TITLE OF THE PROJECT

SafeAgency.com

Submitted By

Ashiful Islam Prince

(Id: 171-35-2031)

Supervised By

Asif Khan Shakir

Lecturer (Senior Scale)

Faculty of Science and Information Technology

Department of Software Engineering

Daffodil International University

FALL-2020

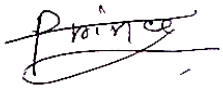
A project (SWE 431) submitted in fulfillment of
the requirements for the degree of BSc in
Software Engineering

APPROVAL

This project titled **"SafeAgency.com"** submitted by **Ashiful Islam Prince, ID: 171-35-2031** to the department of Software Engineering, Daffodil International University has been accepted as satisfaction for the partial fulfillments for the requirements of the degree of B.Sc. in Software Engineering.

Supervised By

Asif Khan Shakir
Lecturer (Senior Scale)
Department of Software Engineering
Daffodil International University



Submitted By

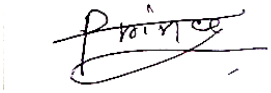
Ashiful Islam Prince
ID: 171-35-2031
Department of Software Engineering
Daffodil International University

DECLARATION

I, the author, hereby declare that the project becomes the property of Daffodil International University (DIU). I give permission to the DIU that they can change in whole or in part for the purpose of research or academic exchange only.

Supervised By

Asif Khan Shakir
Lecturer (Senior Scale)
Department of Software Engineering
Daffodil International University



Submitted By

Ashiful Islam Prince
ID: 171-35-2031
Department of Software Engineering
Daffodil International University

ACKNOWLEDGEMENT

Firstly, I want to thank the almighty Allah who gives me the opportunity to complete the project by keeping me continuously healthy in terms of physically and mentally. I want to thank my parents who are always inspiring me to continue my studies as well as to complete the project during the difficult situation in the covid-19. I want to thank all of the teacher who taught me to achieve the required skills to complete this project. I want to thank my well-wishers as well as friends who always inspiring me that I can complete the project during this Covid-19. Finally, I want to thank my supervisor Asif Khan Shakir sir who helped me to complete this project by giving continuously proper guidelines, feedback through the online meeting. Specially he helped me to continue my studies as well as to complete the project by giving mentally support during this difficult situation.

ABSTRACT

The project I have completed called ‘SafeAgency.com’. This is a web application. Most of the people of our country have faced difficult situation when they are going to land a job at abroad. They are misleading by the bad agent. As a result, they lost their valuable money. Sometimes they lost their lives. Apart from that I have been completed this project for them which ensures the safety and user- friendliness. There have been four types of user in this system. Employer, Agent, Super Admin and Candidate. This web application will make the outstanding interaction between these users where employer is going to land a job, admin is going to approved the current job, candidate can see the job posts if he can then he is going to apply for a job. If he can’t then he can find the agent specialist who offered different packages for him. He can land a job through the expected agent. Admin is going to ensure the safety by filtering agent and employer’s accounts.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DIU = Daffodil International University

SWE= Software Engineering Department

1. CHAPTER-01 INTRODUCTION

1.1. Project Overview

SafeAgency.com is a web application. This is developed for the purposes to assist the people who are willing to land a job at abroad. This web application will make the outstanding interaction between the employer, agent, candidate and the super admin to pursue the vision of this project. Employer will manage the job, admin will approve the job, candidate will see the job as well as apply the job. If he is not able to do these stuffs, he or she can choose the agent specialist by seeing their profile and packages. He can hire the agent for the completion of their expectation. Agent can manage their profile, can make the interaction between candidate and employer through the live chat. Finally, the candidate is going to get the safe service. Because the admin is filtering and monitoring the agent account, employer account as well.

1.2. Project Purpose

1.2.1. Background

This project is implemented with the purposes of provide safe and easiest service to the candidate who are willing to land their dream job at abroad. The outcome of this project deserves as successful interaction between agent, candidate, employer.

1.2.2. Benefits and Beneficiaries

- Candidate can land their dream job at abroad
- Candidate can get the safe services from this system
- Candidate can get the easiest platform to get their desired job
- Employer can find their desired candidate
- Agent can earn money by providing safe services to the candidate

1.2.3. Goals

- To ensure the safe service
- To ensure the user-friendly service

1.3. Stakeholders

- Employer
- Candidate
- Agent
- Super Admin

1.4. Proposed System Model (Block Diagram)

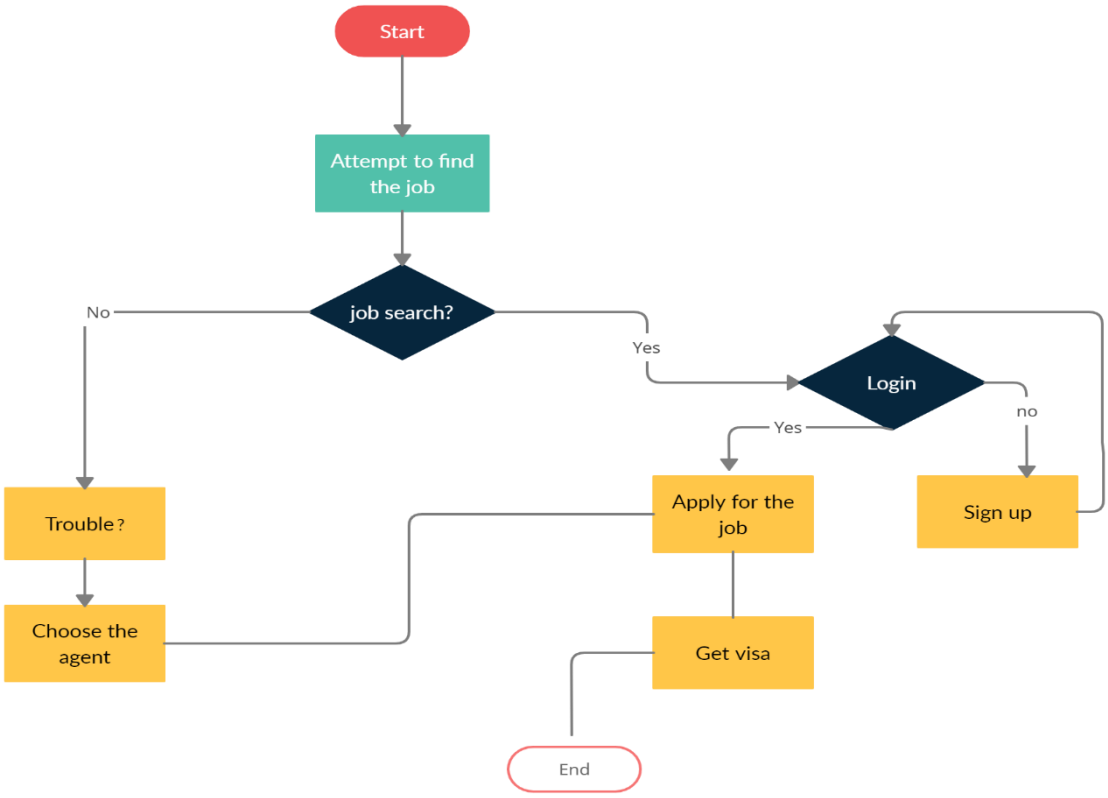


Figure 1. 1: Overview of proposed system model

1.5. Project Schedule

1.5.1. Gant Chart

Figure 1. 2: Gant Chart

Project Proposal	Ashiful Islam Prince	5 Days
Software Requirement Specification	Ashiful Islam Prince	12 Days
Software Design	Ashiful Islam Prince	19 Days
Coding	Ashiful Islam Prince	43 Days
Software Testing	Ashiful Islam Prince	22 Days
Project Finalization	Ashiful Islam Prince	22 Days

Figure 1. 3: Overview of work distribution

2. CHAPTER-02 Software Requirement Specification

2.1. Functional Requirements

In below I discussed all about of functional requirements of this system in detail.

