.execute('select * from user where mail = ?', [request.form.get('email')]).fetchone()
        print(type(email_found))
        if email_found is None:
            print("No Such User Found")
```

login.py

```
            msg = "No Such User Found"
            return render_template('change_password.html', msg=msg, change=change)
        else:
            found = True
            session['mail'] = dmail
            msg = Message("OTP To Authenticate", sender="thrinadh.manubothu@gmail.com", recipients=[dmail])
            msg.body = f"Your OTP to Authenticate: {sent_otp}"
            with current_app.app_context():
                current_app.extensions['mail'].send(msg)
            return render_template('change_password.html', found=True)
    return render_template('change_password.html', change=change)


new *
@login_blueprint.route('/forgotpasswordotp', methods=["POST", "GET"])
def forgotpasswordotp():
    if request.method == "POST":
        print(request.form.get('otp'))
        received = request.form.get('otp')
        print(received)
        print(type(received))
        if int(received) != sent_otp:
            msg = "OTP Doesn't Match"
            return render_template('change_password.html', msg=msg)
        else:
            return render_template('change_password_data.html')
    msg = "Please check with the details submitted."
    return render_template('change_password.html', msg=msg)
```

```
new *
@login_blueprint.route('/updatepassword', methods=["POST", "GET"])
def updatepassword():
    if request.method == "POST":
        password1 = request.form.get('password1')
        password2 = request.form.get('password2')
        if password1 != password2:
            msg = "Passwords Doesn't Match"
            return render_template('change_password_data.html', msg=msg)
        else:
            db = get_db()
            db.execute("update user set password = ? where mail = ?", [password1, session['mail']])
            db.commit()
            session.pop('mail')
            message = "Your password Has been Updated"
            return render_template('change_password_data.html', message=message)
    return render_template('change_password_data.html')
```

(login.html)

login.py × login.html ×

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>
<div class="container">
  <h2>Login Form</h2>
  <form action="/login" method="post">
    {{form.csrf_token}}
    <div class="form-group">
      {{form.mail.label}}
       <input type="text" class="form-control" name="mail" placeholder="Enter mail" >
    </div>
    <div class="form-group">
      {{form.password.label}}
      <input type="password" class="form-control" id="pwd" placeholder="Enter password" name="password">
    </div>
    <button type="submit" class="btn btn-default">{{form.submit}}</button>
  </form>
  </form>
  <form action="/forgotpassword" method="GET">
    <input type="submit" value="Forgot Password?">
  </form>
</div>

</body>
</html>
```

html › body › div.container › form

(change_password.html)

login.py | change_password.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Change Password</title>
</head>
<body>
    {% if change %}
    <form action="/forgotpassword" method="POST">
        <label for="email">Enter Mail:</label>
        <input type="email" name="email">
        <input type="submit" value="submit">
    </form>
    {% endif %}
    {% if msg %}
        {{msg}}
    {% endif %}
    {% if found %}
    <form action="/forgotpasswordotp" method="post">
        <label for="'otp">Enter OTP Received to mail: </label>
        <input type="number" name="otp">
        <input type="submit" value="submit">
    </form>
    {% endif %}
</body>
</html>
```

(change_password_data.html)

login.py | change_password_data.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Changing Password</title>
</head>
<body>
    {% if msg %}
        {{msg}}
    {% endif %}
    <form action="/updatepassword" method="post">
      <label for="password1">Enter Password: </label>
       <input type="password" name="password1">
      <label for="password2">Enter Confirm Password: </label>
      <input type="password" name="password2">
      <input type="submit" value="submit">
    </form>
    {% if message %}
        {{message}}
        <a href="/">Click Here to Redirect to home</a>
    {% endif %}
</body>
</html>
```

Code for Jobseeker_home_page

(jobseeker_home.py)

```python

THRINADH43 *
@jobseeker_home.route('/jobseeker_home_page')
def jobseeker_home_page():
    search = True
    if 'id' not in session or session['type'] != 0:
        session.clear()
        return redirect('/login')
    db = get_db()
    user = db.execute('select user.name as name from user where id = ?', [session['id']])
    userresult = user.fetchone()
    session['name'] = userresult['name']
    return render_template('jobseeker_home_page.html', search=search)


```

(jobseeker_home_page.html)

```html
    <div class="container">
      <div class="row">
        <div class="col-md-10"></div>
        <div class="col-md-10">
          <p> Hello, {{session['name']}}</p>
            {% if search %}
          <form action="{{url_for('login_blueprint.jobseeker_home.jobseeker_search_jobid')}}" method="post">
          <label for="jobid">Search JobID: </label>
          <input type="number" name="jobid" placeholder="Enter JobID">
          <input type="submit">
          </form>
           {% endif %}
        </div>
        <div class="col-md-4"></div>
      </div>
    </div>
```

Code for Jobseeker_serach_jobid

(jobseeker_home.py)

```
new *
@jobseeker_home.route('/jobseeker_search_jobid', methods=["POST", "GET"])
def jobseeker_search_jobid():
    search = True
    if request.method == "POST":
        jobid = request.form.get('jobid')
        db = get_db()
        jobresult = db.execute('select jobposting.description as description,'
                               ' jobposting.experience as experience, jobposting.title as title, jobposting.link as link,'
                               'jobposting.type as type, jobposting.createdby as createdby, jobposting.location1 as location1,'
                               'jobposting.location2 as location2, jobposting.skill1 as skill1, jobposting.skill2 as skill2,'
                               'jobposting.remote as remote from jobposting where jobid = ?', [jobid])
        jobsearch = jobresult.fetchone()
        if jobsearch is None:
            msg = "The Job ID is not Found"
            return render_template('jobseeker_home_page.html', msg=msg, search=search)
        createdid = jobsearch['createdby']
        created = db.execute('select user.name as name from user where id = ?', [createdid])
        createdname = created.fetchone()
        createdidname = createdname['name']
        print(type(jobsearch))
        session['jobid'] = jobid
        return render_template('jobseeker_home_page.html', createdidname=createdidname, jobsearch=jobsearch,
                               jobid=jobid, search=search)

```

Code for Jobseeker_apply

(jobseeker_home.py)

```
new *
@jobseeker_home.route('/jobseeker_apply', methods=["POST", "GET"])
def jobseeker_apply():
    # print(session['job'])
    db = get_db()
    check_applied = db.execute('select * from applied where applicantid = ?  and jobid = ?',
                               [session['id'], session['jobid']])
    check = check_applied.fetchone()
    if check is not None:
        msg = "You already Applied to this Job."
        return render_template('jobseeker_home_page.html', msg=msg)
    applied = db.execute('insert into applied (applicantid,jobid) values (?,?)', [session['id'], session['jobid']])
    apply = db.commit()
    print(session['jobid'])
    session.pop('jobid')
    msg = "Applied to job"
    # TODO: Check that the jobseeker applies only once.If They Tries to attemt second time. Alert them not to do it.
    return render_template('jobseeker_home_page.html', msg=msg)

```

Code for Jobseeker_applied_list

(jobseeker_home.py)

```
new *
@jobseeker_home.route('/applied_list', methods=["POST", "GET"])
def applied_list():
    db = get_db()
    apply_list = db.execute('select a.jobid as jobid,j.skill1 as skill1,'
                            'j.title as title, j.type as type, j.experience as experience, u.name as name '
                            'from applied a JOIN jobposting j on j.jobid = a.jobid'
                            ' JOIN user u on u.id = j.createdby where a.applicantid = ?', [session['id']])
    apply = apply_list.fetchall()
    if len(apply) == 0:
        msg = "You didn't apply to any job.\nplease check for suitable jobs and apply."
        return render_template('jobseeker_home_page.html', msg=msg)
    return render_template('jobseeker_home_page.html', apply=apply)
```

(jobseeker_home_page.html)

```
{% if apply %}
    {% for d in apply %}
        <p align="center">JobId:{{d['jobid']}} Title: {{d['title']}} Type: {{d['type']}}</p>
        <p align="center">Experience: {{d['experience']}} Skill1: {{d['skill1']}} Createdby: {{d['name']}}</p>
    {% endfor %}
{% endif %}
{% if d %}
```

Code for jobseeker_googleapi

(jobseeker_home.py)

```
new *
@jobseeker_home.route('/google_api', methods=["GET", "POST"])
def google_api():
    d = Data()
    if request.method == "POST":
        return request.data
    return render_template('jobseeker_google_api.html', d=d)
```

(jobseeker_google_api.html)

```
29      <script async src="https://cse.google.com/cse.js?cx=1235283dfebe640d7">
30          {{d.job}}
31      </script>
32      <div class="gcse-search" ></div>
```

Code for jobseeker_myprofile

(jobseeker_home.py)

```
new *
159 @jobseeker_home.route('/myprofile', methods=["GET", "POST"])
160 def myprofile():
161     my = True
162     db = get_db()
163     profile = db.execute('select j.jobtype as jobtype,'
164                          'j.bio as bio,j.skill1 as skill1, j.skill2 as skill2,'
165                          'j.location1 as location1,j.location2 as location2,j.remote as remote,'
166                          'j.githublink as github,j.linkedinlink as linkedin,j.experience as experience'
167                          ' from jobseeker j where j.id = ?', [session['id']])
168     myprofile = profile.fetchone()
169     return render_template('jobseeker_home_page.html', myprofile=myprofile, my=my)
170
171
```