Table 2. 1: Functional Requirements

ID	NAME	DESCRIPTION	STACKHOLDER
FRQ-01	**Can Create Applicant Profile**	This module is submitted for applicant who give some crucial information to this form. Like educational, experience, additional, personal information which can be update in further time when it will be needed. Here applicant is directly involved. Applicant profile can be seen by the employer.	Candidate, Employer
FRQ-02	**Can Show Posted Job**	An applicant can show the posted job which is posted by several employer.	Candidate
FRQ-03	**Can Search Job by the Job Category or Job Type**	An applicant can search his dream job by typing job category or Job Type on the search button.	Candidate
FRQ-04	**Can Apply for Job**	An applicant applies for his dream job. But when he would go for apply job, one thing should be justified that is-applicant must be logged in Employer can show the applied job for particular applicant.	Candidate, Employer
FRQ-05	**Can Show Job Confirmation**	After job application the applicant may be selected by the particular employer. Sometimes probably not	Candidate, Employer

		selected. But an email will be sent to the particular applicant which is considered as job confirmation.	
FRQ-06	**Update Applicant Profile**	An applicant can easily update his career resume when it is necessary.	Candidate, Employer
FRQ-07	**Can create employer profile**	An employer can create his profile included name, email, password, company name and details. If an applicant wants to know the employer profile then he can do it easily by clicking show employer profile.	Employer, Applicant
FRQ-08	**Can see applied job**	An employer can easily show the applied job which comes from employers.	Employer, Applicant
FRQ-09	**Can post job category**	An employer can post job category included job category name, category type etc. which will be seen by the applicant.	Employer, Applicant
FRQ-10	**Can post job type**	An employer can post job type included job type name. like government organization, software farm. which will be seen by the applicant.	Employer, Applicant
FRQ-11	**Can update job category**	An employer update job category when it is needed.	Employer, Applicant
FRQ-12	**Can update job type**	An employer can update job type when it is needed.	Employer, Applicant

FRQ-13	**Can delete job responsibilities, job type and category**	An employer can delete job category, type and responsibilities.	Employer, Applicant
FRQ-14	**Can see applicant and employer profile**	A super admin can show applicant as well as employer profile.	Super admin
FRQ-15	**Can show employer and applicant profile**	A super admin can show employer and applicant profile.	Super Admin Employer, Applicant
FRQ-16	**Manage Profile**	An agent can manage their profile by creating, updating and deleting their profile	Agent
FRQ-17	**Live Chat**	An agent can make their communication with the candidate and an employer	Agent, Candidate, Employer
FRQ-18	**Can approve hiring requests**	An agent can accept or reject the hire request which comes from the candidate	Agent, Candidate
FRQ-19	**Can send hiring requests**	An agent can send the hiring request to the employer	Agent, Employer
FRQ-20	**Can see approved lists**	An agent can see the approved list	Agent, Candidate
FRQ-21	**Can manage services**	An agent can add, update, delete services based on the packages	Agent,
FRQ-22	**Log in**	Every user can log in	Agent, Candidate, Employer, Super Admin
FRQ-23	**Registration**	Every user can register	Agent, Candidate, Employer, Super Admin

2.2. Non-Functional Requirements

In below I discussed all about of functional requirements of this system in detail.

Table 2. 2: Non-Functional Requirements

ID	Name	Description	Non-Functional Requirement	Priority
NFR-01	Security	Developed multi-authentication module in this system which gives strong security when someone tries to get the access of the system	Non-Functional	High
NFR-02	Availability	This system is available for the 24 hours in a day	Non-Functional	Medium
NFR-03	Accuracy	This system has been developed with the concern of highest accuracy.	Non-Functional	High
NFR-04	Maintenance	This system provides services to the candidate as user-friendliness. They can navigate easily to access the services	Non-Functional	Medium

2.3. Performance Requirements

2.3.1. Speed and Latency Requirements

- Required data set has been maintained through the MYSQL using Laravel artisan command like php artisan migrate
- Sometimes, I have been performed the rollback –step=1 operation at the time of updating of the last migration file which gives the outstanding speed to maintain the databases

2.3.2. Precisions or Accuracy Requirements

- This system has been developed with the concern of highest accuracy. All of the user of this system can navigate and find the result as their expectation
- When they are going to change their information then they can easily do this without any types of hassles.

2.3.3. Capacity Requirements

- This system has the capability to handle diverse types of user at the same time
- Employer can make the communication with candidate and employer by using the live chat system. No problem occurs at the time of handling lot of users.

2.4. Dependability Requirements

2.4.1. Reliability Requirements

- When the user is going to manage their profile, they must be logged in to the system.
- Super admin is going to filter the agent and employer. If see that something is wrong then he can reject the registration requests.
- Agent is going to check whether the request is approved or not when he makes the request to the employer.
- Agent is going to check whether the candidate pay the amount or not
- Candidate can navigate the agent profile if they do not capable to complete the job process

2.4.2. Availability Requirements

- Intended to interact the xampp server for the purposes of exchanging data to the databases as well as handling the request and sent back the result to the client
- Anytime the user can get the access of this system but he must be authenticated for accessing some sensitive module.
- Preferable browser is chrome for this system.

2.5. Maintainability and Supportability Requirements

2.5.1. Maintenance Requirements

- This system is handled by the authority.
- Except the permission of the authority no one can get the access of this system
- Maintenance has been done by the authority
- Permission can be modified anytime

2.5.2. Supportability Requirements

- If the user faces any difficulty on any module then they can give the feedback to our maintenance team
- System can get the alert when something is going wrong.

2.5.3. Adaptability Requirements

Candidates can get a safe and user-friendliness service from this system at any time without facing any kind of difficulty. Because this system has been developed with the highest concern of the user's perspective.

2.5.4. Scalability or Extensibility Requirements

- Scalability has been ensured by the side of developer panel.
- This is developed by taking suggestions from the expert people. They gave the suggestions about how to ensure the scalability module of this system

2.6. Security Requirements

2.6.1. Access Requirements

- Only authenticated user can enter to their panel
- Guests can allow for the lending pages like candidate can view the agent's profile, can view the job, can search job, can subscribe to this website etc. In these cases, they do not need any authentication.
- Employer can not directly complete their registration without the super admin's acknowledgement
- Agent cannot directly complete their registration without the super admin's acknowledgement

2.6.2. Integrity Requirements

- This system has been navigated by the employer, candidate, agent and super admin

2.6.3. Privacy Requirements

- All of the sensitive information of these user's is going to be protected. We are not going to share any kind of user's credentials with any other third person. Because user is our first priority

2.7. Usability and Human Interaction Requirements

Table 2. 3: Usability and Human Interaction Requirements

UH-01	**User Friendly**
Description	This system is user friendly because of easiest features.
Stakeholders	Employer, Candidate, Super Admin, agent

2.7.1. Ease of Use Requirements

The system is easy to use and can easily be understandable.

Table 2. 4: Ease of Use Requirements

UH-02	The system must be usable for candidate with all associate stakeholders.
Description	The system indicates the several possibilities that the candidate has to go on in using the system. The candidates are allowed to undo any of the operation.
Stakeholders	Employer, Candidate, Super Admin

2.7.2. Understand Ability and Politeness Requirements

Table 2. 5: Understand Ability and Politeness Requirements

UH-03	The system has been made by targeting the local people of the Bangladesh.
Description	Since, the target is local people that's why we are trying our best to make this system more understandable. Such as we are implementing the payment gateway called SSL COMMERZE for thinking about payment method like Bkash etc where people can easily pay their most valuable money.
Stakeholders	Employer, Candidate, Super Admin, Agent

2.7.3. Accessibility Requirements

- Login as employer
- Login as candidate
- Login as super admin
- Login as agent
- Logout as employer
- Logout as candidate
- Logout as super admin
- Logout as agent

2.7.4. User Documentation Requirements

Table 2. 6: User Documentation Requirements

UH-03	The system developer documentation specifically
Description	To develop this project, we have specified requirement of applicant documentation. The teams are involved to this project documentation.
Stakeholders	System Developer

2.8. Look and Feel Requirements

The look and feel requirements elaborate the intended spirit, the format, or the style of the job post appearance as well as the profile appearance. These requirements specify the intention of the appearance perspective, and are not a detailed design of a user interface.

2.8.1. Appearance Requirements

It should be clear to the super admin, agent, employer and candidate which fields need to be filled and which can be left blank in this system.

Table 2. 7: Appearance Requirements

LF-01	Labels of mandatory fields must be bold and font ‘times new roman’
Description	Several input field like job type, job category must be bold
Stakeholders	Employer, Super Admin, Candidate

2.8.2. Style Requirements

We are trying to make our web application more interactive as much as possible for making the interaction between candidate, employer, agent and super admin. Specially, we focused on the candidate who is our first priority in this system

Table 2. 8: Style Requirements

LF-02	The style module is one of the highest priorities when focusing on design
Description	To complete this segment, we follow the cascading style sheet and bootstrap framework
Stakeholders	Super Admin

2.9. Operational and Environmental Requirements

This requirement focuses on how the candidate, agent, employer, super admin is going to operate and the navigate the system like online service, including interfaces and interoperability with other systems as well. The requirements establish how well and efficient way has been followed.

2.9.1. Expected Physical Requirements

This system positions offer a fast-paced service environment, where everyone pitches in and participates in all functions and methodology. All sectors require constant physical activity, including standing, walking, reaching and grasping as well. All employer, applicants are expected to climb ladders, bend and kneel to maintain the visual view of this system. In order to best perform these responsibilities, employees must be able to communicate face-to-face and use the live chat system.

2.9.2. The Requirements for Interfacing with Adjacent System

There is no specific interfacing with adjacent system requirements

2.9.3. Release Requirements

There are no specific release requirements but sometimes it’s depends on candidate’s expectation. The project will be released by maintaining version as much as possible.

2.10. Legal Requirements

These requirements consider any violence of rules and regulation and which rules should be followed to maintain this system
There is no violation exists in this system. All the users can use it easily and perform their activity.

2.10.1. Compliance Requirements

There are no specific compliance requirements but sometimes it's depends on customer expectation

2.10.2. Standard Requirements

There are exists several standard requirements

Table 2. 9: Standard Requirements

LF-01	Types of Job and Category
Description	Considered as private organization
Stakeholders	Employer