(jobseeker_home_page.html)

```
75      {% if my %}
76      <p>BIO: {{myprofile['bio']}}</p><br>
77      <p>Job Type: {{myprofile['jobtype']}}</p>
78      <p>:Experience {{myprofile['experience']}}</p><br>
79      <p>Skill1: {{myprofile['skill1']}}</p><br>
80      <p>Skill2: {{myprofile['skill2']}}</p><br>
81      <p>Location1: {{myprofile['location1']}}</p><br>
82      <p>Location2: {{myprofile['location2']}}</p><br>
83      <p>Remote: {{myprofile['remote']}}</p><br>
84      <p>Github: {{myprofile['github']}}</p>
85      <p>Linkedin: {{myprofile['linkedin']}}</p>
86      <h4><a href="{{url_for('login_blueprint.jobseeker_home.download')}}">Click Here to download Resume</a>
87      <a href="{{url_for('login_blueprint.jobseeker_home.update')}}">Click Here to update profile</a></h4>
88      {% endif %}
```

Code for Jobseeker_download

(jobseeker_home.py)

```
new *
@jobseeker_home.route('/download', methods=["GET", "POST"])
def download():
    db = get_db()
    file = db.execute('select jobseeker.resume as resume from jobseeker where id = ?', [session['id']])
    fileresult = file.fetchone()
    filename = fileresult['resume']
    try:
        return send_file(f"./resumes/{filename}", as_attachment=True)
    except Exception as e:
        return str(e)
```

Code for Jobseeker_update

(jobseeker_home.py)

jobseeker_home.py | jobseeker_home_page.html

```
@jobseeker_home.route('/update', methods=["GET", "POST"])
def update():
    to_update = True
    userdata = Userdetails()
    db = get_db()
    original_data = db.execute("select j.jobtype as jobtype,j.experience as experience,j.linkedinlink as linkedin,"
                               "j.githublink as github,j.skill1 as skill1,j.skill2 as skill2, "
                               "j.location1 as location1,j.location2 as location2,j.bio as bio,j.occupation as occupation,"
                               "j.graduation as graduation,j.remote as remote,j.resume as resume,"
                               "j.college as college from jobseeker j where id = ?", [session['id']])
    original = dict(original_data.fetchone())
    if request.method == "POST":
        file = userdata.file.data
        filename = secure_filename(file.filename)
        bio = userdata.bio.data
        jobtype = userdata.jobtype.data
        skill1 = userdata.skill1.data
        skill2 = userdata.skill2.data
        location1 = userdata.location1.data
        location2 = userdata.location2.data
        remote = userdata.remote.data
        occupation = userdata.occupation.data
        graduation = userdata.graduation_year.data
        if graduation:
            experience = "0-2"
        else:
            experience = userdata.experience.data
        previousexperience = userdata.previous_experience.data
        resume = filename
        githublink = userdata.githublink.data
```

jobseeker_home.py | jobseeker_home_page.html

```
219         linkedinlink = userdata.linkedinlink.data
220         college = userdata.college.data
221         if original['bio'] != bio:
222             if bio:
223                 print("Query Entered here")
224                 original['bio'] = bio
225         if original['resume'] != resume:
226             if resume:
227                 original['resume'] = resume
228                 if file and allowed_file(file.filename):
229                     filename = secure_filename(file.filename)
230                     file.save(os.path.join(upload_path, filename))
231         if original['jobtype'] != jobtype:
232             if jobtype != "None":
233                 original['jobtype'] = jobtype
234         if original['skill1'] != skill1:
235             if skill1 != "None":
236                 original['skill1'] = skill1
237         if original['skill2'] != skill2:
238             if skill2 != "None":
239                 original['skill2'] = skill2
240         if original['location1'] != location1:
241             if location1 != "None":
242                 original['location1'] = location1
243         if original['location2'] != location2:
244             if location2 != "None":
245                 original['location2'] = location2
246         if original['remote'] != remote:
247             original['remote'] = remote
248         if original['occupation'] != occupation:
```

jobseeker_home.py | jobseeker_home_page.html

```
250                 original['occupation'] = occupation
251         if original['graduation'] != graduation:
252             if graduation != "None":
253                 original['graduation'] = graduation
254         if original['linkedin'] != linkedinlink:
255             if linkedinlink:
256                 original['linkedin'] = linkedinlink
257         if original['github'] != githublink:
258             if githublink:
259                 original['github'] = githublink
260         if original['college'] != college:
261             if college:
262                 original['college'] = college
263         if original['experience'] != experience:
264             original['experience'] = experience
265         try:
266             db.execute(
267                 'update jobseeker set bio = ?, jobtype = ?, skill1 = ?, skill2 = ?, location1 = ?,'
268                 ' location2 = ?, remote = ?, occupation = ?,'
269                 'graduation = ?, experience = ?, resume = ?, githublink = ?, linkedinlink = ?, college = ? where id = ?',
270                 [original['bio'], original['jobtype'], original['skill1'], original['skill2'], original['location1'],
271                  original['location2'], original['remote'], original['occupation'], original['graduation'],
272                  original['experience'],
273                  original['resume'], original['github'], original['linkedin'], original['college'], session['id']])
274             db.commit()
275             msg = "Data Updated Successfull!"
276             return render_template('jobseeker_update.html', msg=msg)
277         except Exception as e:
278             raise e
279     return render_template('jobseeker_update.html', form=userdata, to_update=to_update)
```

(jobseeker_update.html)

jobseeker_home.py | jobseeker_update.html | jobseeker_home_page.html

```
29              {% if msg %}
30                  <h4>{{msg}}</h4>
31              {% endif %}
32              {% if to_update %}
33              <form method="POST" enctype="multipart/form-data" action="{{url_for('login_blueprint.jobseeker_home.update')}}">
34                  {{ form.hidden_tag() }}<br>
35                  {{form.jobtype.label}} {{form.jobtype}}<br>
36                  {{form.bio.label}} {{form.bio}}
37                  {{form.githublink.label}} {{form.githublink}}<br>
38                  {{form.linkedinlink.label}} {{form.linkedinlink}}<br>
39                  {{form.college.label}} {{form.college}}<br>
40                  {{form.skill1.label}} {{form.skill1(id='skill1')}}<br>
41                  {{form.skill2.label}} {{form.skill2(id='skill2')}}<br>
42                  {{form.location1.label}} {{form.location1(id='location1')}}<br>
43                  {{form.location2.label}} {{form.location2(id='location2')}}<br>
44                  <div>
45                      <label for="occupation">{{form.occupation.label}}</label>
46                          {{ form.occupation(onchange="showDiv(this)") }}<br>
47                  </div>
48              <div id="student" style="...">
49                  <div>
50                      <label for="student"> {{form.graduation_year.label}}</label>
51                      {{ form.graduation_year }}<br>
52                  </div>
53              </div>
54              <div id="professional" style="...">
55                  <div>
56                      <label for="professional">{{form.experience.label}}</label>
57                      {{ form.experience }}<br>
58                  </div>
```


jobseeker_home.py
jobseeker_update.html
jobseeker_home_page.html

<label for="file">{{form.file.label}}</label>
{{form.file}}

{{ form.submit }}

</form>
{% endif %}
<script>
window.onload = function() {
var location1 = document.getElementById("location1");
var location2 = document.getElementById("location2");
var skill1 = document.getElementById("skill1");
var skill2 = document.getElementById("skill2");
location1.onchange = function () {
location2.value = "";
for (var i = 0; i < location2.options.length; i++) {
if (location1.value === location2.options[i].value) {
location2.options[i].style.display = "none";
} else {
location2.options[i].style.display = "block";
}
}
}
}
function showDiv(select){
html › body › script › Window.onload()

jobseeker_home.py | jobseeker_update.html | jobseeker_home_page.html

```
88      function showDiv(select){
89          var div_student = document.getElementById("student");
90          var div_professional = document.getElementById("professional");
91          if(select.value === "Student"){
92              div_student.style.display = "block";
93              div_professional.style.display = "none";
94          }
95          else{
96              div_student.style.display = "none";
97              div_professional.style.display = "block";
98          }
99      }
100     var input = document.querySelector("input[type='file']");
101     var form = document.querySelector("form");
102     var maxSize = 3 * 1024 * 1024; // 3MB
103     var allowedExtensions = ["pdf"];
104
105 form.addEventListener("submit", function(e) {
106   var file = input.files[0];
107   if (file) {
108     var fileSize = file.size;
109     var fileExtension = file.name.split(".")[1].toLowerCase();
110     if (fileSize > maxSize) {
111       alert("File size should be less than 3MB");
112       e.preventDefault();
113     } else if (allowedExtensions.indexOf(fileExtension) === -1) {
114       alert("Only PDF files are allowed");
115       e.preventDefault();
116     } else {
117       // submit the form
```

Code for Jobseeker apply/jobid

(jobseeker_home.py)

```
new *
@jobseeker_home.route('/apply_job/<jobid>', methods=["POST", "GET"])
def apply_job(jobid):
    jobid = jobid
    db = get_db()
    check_applied = db.execute('select * from applied where applicantid = ?  and jobid = ?',
                               [session['id'], jobid])
    check = check_applied.fetchone()
    if check is not None:
        msg = "You already Applied to this Job."
        return render_template('jobseeker_home_page.html', msg=msg)
    applied = db.execute('insert into applied (applicantid,jobid) values (?,?)', [session['id'], jobid])
    apply = db.commit()
    msg = "Applied to job"
    # TODO: Check that the jobseeker applies only once.If They Tries to attemt second time. Alert them not to do it.
    return render_template('jobseeker_search_name.html', msg=msg)