3. CHAPTER-03 SYSTEM ANALYSIS

3.1. USE CASE

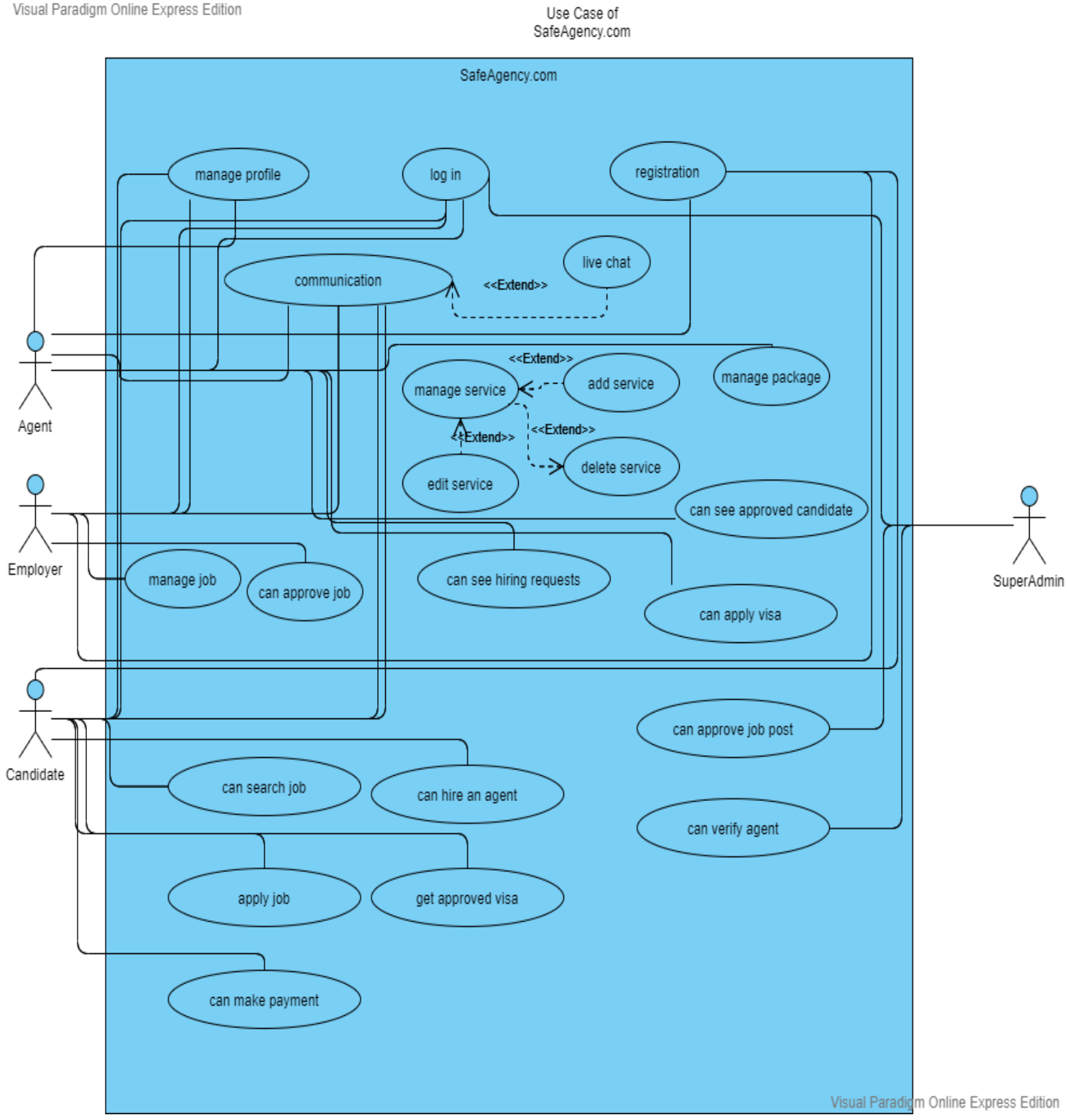


Figure 3. 1: Overview of use case

3.2. USE CASE DESCRIPTION

3.2.1. LOGIN

Table 3. 1: Login

Use Case	Login	
Goal <a longer statement of the goal in context if needed>	Employer, Candidate, Agent, Super Admin can get access to the system through this login system and get some credential resources	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	User can get into the system successfully. After that they perform their action as expected successfully.	
Failed End Condition <the state of the world if goal abandoned>	User cannot get into the system. After that they could not perform their action as expected successfully	
Primary Actors:	Employer, Candidate, Agent	
Secondary Actors	Super Admin	
Trigger <the action upon the system that starts use case>	Login request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User gives the required credentials and hit the login button to get the access into the system
	2	Credentials contains email, password that is required to match. If matches then login is going to be successful and if not then they are going to redirect to their login page again.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	The user enters their credentials including email and password and password. Obviously, password needs to be stored as encrypted process.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including email and password for completion of the whole process.

3.2.2. REGISTRATION

Table 3. 2: Registration

Use Case	Registration	
Goal <a longer statement of the goal in context if needed>	Users are going to give their required information such as skill, interest, bio etc. for completion of the registration process.	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	User gives the required information and hit enter and then the registration is going to be successful without any errors. After that they can get into their login page.	
Failed End Condition <the state of the world if goal abandoned>	User gives the required information and hit enter and then the registration is not completed. The view page appears with an error.	
Primary Actors:	Employer, Candidate, Agent	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Registration request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User gives the required information and hit the login button to get the access into the login system
	2	Credentials may contain skill, bio, contact which varies user to user. If everything is ok then their registration process is going to be successful and if not then they are going to redirect to their sing up page again with the error message.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	The user enters their sensitive and accurate information including skill, bio, interests, about etc. then again hit enter to complete the registration process successfully.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including skill, bio, about, contact etc. for completion of the whole process.

3.2.3. MANAGE PROFILE

Table 3. 3: Manage Profile

Use Case	Manage Profile	
Goal <a longer statement of the goal in context if needed>	Users can manage their profile by creating, updating and deleting their information.	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	User gives all of the required information to create the profile and hit enter. After that he creates his profile successfully and get into their own panel successfully.	
Failed End Condition <the state of the world if goal abandoned>	User gives all of the required information to create the profile and hit enter. After that he could not create his profile successfully and could not get into his own panel successfully.	
Primary Actors:	Employer, Candidate, Agent	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Profile creation request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User gives the required information to create his profile. After he hits enter then the profile has been created successfully, after that he gets into his panel. Later he or she can update and delete his/her profile and can do the others stuffs.
	2	Profile contains in detail information rather than registration. Like skill, education, experience, bio, interests etc. would be the required information.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	Sometimes they can be fall into the trouble if they could not give their accurate and required information. So, he needs to provide accurate and required information to complete this process successfully.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including skill, bio, about, contact etc. for completion of the whole process.

3.2.4. LIVE CHAT

Table 3. 4: Live chat

Use Case	Live Chat	
Goal <a longer statement of the goal in context if needed>	To build the communication a user can use the live chat features	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	User successfully complete his communication through the live chat features	
Failed End Condition <the state of the world if goal abandoned>	User could not complete his communication through the live chat features	
Primary Actors:	Employer, Candidate, Agent	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Live chat request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	When the agent approves the candidate then the live chat option is going to open and it comes with containing the little information of the candidate. After that the agent can make the communication with the candidate and further proceed.
	2	Live chat is going to open if matches the condition like agent must approve the candidate.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	If the agent could not make the live chat properly then the alternative way, he should follow he can make a phone call to the candidate.
Quality Requirements	**Step**	**Requirement**
	1	Requirements must need to full-fil to complete this whole process

3.2.5. MANAGE SERVICES

Table 3. 5: Manage Service

Use Case	Manage Services	
Goal <a longer statement of the goal in context if needed>	Users can manage their services by creating, updating and deleting the services information.	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	User gives all of the required information to create the service and hit enter. After that he creates the service successfully and get into their service post panel successfully.	
Failed End Condition <the state of the world if goal abandoned>	User gives all of the required information to create the service and hit enter. After that he could not create the services successfully and could not get into the service post panel successfully.	
Primary Actors:	Employer	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Service creation request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	User gives the required information to create the service. After he hits enter then the service has been created successfully, after that he gets into the service post panel. Later he or she can update and delete the service and can do the others stuffs.
	2	Service contains in detail information. Like service type, service name, package info would be the required information.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	Sometimes they can be fall into the trouble if they could not give their accurate and required information. So, he needs to provide accurate and required information to complete this process successfully.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including service type, name, package details etc. for completion of the whole process.

3.2.6. CAN APPROVE JOB

Table 3. 6 Can approve job

Use Case	Can approve job	
Goal <a longer statement of the goal in context if needed>	To approve a job after comes the requests from the side of a candidate	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	Approve requests comes successfully and approved successfully	
Failed End Condition <the state of the world if goal abandoned>	Approve request could not come successfully and could not approved successfully	
Primary Actors:	Employer, Candidate	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Approve job request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	When the candidate applies for a job then the employer can see the job. After that he can approve or reject the job by seeing the candidate information
	2	After completion the process the candidate can see his applied job status
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	If the candidate could not able to make a job directly, he can choose an agent and make a hiring request to complete the process apart from him
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including candidate skill, education, experience, contacts etc.

3.2.7. CAN SEE HIRING REQUESTS

Table 3. 7 Can see hiring requests

Use Case	Can see hiring requests	
Goal <a longer statement of the goal in context if needed>	To see the hiring requests after the candidate, make the hiring requests from the home panel.	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	Hiring requests comes successfully and appear successfully	
Failed End Condition <the state of the world if goal abandoned>	Hiring request could not come successfully and could not appear successfully	
Primary Actors:	Agent, Candidate	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Hiring requests request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	When the candidate makes a request for a package then the agent can see the requests. After that he can approve or reject the requests by seeing the candidate information
	2	After completion the process the candidate can see his hiring requests status
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	If the candidate could not able to make a hiring requests directly, he can choose another agent and make a hiring request to complete the process.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including candidate skill, education, experience, contacts etc.

3.2.8. CAN SEE APPROVED CANDIDATE

Table 3. 8: Can see approved candidate

Use Case	Can see approved candidate	
Goal <a longer statement of the goal in context if needed>	To see the approved candidate after the candidate, make the hiring requests from the home panel.	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	Agent is able to see the approved candidate successfully.	
Failed End Condition <the state of the world if goal abandoned>	Agent is not able to see the approved candidate successfully	
Primary Actors:	Agent, Candidate	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	See the approved candidates	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	When the candidate makes a request for a package then the agent can see the requests. After that he can approve or reject the requests by seeing the candidate information and he can see the approved candidates for further proceed.
	2	After completion the process the agent can see the approved candidates in detail.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	If the candidate could not able to see the approved candidates then the preferable suggestion is to make sure that you are an authenticated person
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including candidate skill, education, experience, contacts etc.

3.2.9. CAN APPLY FOR A VISA

Table 3. 9: Can apply for a visa

Use Case	Can apply for a visa	
Goal <a longer statement of the goal in context if needed>	Agents can apply for a visa apart from the candidate.	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	Visa application sent successfully and approved successfully.	
Failed End Condition <the state of the world if goal abandoned>	Visa application not sent successfully and not approved successfully	
Primary Actors:	Candidate, Agent	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Visa application request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	Agent gives the required information to make the application. After he hits enter then the application has been sent successfully, after that he can see the requests status and give the feedback to the candidate using live chat.
	2	After approved the visa he gets the approval letter.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	Sometimes they can be fall into the trouble if they could not give their accurate and required information. So, he needs to provide accurate and required information to complete this process successfully.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including nid, passport, name, contact etc. for completion of the whole process.

3.2.10. CAN HIRE AN AGENT

Table 3. 10: Can hire an agent

Use Case	Can hire an agent	
Goal <a longer statement of the goal in context if needed>	To hire an agent, he needs to send a hire requests to the agent.	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	Hiring requests sends successfully and appear successfully	
Failed End Condition <the state of the world if goal abandoned>	Hiring request could not sent successfully and could not appear successfully.	
Primary Actors:	Agent, Candidate	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Hiring requests request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	When the candidate makes a request for a package then the agent can see the requests. After that he can approve or reject the requests by seeing the candidate information
	2	After completion the process the candidate can see his hiring requests status
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	If the candidate could not able to make a hiring requests directly, he can choose another agent and make a hiring request to complete the process.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including candidate skill, education, experience, contacts etc.

3.2.11. CAN SEARCH JOB

Table 3. 11: Can search job

Use Case	Can search a job	
Goal <a longer statement of the goal in context if needed>	To search a job and find the desired job.	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	Candidate performs the search operation successfully and find the job successfully.	
Failed End Condition <the state of the world if goal abandoned>	Candidate could not perform search operation successfully and could not find the job successfully.	
Primary Actors:	Candidate	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Search request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	When the candidate attempts a search operation for getting a job then the candidate can see the diverse types of job.
	2	After completion the process the candidate can apply for a job.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	If the candidate could not able to make a search operation directly, he can try out with the job type and job keyword instead of job location.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including job type, job keyword, job location etc.

3.2.12. CAN APPLY FOR A JOB

Table 3. 12: Can apply for a job

Use Case	Can apply for a job	
Goal <a longer statement of the goal in context if needed>	Agents can apply for a job if he knows about the process properly.	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	Job application sent successfully and approved successfully.	
Failed End Condition <the state of the world if goal abandoned>	Job application not sent successfully and not approved successfully	
Primary Actors:	Candidate, Employer	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Job application request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	Candidate gives the required information to make the application. After he hits enter then the application has been sent successfully, after that he can see the requests status.
	2	After approved the job he gets the job approval letter.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	Sometimes they can be fall into the trouble if they could not give their accurate and required information. So, he needs to provide accurate and required information to complete this process successfully.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including preferable salary, skills, name, contact etc. for completion of the whole process.

3.2.13. CAN MAKE A PAYMENT

Table 3. 13: Can make a payment

Use Case	Can make a payment	
Goal <a longer statement of the goal in context if needed>	Candidate can make a payment and get the service properly.	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	Payment is processed successfully and got the service successfully.	
Failed End Condition <the state of the world if goal abandoned>	Payment is not processed successfully and not got the service successfully.	
Primary Actors:	Candidate, Agent	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Payment request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	After got the approval confirmation from the side of the agent then the candidate is able to make a payment.
	2	After complete the payment process, he can got the service properly.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	Sometimes he is not able to make the payment directly. Before making the payment request, please check that whether you make the hiring requests to the agent or not.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including preferable amount, package, name, contact etc. for completion of the whole process.

3.2.14. CAN GET APPROVED VISA

Table 3. 14: Can get approved visa

Use Case	Can get an approved visa	
Goal <a longer statement of the goal in context if needed>	To get the approved visa after completion of the process from the side of the agent	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	Get the visa successfully.	
Failed End Condition <the state of the world if goal abandoned>	Could not get the visa successfully.	
Primary Actors:	Agent, Candidate	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Visa in detail comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	When every process has been successfully done by the agent then agent is going to give the visa in details to the candidate
	2	After getting the visa he can be marked as successful.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	If he does not find the visa then he can make enquiry to our system admin. Then the admin will go for the justification process.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including flight schedule, company info etc.

3.2.15. CAN APPROVE JOB POST

Table 3. 15: Can approve job post

Use Case	Can approve job post	
Goal <a longer statement of the goal in context if needed>	To approve a job post after comes the requests from the side of an employer	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	Approve requests comes successfully and approved successfully	
Failed End Condition <the state of the world if goal abandoned>	Approve request could not come successfully and could not approved successfully	
Primary Actors:	Employer, Super Admin	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Approve job request comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	When the employer makes a job post then the super admin can see the job post. After that he can approve or reject the job post by seeing the job post's information
	2	After completion the process the employer can see his made job post status
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	If the employer could not able to make a job posts directly then preferable suggestions is to make sure that your information is correct.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including company info, name, vacancy, salary etc.