```

Code for jobseeker_searchjob

(jobseeker_home.py)

```
jobseeker_home.py
@jobseeker_home.route('/searchjob', methods=["GET", "POST"])
def searchjob():
    searchname = SearchJob()
    if request.method == "POST":
        print("Request in POST method")
        title = request.form.get("title")
        print(f"Title: {title}")
        db = get_db()
        user_result = db.execute('select j.skill1 as skill1, j.skill2 as skill2, j.experience as experience,'
                                 'j.location1 as location1,j.remote as remote,'
                                 'j.jobtype as jobtype from jobseeker j where id = ?', [session['id']]).fetchone()
        job_result = db.execute(
            'select j.jobid as jobid,'
            'j.location2 as location2,j.description as description,j.link as link, u.name as name '
            'from jobposting j JOIN user u on u.id = j.createdby '
            ' where title = ? and skill1 = ? and skill2 = ? and location1 = ? and j.type = ? and '
            'experience = ?',
            [title, user_result['skill1'], user_result['skill2'], user_result['location1'],
             user_result['jobtype'], user_result['experience']]).fetchall()
        print(user_result)
        print(job_result)
        print(type(job_result))
        if len(job_result) == 0:
            msg = "No Jobs Found According to your Preferences. Skills, Location, Experience are taken into " \
                  "consideration "
            return render_template('jobseeker_search_name.html', msg=msg)
        else:
            return render_template('jobseeker_search_name.html', job_result=job_result, user_result=user_result,
                                   title=title)
    return render_template('jobseeker_search_name.html', searchname=searchname, session=session)
```

Code for Jobseeker_updatepassword

(jobseeker_home.py)

```python
new *
@jobseeker_home.route('/updatepassword', methods=["POST", "GET"])
def updatepassword():
    passwordupdate = PasswordUpdate()
    db = get_db()
    if request.method == "POST":
        old = request.form.get("old_password")
        new = request.form.get("new_password")
        conform_new = request.form.get("conform_new")
        oldpassword = db.execute("select user.password as password from user where id = ?", [session['id']]).fetchone()
        if oldpassword['password'] != old:
            passmsg = "Old Password Doesn't Match"
            return render_template('jobseeker_home_page.html', passmsg=passmsg)
        if new != conform_new:
            passmsg = "New Passwords Doesn't Match"
            return render_template('jobseeker_home_page.html', passmsg=passmsg)
        try:
            db.execute("update user set password = ? where id = ?", [new, session['id']])
            db.commit()
            passmsg = "Password Updated!"
            return render_template('jobseeker_home_page.html', passmsg=passmsg)
        except Exception as e:
            raise e
    return render_template('jobseeker_home_page.html', passwordupdate=passwordupdate)


```

(jobseeker_home.html)

```
    {% if passwordupdate %}
        <form action="/login/updatepassword" method="post">
            {{passwordupdate.old_password.label}}{{passwordupdate.old_password}}<br>
            {{passwordupdate.new_password.label}}{{passwordupdate.new_password}}<br>
            {{passwordupdate.conform_new.label}}{{passwordupdate.conform_new}}<br>
            {{passwordupdate.submit}}
        </form>
    {% endif %}
```

Code for Jobseeker_logout

(jobseeker_home)

```python
new *
@jobseeker_home.route('/jobseeker_logout', methods=["POST", "GET"])
def jobseeker_logout():
    if 'id' not in session:
        session.clear()
        return redirect('/login')
    print(f"logging out: {session['id']} and username: {session['name']}")
    session.clear()
    return redirect('/')


```

Code for Recruiter_home

(recruiter_home.py)

```
46

    THRINADH43
47  @recruiter_home.route('/recruiter_home_page', methods=["POST", "GET"])
48  def recruiter_home_page():
49      if 'id' not in session or session['type'] != 1:  # To Protect Routes
50          print(session['id'])
51          session.clear()
52          return redirect('/login')
53      db = get_db()
54      username = db.execute('select user.name as name from user where id = ?', [session['id']])
55      name = username.fetchone()
56      session['name'] = name['name']
57      return render_template('recruiter_home_page.html', name=name['name'])
```

(recruiter_home_page.html)

```
27      <div class="container">
28        <div class="row">
29          <div class="col-md-4"></div>
30          <div class="col-md-4">
31            <h4>Hello {{name}}. It's Great to see you here.</h4>
32            <h4>Let's Empower.</h4>
33            <h4>We would help you in this process</h4>
34          </div>
35          <div class="col-md-4"></div>
36        </div>
37      </div>
```

Code for Recruiter Post Job

(recruiter_home.py)

```python
THRINADH43
@recruiter_home.route('/post_job', methods=["POST", "GET"])
def post_job():
    form = Create_Job()
    if 'id' not in session or session['type'] != 1:  # To Protect Routes
        if 'id' in session:
            print(session['id'])
        session.clear()
        return redirect('/login')
    if form.validate_on_submit():
        session['data'] = form.data
        return redirect(url_for('login_blueprint.recruiter_home.confirm'))
    return render_template('job_posting.html', form=form)

```

(job_posting.html)

recruiter_home.py | Job_posting.html | recruiter_home_page.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
      <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    <title>Job Posting Page</title>
    <script>
    window.onload = function() {
        var location1 = document.getElementById("location1");
        var location2 = document.getElementById("location2");
        var skill1 = document.getElementById("skill1");
        var skill2 = document.getElementById("skill2");
        location1.onchange = function () {
            location2.value = "";
            for (var i = 0; i < location2.options.length; i++) {
                if (location1.value === location2.options[i].value) {
                    location2.options[i].style.display = "none";
                } else {
                    location2.options[i].style.display = "block";
                }
            }
        }
    }
    </script>
</head>
<body>
    <nav class="navbar navbar-inverse">
      <div class="container-fluid">
```

html › head › script › Window.onload()

recruiter_home.py | Job_posting.html | recruiter_home_page.html

```
<div class="container-fluid">
  <div class="navbar-header">
    <a class="navbar-brand" href="#">Career Connect</a>
  </div>
  <ul class="nav navbar-nav">
    <li class="active"><a href="/login/recruiter_home_page">Home</a></li>
    <li><a href="/login/post_job">Post a Job</a></li>
    <li><a href="/login/createdbyme">Created by Me</a></li>
      <li><a href="/login/recruiter/updatepassword">Update Password</a></li>
  </ul>
  <ul class="nav navbar-nav navbar-right">
    <li><a href="/login/recruiter_logout"><span class="glyphicon glyphicon-log-in"></span> Logout</a></li>
  </ul>
 </div>
</nav>
<form method="post" action="/login/post_job">
<div class="container">
  <div class="row">
    <div class="col-md-4"></div>
    <div class="col-md-4">
      <label for="{{ form.title.id }}">{{ form.title.label }}</label>
        {{ form.title(id='title') }}
        <br>
        <label for="{{ form.description.id }}">{{ form.description.label }}</label>
        {{ form.description(id='description') }}
        <br>
        <label for="{{ form.jobtype.id }}">{{ form.jobtype.label }}</label>
        {{ form.jobtype(id='jobtype') }}
    </div>
    <div class="col-md-4"></div>
```

html › body › form › div.container › div.row › div.col-md-4

recruiter_home.py | Job_posting.html | recruiter_home_page.html

```
</div>
    {{ form.csrf_token}}
    {{ form.hidden_tag() }}
    <div class="float-left">
        <label for="{{ form.location1.id }}">{{ form.location1.label }}</label>
        {{ form.location1(id='location1') }}
        <br>
        <label for="{{ form.location2.id }}">{{ form.location2.label }}</label>
        {{ form.location2(id='location2') }}
    </div>
    <div class="float-right">
        <label for="{{ form.skill1.id }}">{{ form.skill1.label }}</label>
        {{ form.skill1(id='skill1') }}
        <br>
        <label for="{{ form.skill2.id }}">{{ form.skill2.label }}</label>
        {{ form.skill2(id='skill2') }}
        <br>
        <label for="{{ form.experience.id }}">{{ form.experience.label }}</label>
        {{ form.experience(id='experience') }}
        <br>
        <label for="{{ form.remote.id }}">{{ form.remote.label }}</label>
        {{ form.remote(id='remote') }}
        <br>
        <label for="{{ form.link.id }}">{{ form.link.label }}</label>
        {{ form.link(id='link') }}
        <br>
        <h5>Click on Submit Twice</h5>
        {{form.submit}}
    </div>
</form>
```

html › body › form › div.float-right

Code for Post Job Confirm

(recruiter_home.py)

```
@recruiter_home.route('/post_job/confirm', methods=["GET", "POST"])
def confirm():
    if 'id' not in session or session['type'] != 1:  # To Protect Routes
        print(session['id'])
        session.clear()
        return redirect('/login')
    if 'data' not in session:
        return redirect(url_for('post_job'))
    data = session['data']
    if request.method == "POST":
        db = get_db()
        try:
            db.execute('insert into jobposting (title, type, description, experience, skill1, skill2, location1, '
                       'location2, remote, createdby, link) values (?,?,?,?,?,?,?,?,?,?,?)',
                       [data['title'], data['jobtype'], data['description'], data['experience'], data['skill1'],
                        data['skill2'], data['location1'], data['location2'], data['remote'], session['id'],
                        data['link']])
            db.commit()
            session.pop('data')
        except Exception as e:
            session.pop('data')
            error = str(e)
            if error is not None:
                if 'UNIQUE constraint' in error:
                    error = "Error: Duplicate Entry"
                elif 'NOT NULL constraint' in error:
                    error = "Error: Missing Required Field"
                else:
                    error = "Error: {}".format(error)
```

```
                    error = "Error: {}".format(error)
                return render_template('job_posting_error.html', error=error)
            msg = "Job Posted. Check in Created by me"
            if 'data' not in session:
                print("Session data cleared")
            return render_template('job_posting_conformed.html', msg=msg)
        return render_template('job_posting_confirm.html', data=data)

```

(job_posting_confirm.html)

recruiter_home.py | job_posting_confirm.html | recruiter_home_page.html

```
27      <form method="post" action="/login/post_job/confirm">
28      <div class="container">
29        <div class="row">
30          <div class="col-md-4"></div>
31          <div class="col-md-4">
32            <label for="{{ data.title.id }}" >Title</label>
33            {{ data['title'] }}
34              <br>
35              <label for="{{ data.description.id }}">DESCRIPTION</label>
36              {{ data['description'] }}
37              <br>
38              <label for="{{ data.jobtype.id }}">Type</label>
39              {{ data['jobtype'] }}
40          </div>
41          <div class="col-md-4"></div>
42        </div>
43      </div>
44          <!-- {{ data.csrf_token}} -->
45          <div class="float-left">
46              <label for="{{ data.location1.id }}">Location 1</label>
47              {{ data['location1'] }}
48              <br>
49              <label for="{{ data.location2.id }}">Location 2</label>
50              {{ data['location2'] }}
51          </div>
52          <div class="float-right">
53              <label for="{{ data.skill1.id }}">Skill 1</label>
54              {{ data['skill1'] }}
55              <br>
56              <label for="{{ data.skill2.id }}">Skill 2</label>
```

html › body › form › div.container › div.row › div.col-md-4 › label

```
57                  {{ data['skill2'] }}
58                  <br>
59                  <label for="{{ data.experience.id }}">Experience</label>
60                  {{ data['experience'] }}
61                  <br>
62                  <label for="{{ data.link.id }}">Link</label>
63                  {{ data['link'] }}
64                  <br>
65                  <label for="{{ data.description.id }}">Remote</label>
66                  {{ data['remote'] }}
67                  <h5>Click on Submit Twice</h5>
68                  <input type="submit" value="Review and Confirm">
69              </div>
70      </form>
71  </body>
72  </html>
```

Code for created by me

(recruiter_home.py)

```
THRINADH43
@recruiter_home.route('/createdbyme', methods=["POST", "GET"])
def createdbyme():
    if 'id' not in session or session['type'] != 1:  # To Protect Routes
        print(session['id'])
        session.clear()
        return redirect('/login')
    db = get_db()
    user_result = db.execute(
        'select jobposting.jobid as jobid, jobposting.title as title from jobposting where createdby = ?',
        [session['id']])
    result = user_result.fetchall()
    print(result)
    print(result[0][0])
    print(result[0]['title'])
    print(len(result))
    for i in range(len(result)):
        print(result[i][0])
    # if result is not None:
    return render_template('recruiter_created.html', result=result)
```

Code for Created by me id

(recruiter_home.py)

```
THRINADH43
@recruiter_home.route('/createbyme/<jobid>', methods=["POST", "GET"])
def createdbymeid(jobid):
    job_id = jobid
    if 'id' not in session or session['type'] != 1:  # To Protect Routes
        print(session['id'])
        session.clear()
        return redirect('/login')
    db = get_db()
    user_result = db.execute('select  jobposting.title as title, jobposting.type as type, jobposting.description as '
                             'description,jobposting.experience as experience,jobposting.skill1 as skill1,'
                             '' 'jobposting.skill2 as skill2, jobposting.location1 as location1,jobposting.location2 '
                             'as location2, jobposting.remote as remote,jobposting.createdby as createdby,'
                             'jobposting.link as link from jobposting where jobid = ?', [job_id])
    result = user_result.fetchone()
    print(type(result))
    print(result[0])
    print(result)
    print(result['title'])
    session['jobid'] = jobid
    return render_template('recruiter_job_created.html', result=result)
```

Code for Eligible

(recruiter_home.py)

```
THRINADH43
@recruiter_home.route('/eligible', methods=["GET", "POST"])
def eligible():
    db = get_db()
    user_result = db.execute('select  jobposting.title as title, jobposting.type as type, jobposting.description as '
                             'description,jobposting.experience as experience,jobposting.skill1 as skill1,'
                             '' 'jobposting.skill2 as skill2, jobposting.location1 as location1,jobposting.location2 '
                             'as location2, jobposting.remote as remote,jobposting.createdby as createdby,'
                             'jobposting.link as link from jobposting where jobid = ?', [session['jobid']])
    result = user_result.fetchone()
    search_result = db.execute('select jobseeker.id as id,jobseeker.skill1 as skill1,jobseeker.skill2 as skill2,'
                               'jobseeker.experience as experience,jobseeker.bio as bio from jobseeker where skill1 = '
                               '? and skill2 = ? and '
                               'location1 = ? and location2 = ? and jobtype = ?  and remote = ?', [result['skill1'],
                                                                                                   result['skill2'],
                                                                                                   result['location1'],
                                                                                                   result['location2'],
                                                                                                   result['type'],
                                                                                                   result['remote']])
    jobresult = search_result.fetchall()
    print(jobresult)
    if len(jobresult) == 0:
        no_result = "No Suitable Candidates Found. We are sorry for that. Please try after few days."
        return render_template('recruiter_job_created.html', no_result=no_result)
    return render_template('recruiter_job_created.html', jobresult=jobresult)
```

Code for Notification

(recruiter_home.py)

```
THRINADH43
@recruiter_home.route('/notify/<id>', methods=["POST", "GET"])
def notify(id):
    db = get_db()
    jobseeker_details = db.execute('select user.name as name,user.mail as mail from user where id = ?', [id])
    jobseeker = jobseeker_details.fetchone()
    jobdetails = db.execute('select  jobposting.title as title, jobposting.type as type, jobposting.description as '
                            'description,jobposting.experience as experience,jobposting.skill1 as skill1,'
                            '' 'jobposting.skill2 as skill2, jobposting.location1 as location1,jobposting.location2 '
                            'as location2, jobposting.remote as remote,jobposting.createdby as createdby,'
                            'jobposting.link as link from jobposting where jobid = ?', [session['jobid']])
    job = jobdetails.fetchone()
    msg = Message("Congratulations, You have been notified by recruiters.", sender="thrinadh.manubothu@gmail.com",
                  recipients=[jobseeker['mail']])
    msg.body = f"Hello {jobseeker['name']}.You have been noticied for the following job. Jobid: {session['jobid']}\n" \
               f"Title:{job['title']}\nDescription:{job['description']}\nLink:{job['link']}\n" \
               f"Please Look with the Given Job id and Apply for it.\n" \
               f"All the Best!"
    with current_app.app_context():
        current_app.extensions['mail'].send(msg)
    message = "Mail Sent"
    # TODO: Send Alert to the recruiter after the mail has been successfully posted.
    return redirect(url_for('login_blueprint.recruiter_home.eligible'))
```

Code for Eligible ID

(recruiter_home.py)

```
THRINADH43
@recruiter_home.route('/eligible/<id>', methods=["POST", "GET"])
def eligibleid(id):
    db = get_db()
    userdata = db.execute('select  jobseeker.bio as bio, jobseeker.college as college,'
                          'jobseeker.githublink as githublink, jobseeker.linkedinlink as linkedin, jobseeker.resume as resume '
                          'from jobseeker where id = ?', [id])
    user = userdata.fetchone()
    return render_template('recruiter_job_created.html', user=user)
```

Code for Appliedby

(recruiter_home.py)

```
THRINADH43
@recruiter_home.route('/appliedby', methods=["POST", "GET"])
def appliedby():
    db = get_db()
    apply_list = db.execute(
        'select a.applicantid as applicantid, j.bio as bio, j.jobtype as jobtype, j.linkedinlink as linkedin,'
        'j.githublink as github, j.college as college, j.skill1 as skill1,'
        ' j.skill2 as skill2,j.experience as experience, j.resume as resume'
        ' from applied a join jobseeker j on j.id = a.applicantid where a.jobid = ?', [session['jobid']])
    apply = apply_list.fetchall()
    return render_template('recruiter_job_created.html', apply=apply)
```

Code for Recruiter_updatePassword

```
THRINADH43
@recruiter_home.route('/recruiter/updatepassword', methods=["POST", "GET"])
def updatepassword():
    name = session['name']
    passwordupdate = PasswordUpdate()
    db = get_db()
    if request.method == "POST":
        old = request.form.get("old_password")
        new = request.form.get("new_password")
        conform_new = request.form.get("conform_new")
        oldpassword = db.execute("select user.password as password from user where id = ?", [session['id']]).fetchone()
        if oldpassword['password'] != old:
            passmsg = "Old Password Doesn't Match"
            return render_template('recruiter_home_page.html', passmsg=passmsg, name=name)
        if new != conform_new:
            passmsg = "New Passwords Doesn't Match"
            return render_template('recruiter_home_page.html', passmsg=passmsg, name=name)
        try:
            db.execute("update user set password = ? where id = ?", [new, session['id']])
            db.commit()
            passmsg = "Password Updated!"
            return render_template('recruiter_home_page.html', passmsg=passmsg, name=name)
        except Exception as e:
            raise e
    return render_template('recruiter_home_page.html', passwordupdate=passwordupdate, name=name)
```

Code for Download Recruiter

(recruiter_home.py)

```
THRINADH43
@recruiter_home.route('/download/<filename>', methods=["GET", "POST"])
def download(filename):
    try:
        return send_file(f"./resumes/{filename}", as_attachment=True)
    except Exception as e:
        return str(e)
```

Code for Recruiter Logout

(recruiter_home.py)

```
THRINADH43
@recruiter_home.route('/recruiter_logout', methods=["POST", "GET"])
def recruiter_logout():
    if 'id' not in session or session['type'] != 1:  # To Protect Routes
        print(session['id'])
        session.clear()
        return redirect('/login')
    print(f"Logged in User Id: {session.get('id')}")
    session.clear()
    return redirect('/')
```

Databases Created in the Application

User table: To Store Details for login.

	cid	name	type	notnull	dflt_value	pk
1	0	id	INTEGER	0	<null>	1
2	1	name	char(50)	1	<null>	0
3	2	mail	char(50)	1	<null>	0
4	3	password	char(50)	1	<null>	0
5	4	type	INT	1	<null>	0
6	5	company	char(50)	0	<null>	0

JobPosting table: To Store Posted Job Data from the recruiters

Output | Result 2

12 rows

	cid	name	type	notnull	dflt_value	pk
1	0	jobid	INTEGER	0	<null>	1
2	1	title	char(50)	1	<null>	0
3	2	type	char(20)	1	<null>	0
4	3	description	char(5000)	1	<null>	0
5	4	experience	char(10)	1	<null>	0
6	5	skill1	char(20)	1	<null>	0
7	6	skill2	char(20)	0	<null>	0
8	7	location1	char(50)	1	<null>	0
9	8	location2	char(50)	0	<null>	0
10	9	remote	char(5)	1	<null>	0
11	10	createdby	INT	0	<null>	0
12	11	link	char(3000)	1	<null>	0

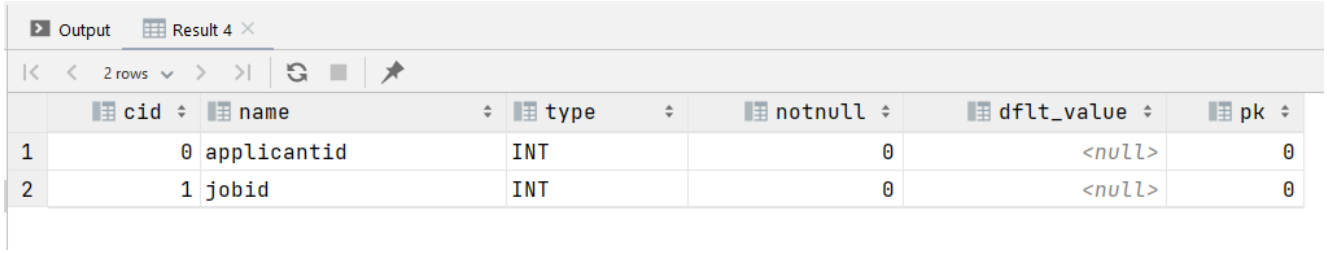
JobSeeker Table: To Store JobSeeker Data

Output | Result 3

16 rows

	cid	name	type	notnull	dflt_value	pk
1	0	id	INT	0	<null>	1
2	1	bio	char(1500)	0	<null>	0
3	2	jobtype	char(25)	0	<null>	0
4	3	skill1	char(25)	0	<null>	0
5	4	skill2	char(25)	0	<null>	0
6	5	location1	char(25)	0	<null>	0
7	6	location2	char(25)	0	<null>	0
8	7	remote	char(5)	0	<null>	0
9	8	occupation	char(25)	0	<null>	0
10	9	graduation	char(4)	0	<null>	0
11	10	experience	char(10)	0	<null>	0
12	11	previousexperience	char(100)	0	<null>	0
13	12	resume	char(150)	0	<null>	0
14	13	githublink	char(2500)	0	<null>	0
15	14	linkedinlink	char(2500)	0	<null>	0
16	15	college	char(50)	0	<null>	0

Applied Table. To Store Applicant id and Jobid



Output | Result 4

2 rows

	cid	name	type	notnull	dflt_value	pk
1	0	applicantid	INT	0	<null>	0
2	1	jobid	INT	0	<null>	0

15. TESTING

The main purpose of testing CareerConnect is to ensure that all the given functionality run smoothly and remains protected.

For Jobseeker:

1. Verify login with valid and invalid data
2. Verify view profile
3. Verify searching for a job with valid and invalid job id
4. Verify applying to the job
5. Verify Change Password
6. Verify Updating the account
7. Verify Downloading Resume
8. Verify Changing Password with Forgot Password.

For Recruiters:

1. Verify creating an account without professional mail
2. Verify creating a job
3. Verify getting the job details
4. Verify getting eligible list
5. Verify getting applied list
6. Verify sending notifications to the user.
7. Verify changing password
8. Verify changing password with forgot password.

Test Case No	Functionality to be checked	Actual input	Actual output	Expected output	Status
1	Verify Recruiter login/ Verify jobseeker login	Given valid mail and valid password.	Login Success	Login Success	Success
		Given valid mail and an invalid password	Deny Login	Deny Login	Success
		Given an invalid mail and valid password	Deny Login	Deny Login	Success
		Given an invalid mail and invalid password	Deny Login	Deny Login	Success
2	Verify Forgot password	Given wrong mail	No Change	No change	Success
		Given correct mail and wrong otp	No change	No change	Success
		Given all fields correct	Changed	Changed	Success
		Given correct mail,otp but 2 passwords doesn’t match	No change	No change	Sucesss

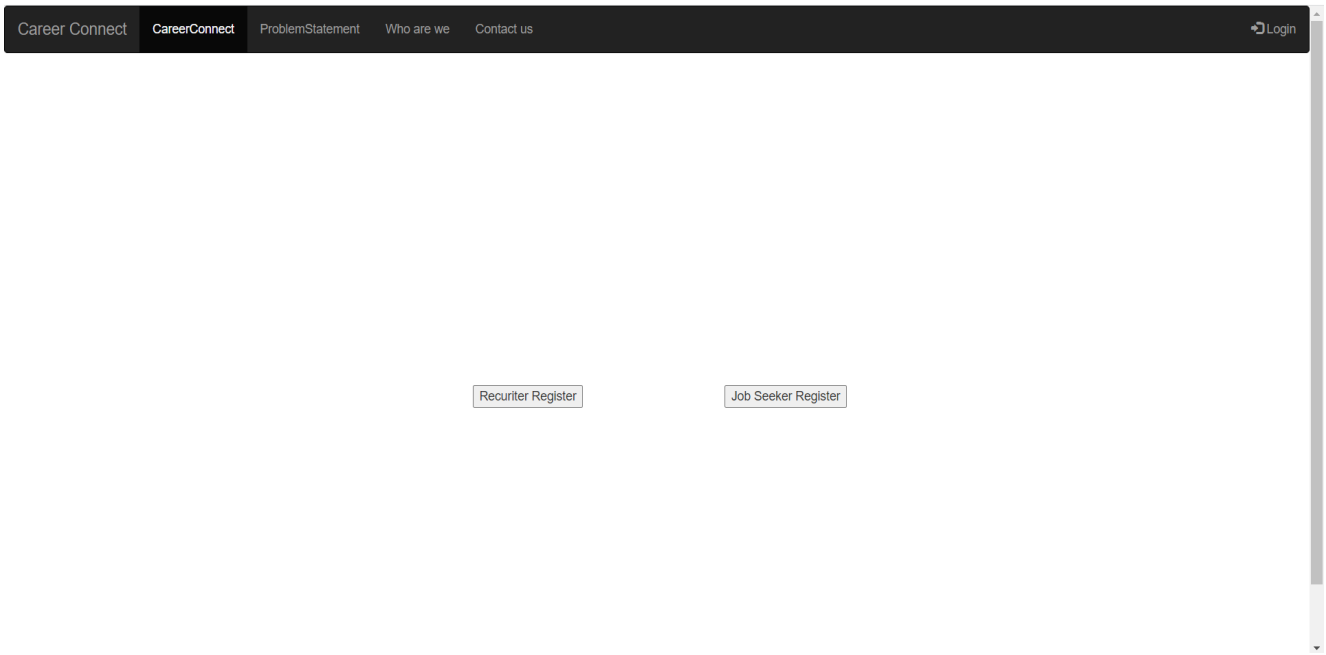
3	Post a Job	Leave out Description	Prompt to enter it	Prompting to enter it	Success
		Fill all details	Save to database	Save to database	Success
4	View Posted Jobs	Clicks on the links to view	Goes to that Job id	Goes to that Job id	success
		Change Job id to some value in url	No job is displayed	No jobs are displayed	Success
5	View Eligible List	Click to get eligible list	Display eligible candidates list	Display eligible candidates list	success
		Clicks to get eligible list	Display message if no one is eligible	Display message if no one is eligible	Success
6	Send Notification	Click to send Notifications	Notification should be sent and display that notification are sent	Notifications are sent but flash messages are not shown on screen.	Partial Success
7	Applied list	Clicks to see applied list	Display applied list	Displaying applied list	Success
		Clicks to see applied list	Display message if no one applied	Displaying message if on one applied	Success
8	Update Password for both recruiters	Give wrong old password	Generate message as wrong old password	Generating message	Success

	and jobseekers	Mismatch new and confirm	Not changed and display message	Not changed and message displayed	Success
		Input all correctly	changed	changed	Success
9	Search JobID	Given correct Job id	Get the job details	Job details appeared	Success
		Given invalid Job Id	Display message	Message Displayed	Success
10	View applied list	Click to view applied list	Display applied list	Applied list displayed	Success
		Click to view applied list	Display message if not applied	Message Displayed	Success
11	Apply to Job	Click to Apply	Apply to Job	Applied to Job	Success
		Click to Re-Apply	Display Message	Message Displayed	Success
12	Update Data	Give all details	Update to database	Database updated	Success
		Leave columns that you don’t want to change	No change to those columns in database	Database not changed to those columns	Success
13	Logout	Click on logout	Logs out the user	User logged out	success

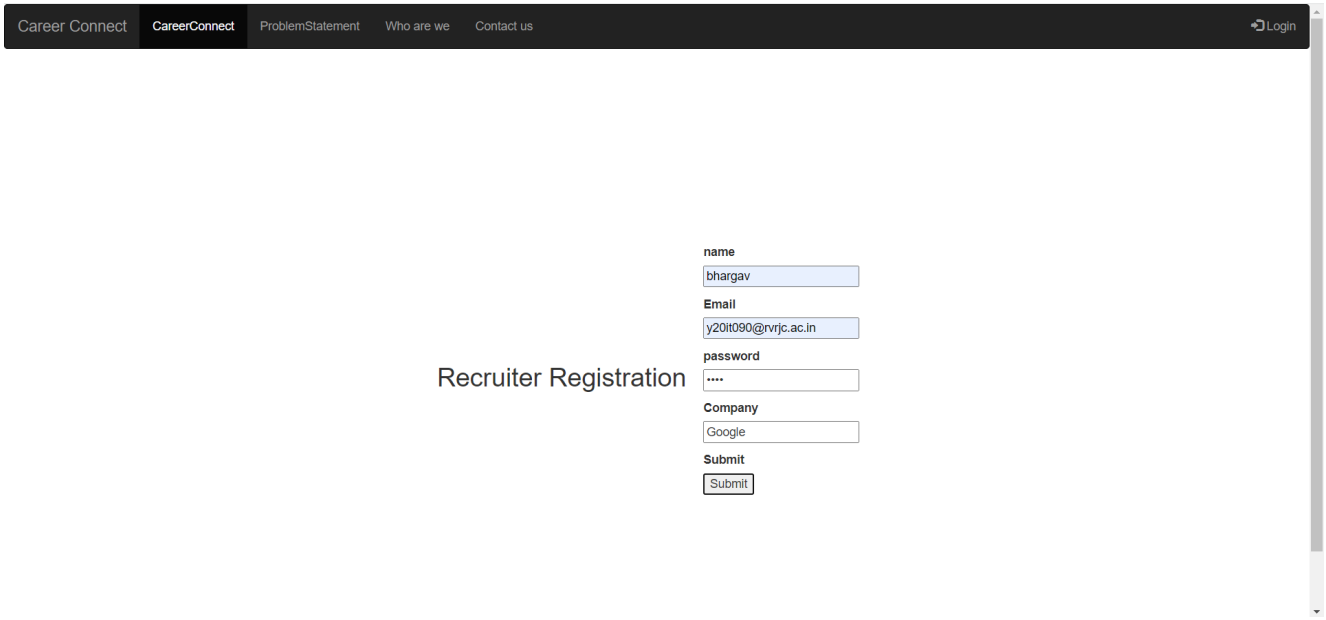
16. IMPLEMENTATION SCREENSHOTS

The following are the runtime Graphical User interfaces developed in the application along with functionality.

Index Page:



Recruiter Register



Before Updating the User table:

Output | main.user

5 rows | Tx: Auto | DDL

	id	name	mail	password	type	company
1	1	venkat	career2@yopmail.com	test	0	<null>
2	2	Thrinadh	career1@yopmail.com	test	0	<null>
3	3	kotireddy	career3@yopmail.com	test	0	<null>
4	4	Chandra	y20it073@rvrjc.ac.in	test	1	anuta
5	7	ajay	y20it074@rvrjc.ac.in	test	1	microsoft

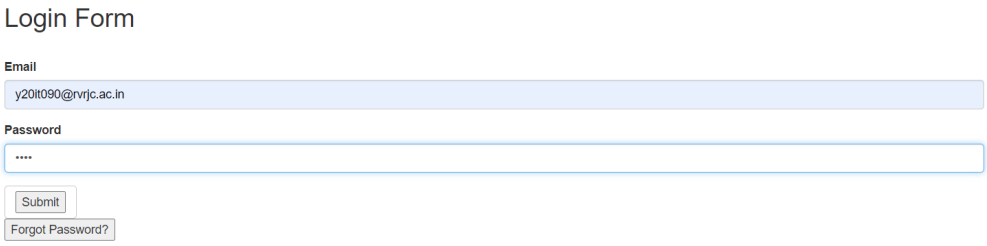
After User Registration and updating the user table:

Output | main.user

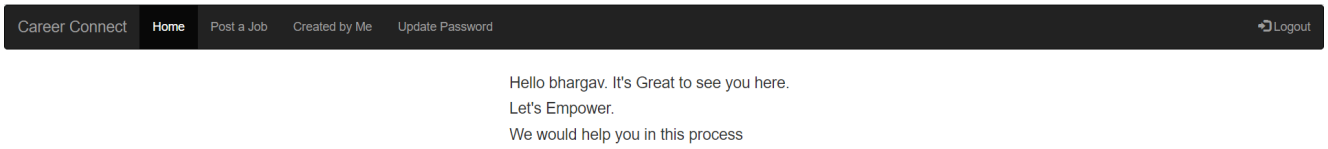
6 rows | Tx: Auto | DDL

	id	name	mail	password	type	company
1	1	venkat	career2@yopmail.com	test	0	<null>
2	2	Thrinadh	career1@yopmail.com	test	0	<null>
3	3	kotireddy	career3@yopmail.com	test	0	<null>
4	4	Chandra	y20it073@rvrjc.ac.in	test	1	anuta
5	7	ajay	y20it074@rvrjc.ac.in	test	1	microsoft
6	8	bhargav	y20it090@rvrjc.ac.in	test	1	google

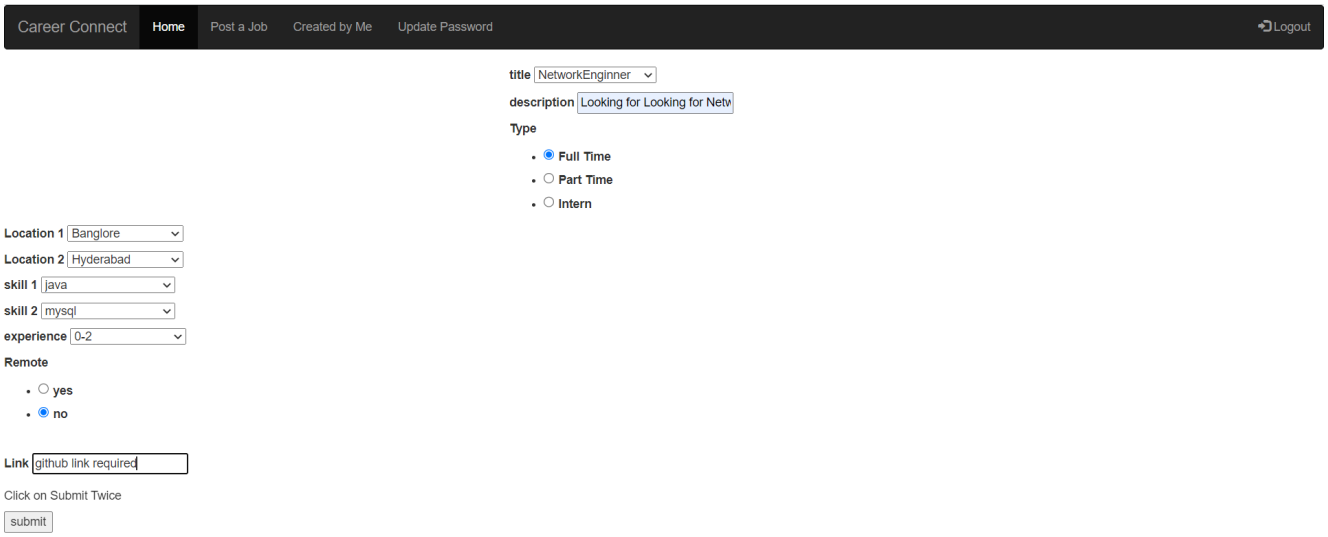
Login Page



Recruiter Home Page:



Recruiter Posting Job:

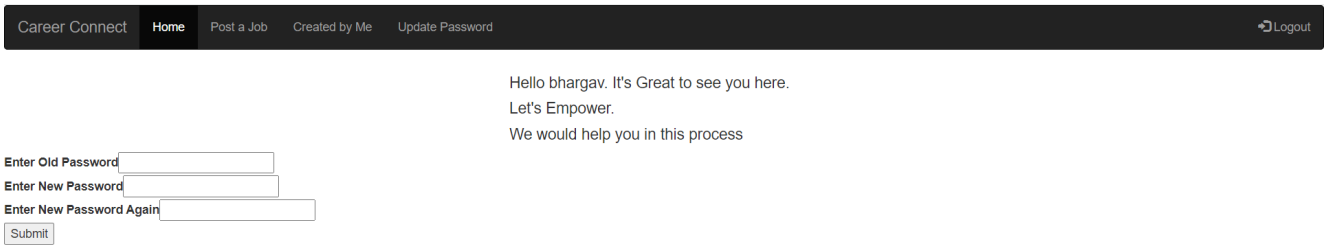


Recruiter Created by me

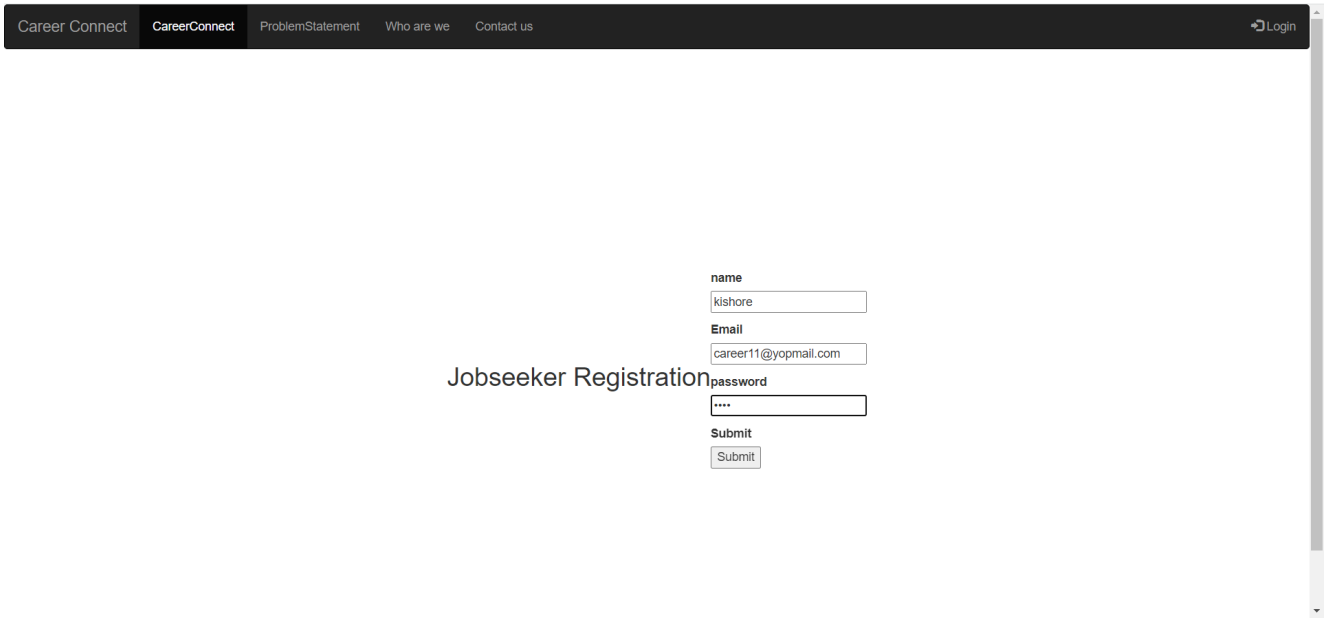
Changes into the Job Posting Database:

	jobid	title	type	description	experience	skill1	skill2	location1	location2
1	1	Python Developer	Full Time	Looking for experienced Python Developer	8+	python	mongodb	Hyderabad	Chennai
2	2	Network Engineer	Full Time	Looking for fresher in Networking	0-2	python	mysql	Hyderabad	Chennai
3	3	Softwaredeveloper	Full Time	Looking for Software Developers	0-2	c++	mysql	Hyderabad	Chennai
4	4	NetworkEnginner	Full Time	Looking for Looking for Network Enginners	0-2	python	mongodb	Hyderabad	Banglore
5	5	Softwaredeveloper	Full Time	Testing the application with title	0-2	python	mysql	Hyderabad	Chennai
6	6	Sr.Software	Full Time	Looking for senior software	0-2	python	mysql	Hyderabad	Chennai
7	7	MLEnginner	Full Time	Looking for ML Engineers	0-2	python	mysql	Hyderabad	Chennai
8	8	NetworkEnginner	Full Time	Looking for Looking for Network Enginners	0-2	java	mysql	Banglore	Hyderabad

Recruiter Update Password:



Job Seeker Registration:



Job Seekers Details Page:

Type

- (•) Full Time
- ( ) Part Time
- ( ) Intern

description I am an Engineering Stude GitHub Link ThrinadhManubothu

Linkedin Link thrinadh-manubothu

College R.V.R & J.C College of Enc

skill 1 python

skill 2 mysql

Location 1 Hyderabad

Location 2 Banglore

I am a Student

Graduation Year 2024

Upload a PDF file Choose File List-of-Word...y-section.pdf

submit

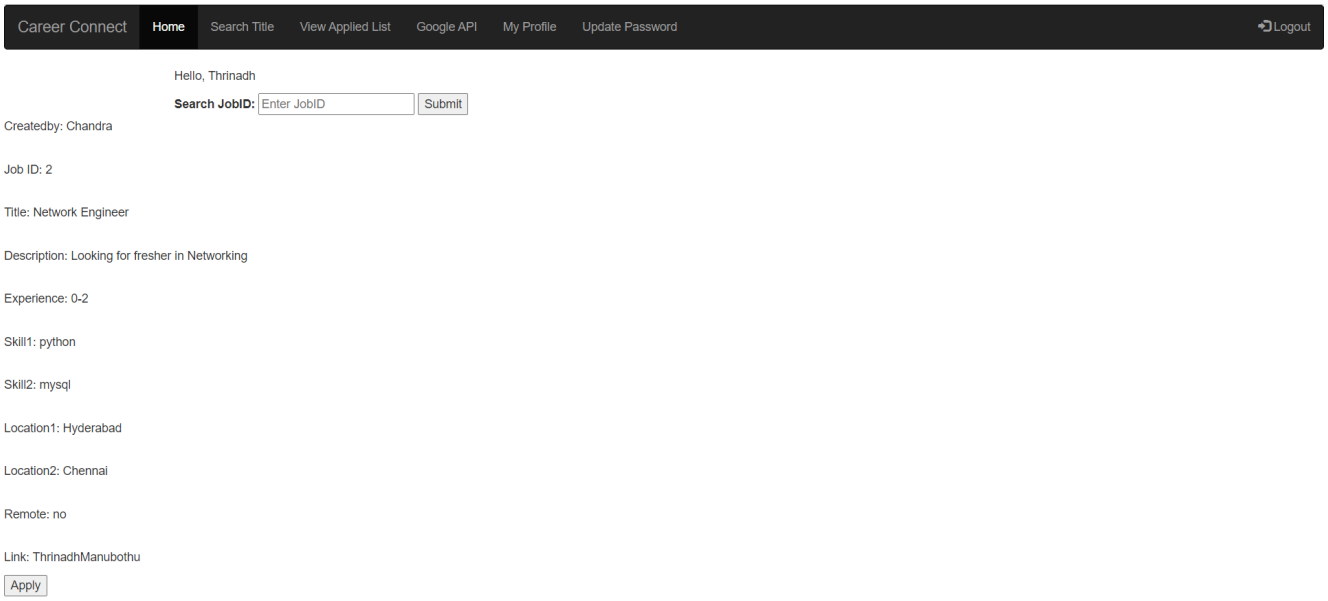
Data Updated in Jobseeker Table:

Output main.jobseeker

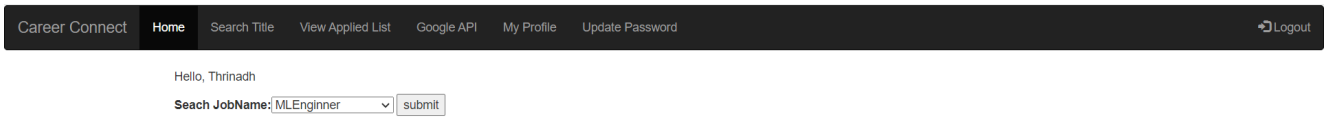
7 rows Tx: Auto DDL CSV

	id	bio	jobtype	skill1	skill2	location1	location2	remote	occupation	graduation	experience	pr
1	1	I am in 3rd year of B.tech	Full Time	python	mysql	Hyderabad	Chennai	no	Student	2024	0-2	<null
2	2	I am learning Networking	Full Time	python	mysql	Hyderabad	Chennai	no	Student	2024	0-2	<null
3	3	Looking for Hacking Jobs	Intern	java	mongodb	Banglore	Chennai	no	Professional	None	0-2	<null
4	5	I am learning Networking	Full Time	None	None	None	None	no	None	None	0-2	<null
5	9	I am in my B.Tech	Full Time	python	mysql	Banglore	Hyderabad	no	Professional	None	0-2	<null
6	10	Testing the application	Full Time	java	mongodb	Hyderabad	Chennai	no	Professional	None	0-2	<null
7	11	I am an Engineering Student	Full Time	python	mysql	Hyderabad	Banglore	no	Student	2024	0-2	<null

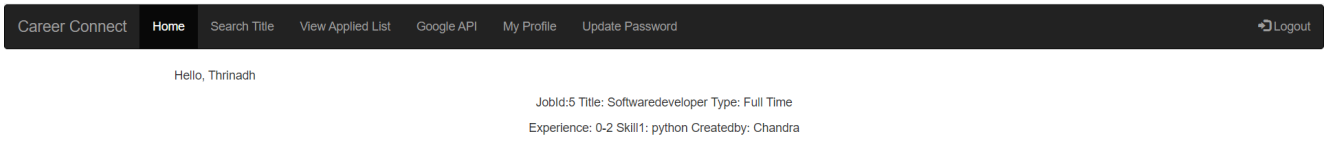
Job Serach with ID:



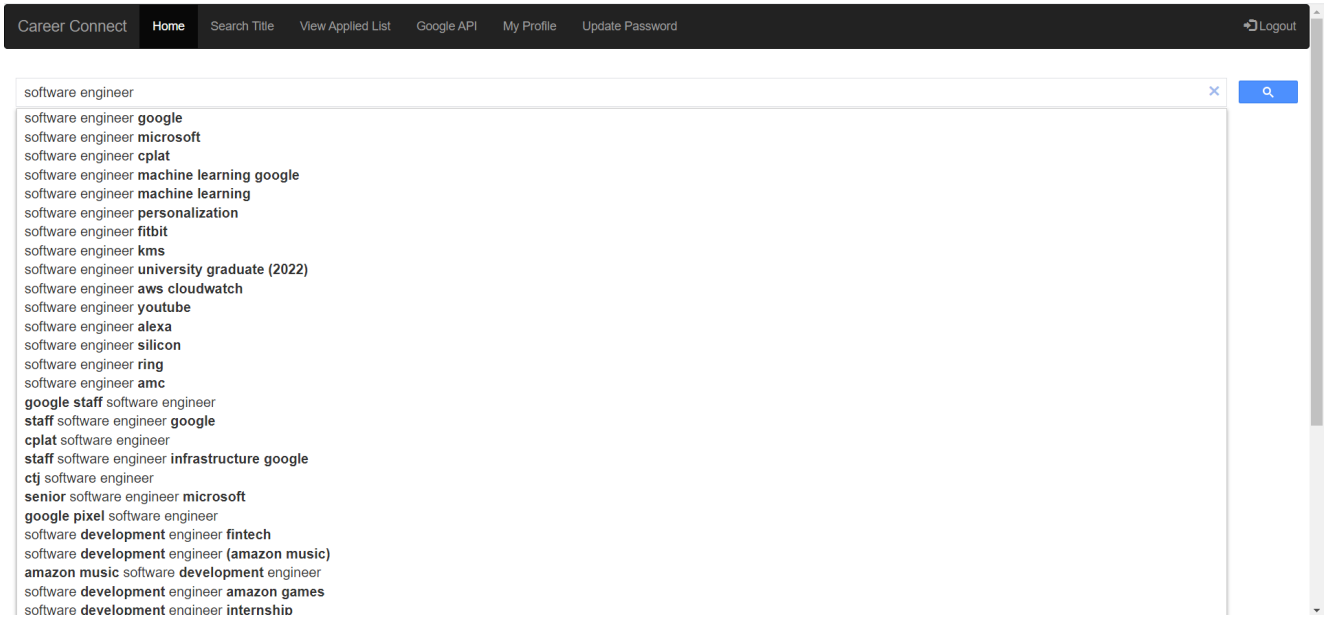
Job Search with Title:



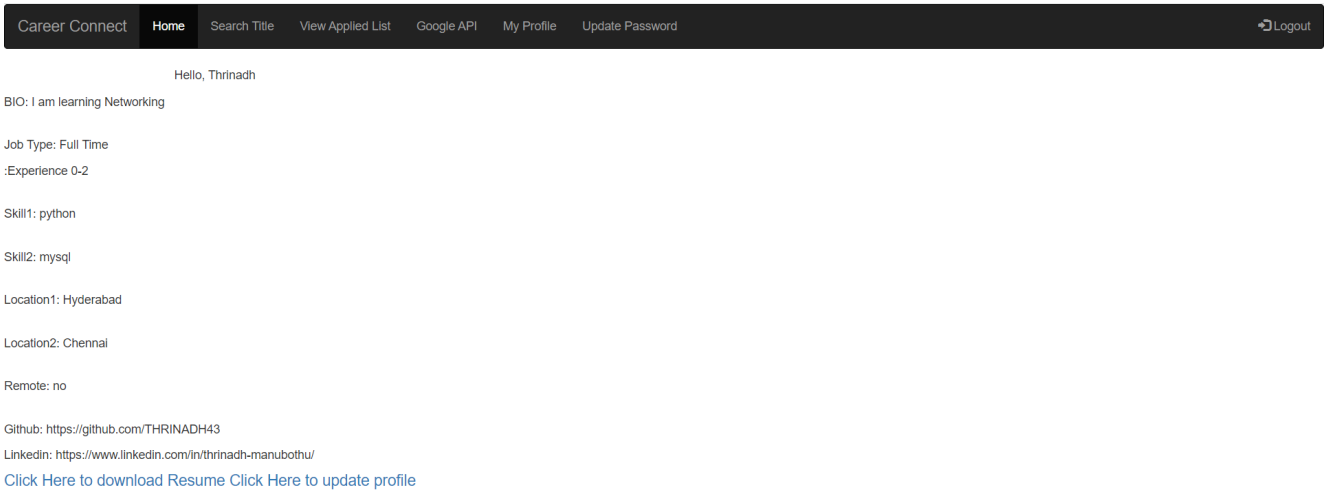
Job Seeker Applied List:



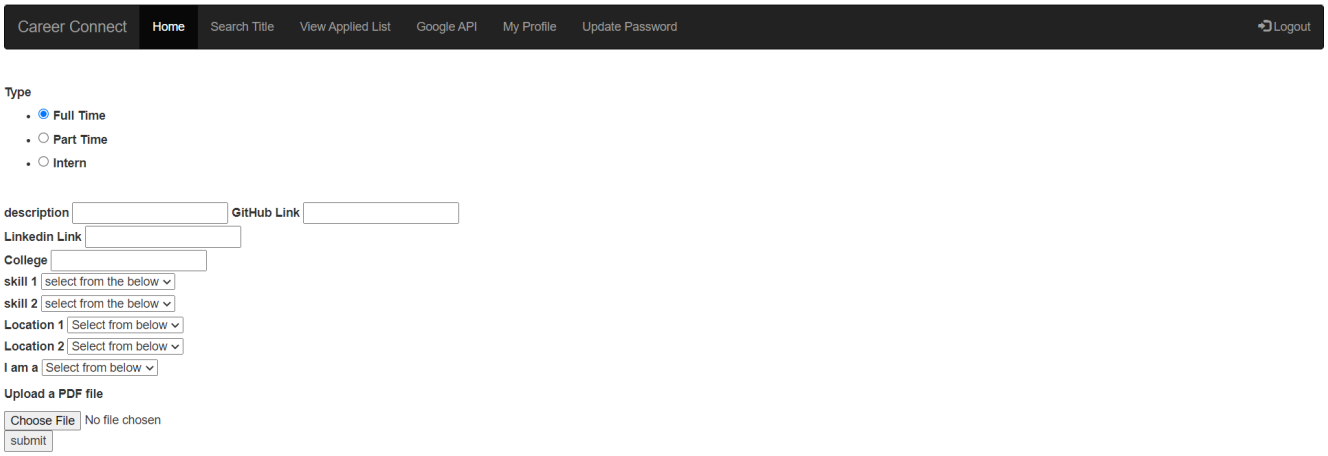
Jobseeker Search with Google API:



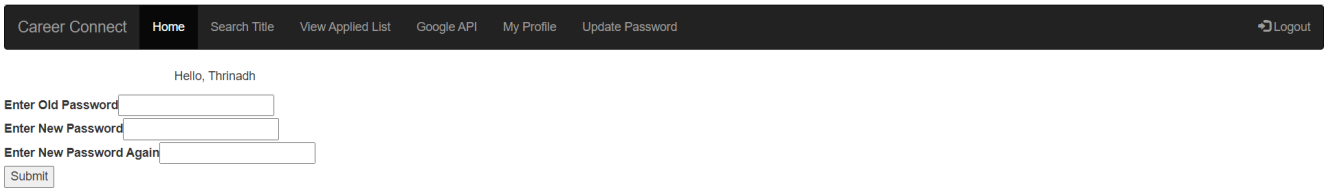
Jobseeker My Profile:



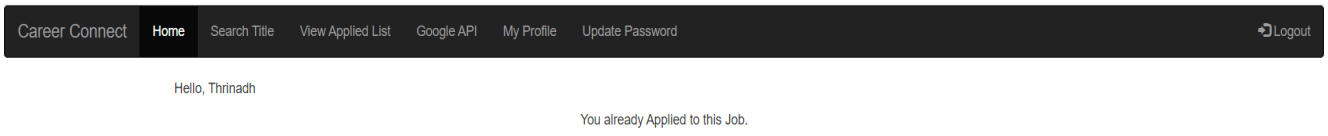
Jobseeker Update profile:



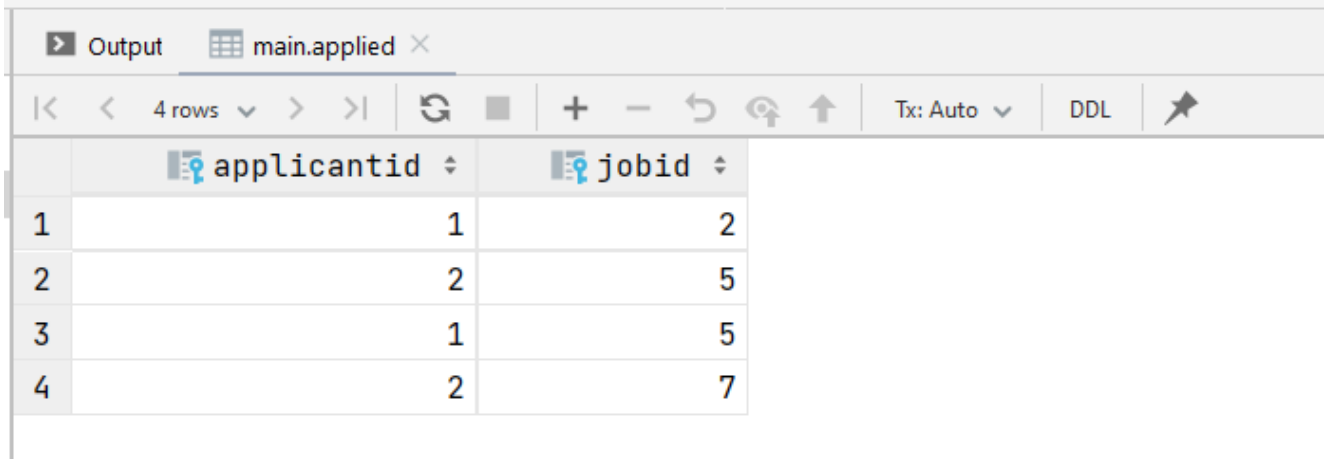
Jobseeker Update Password:



Jobseeker Applying to Job.



Changes to the Applied Table.



	applicantid	jobid
1	1	2
2	2	5
3	1	5
4	2	7

17. CONCLUSION

CareerConnect is the system that helps to seamlessly apply to Jobs and Post Jobs.

By allowing only authorized people to post Jobs. CareerConnect would reduce the chances of spamming job seekers and wasting their precious time.

As Recruiters were also able to find the correct talent at the correct time. This could help the organizations in terms of both profit and market.

References:

- Anthony Herbert “The Ultimate Flask Course”, Packt Publishing
- Meta Brains “Bootstrap 5 Course: Build Responsive websites like a pro”, Packt Publishing
- Joel Perras “ Flask Blueprints”, Packt Publishing
- Jay A. Kreibich “Using SQLite, O'Reilly Media, Inc.
- Software Engineering Resources : - www.rspa.com/spi/