3.2.16. CAN VERIFY AGENT

Table 3. 16: Can verify agent

Use Case	Can verify agent	
Goal <a longer statement of the goal in context if needed>	To verify an agent	
Preconditions <what we expect is already the state of the worlds>	N/A	
Success End Condition <the state of the world upon successfully complete>	Agent verification has been completed successfully.	
Failed End Condition <the state of the world if goal abandoned>	Agent verification has not been completed successfully.	
Primary Actors:	Agent, Super Admin	
Secondary Actors	N/A	
Trigger <the action upon the system that starts use case>	Agent verification process comes in	
Description/main Success Scenario <the steps of the scenario from the trigger to goal delivery and any clean up after>	**Step**	**Action**
	1	When an agent is going to create a registration then a request has been sent to the super admin. After that the admin will verify the agent whether the provided information is correct or not.
	2	If all of the information is correct which comes from the side of an agent then the agent can complete his registration. Otherwise, he cannot complete his registration.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case >	**Step**	**Branching Action**
	1a	If the agent is not able to complete his registration process successfully then the preferable suggestion is to make sure that you provide the correct information of yours.
Quality Requirements	**Step**	**Requirement**
	1	Requirements are including company info, license etc.

3.3. ACTIVITY DIAGRAM

3.3.1. LOGIN

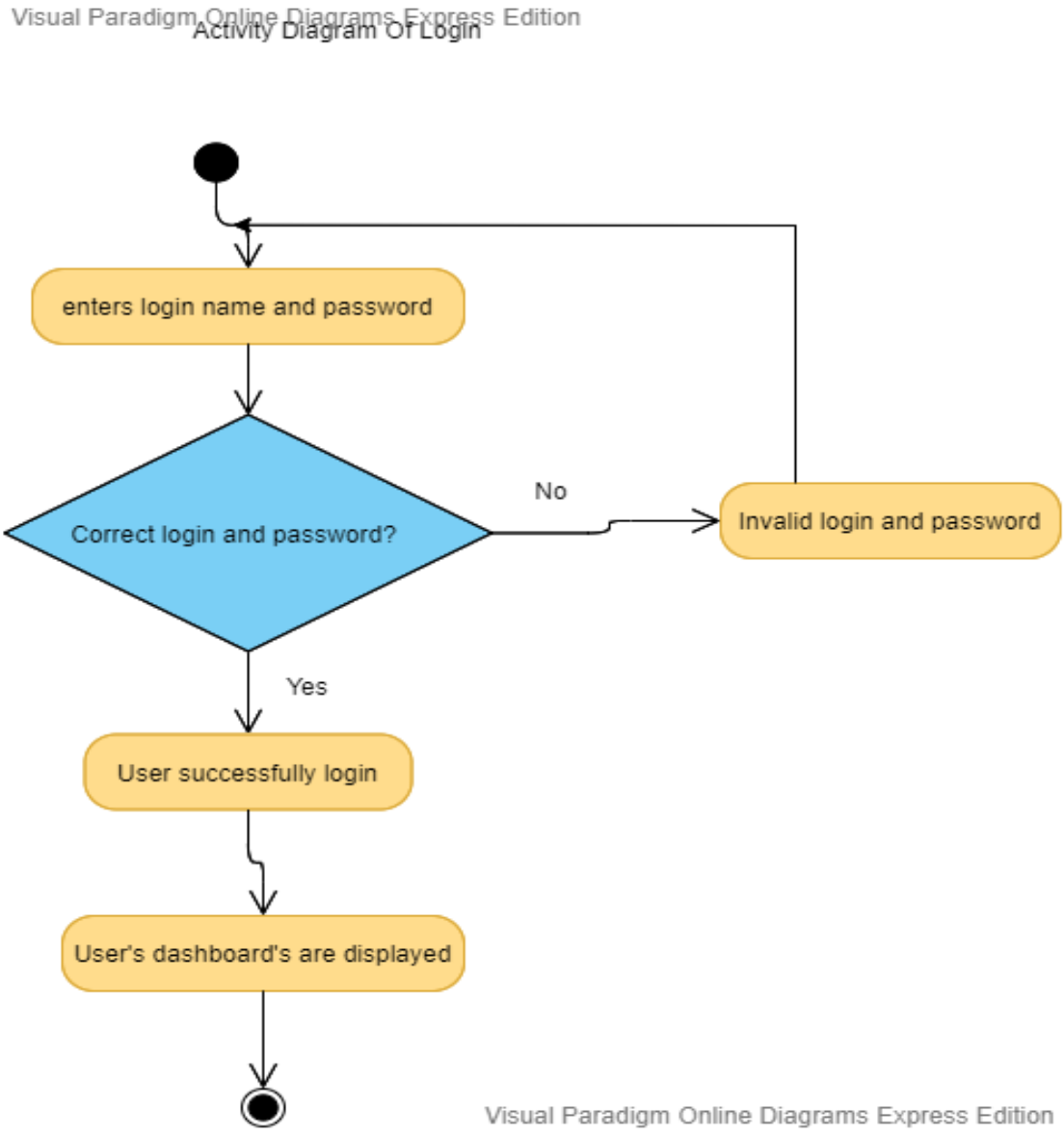


Figure 3. 2: Overview of user login

3.3.2. REGISTRATION

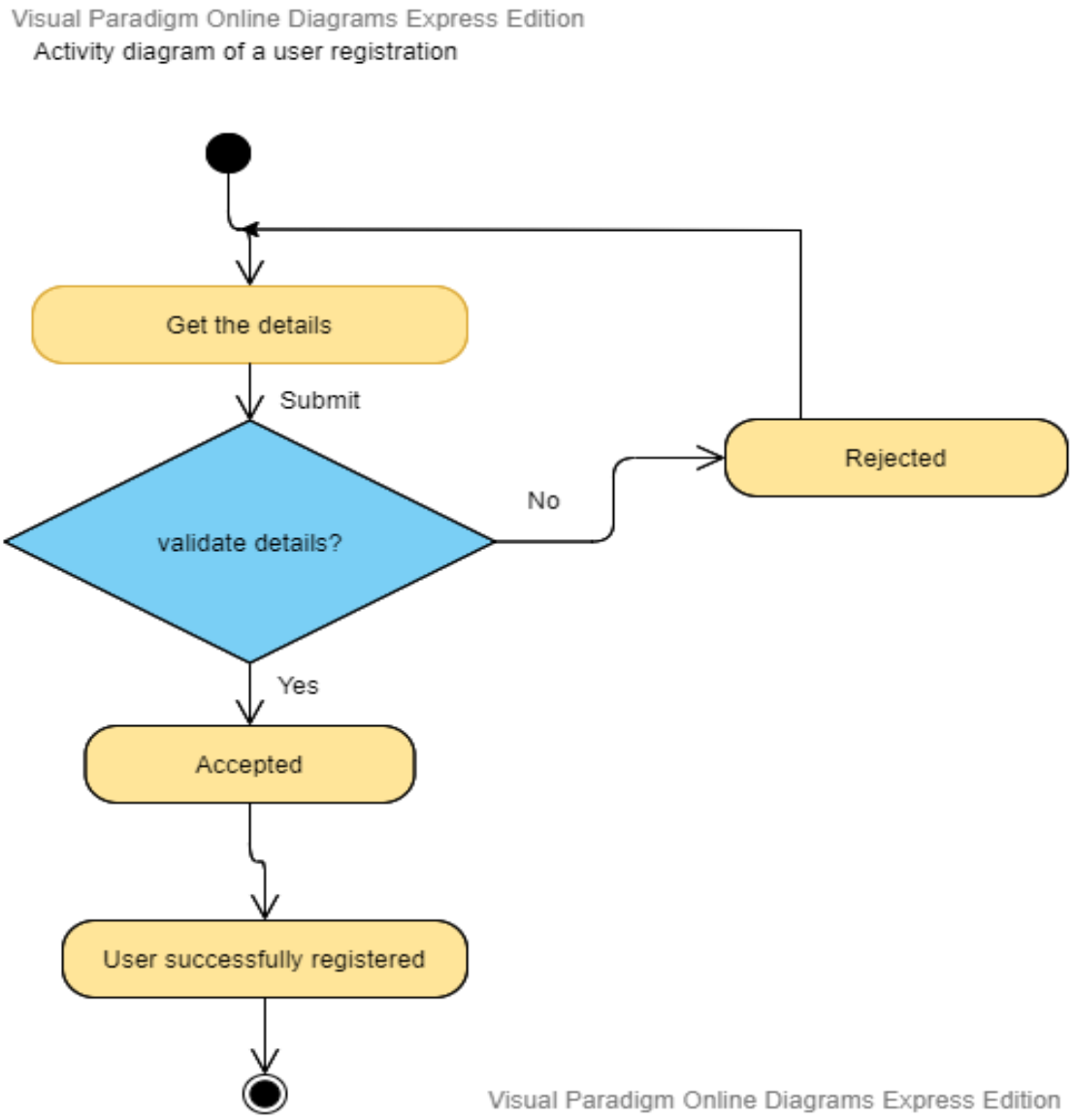


Figure 3. 3: Overview of user registration

3.3.3. MANAGE PROFILE

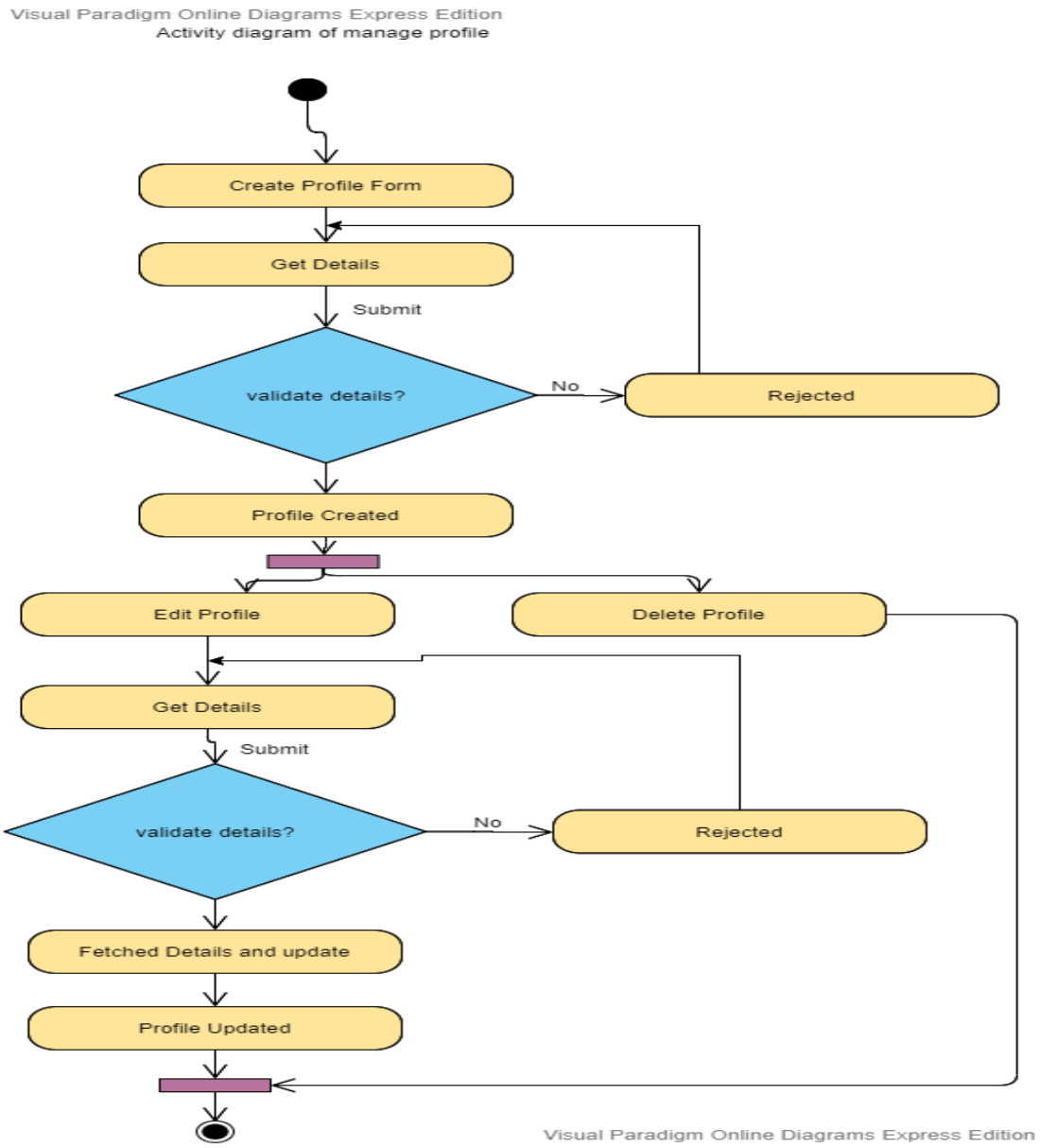


Figure 3. 4: Overview of manage profile

3.3.4. LIVE CHAT

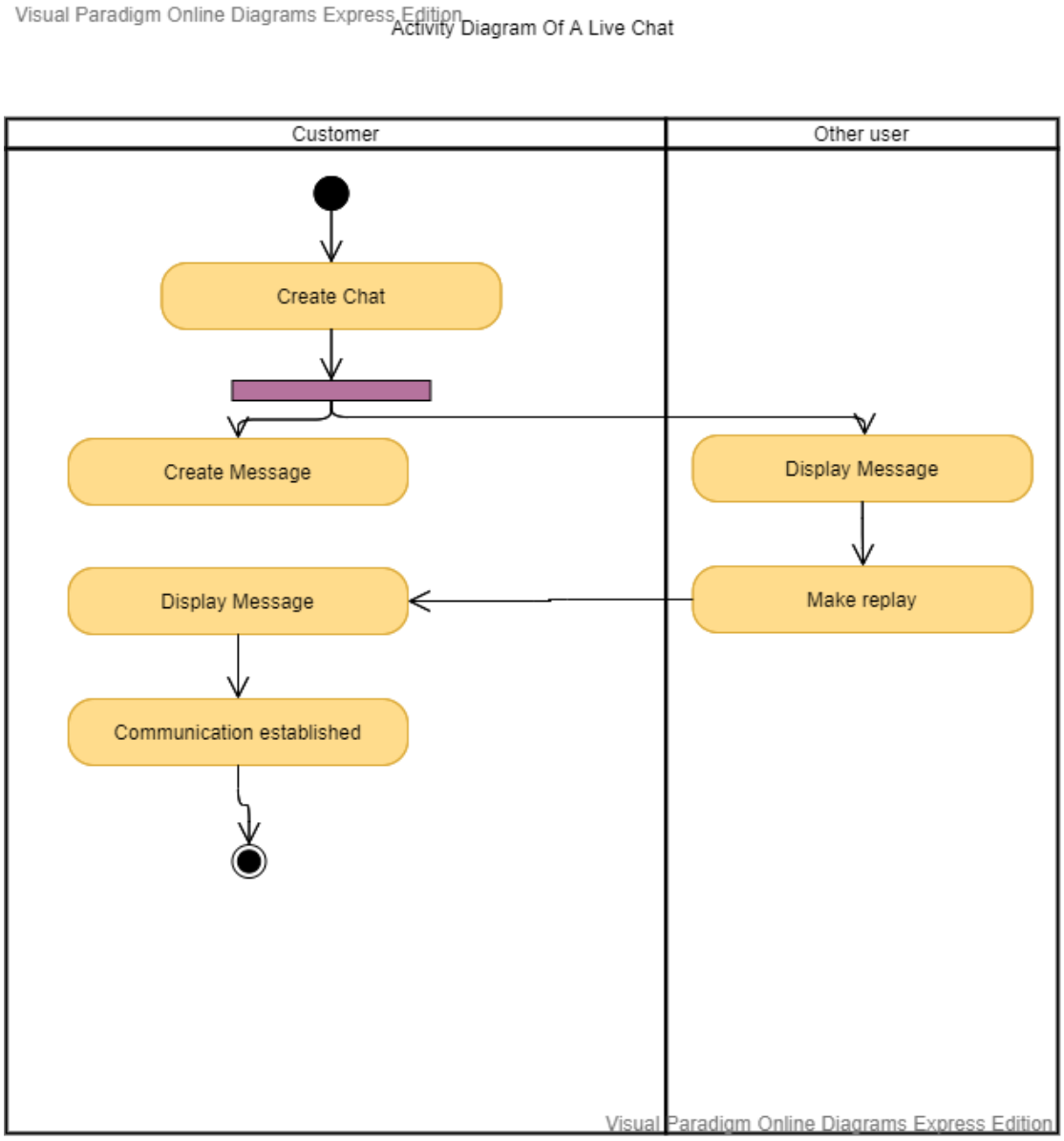


Figure 3. 5: Overview of live chat

3.3.5. CAN SEE CANDIDATES

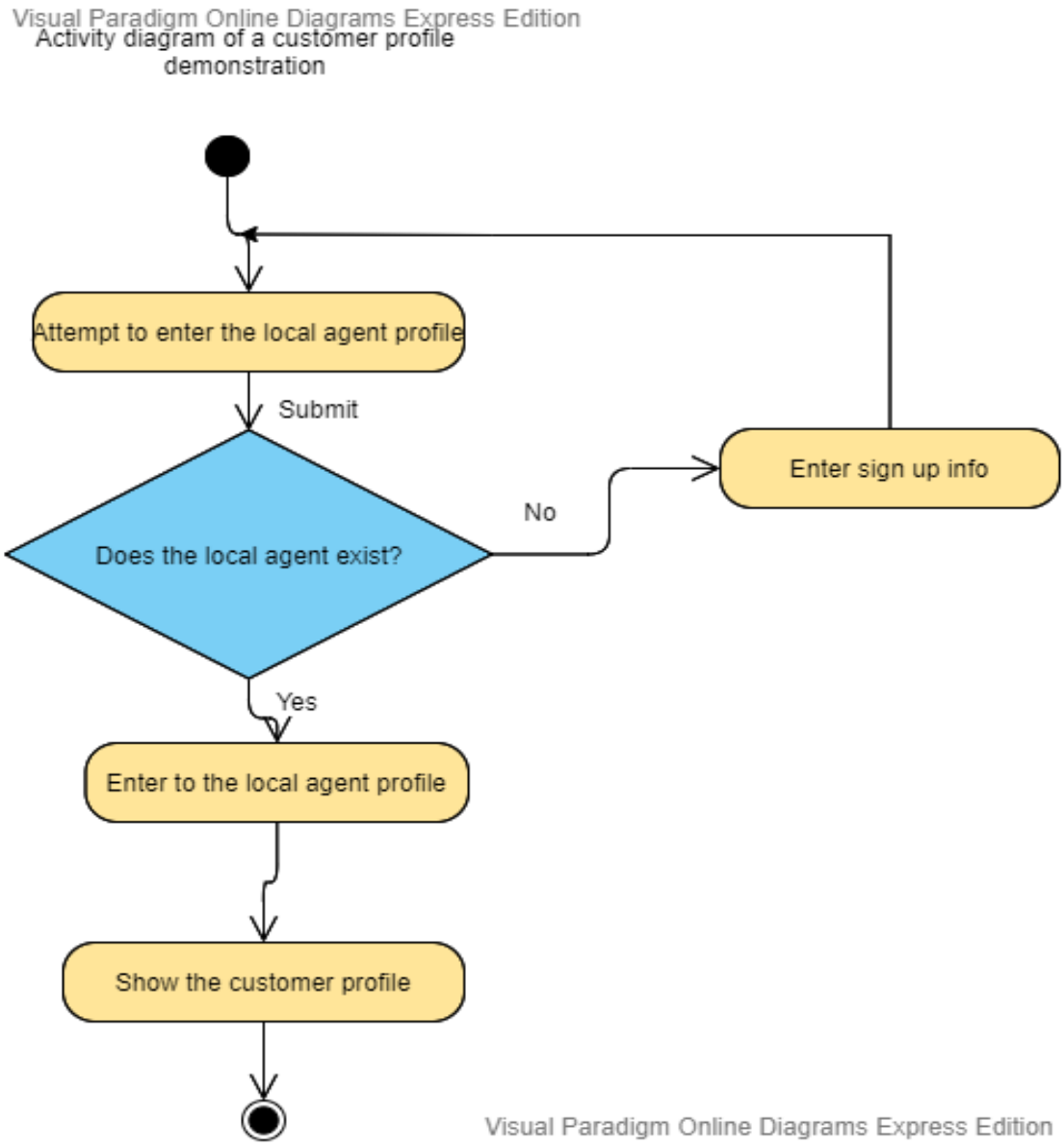


Figure 3. 6: Overview of see candidates

3.3.6. JOB DETAILS

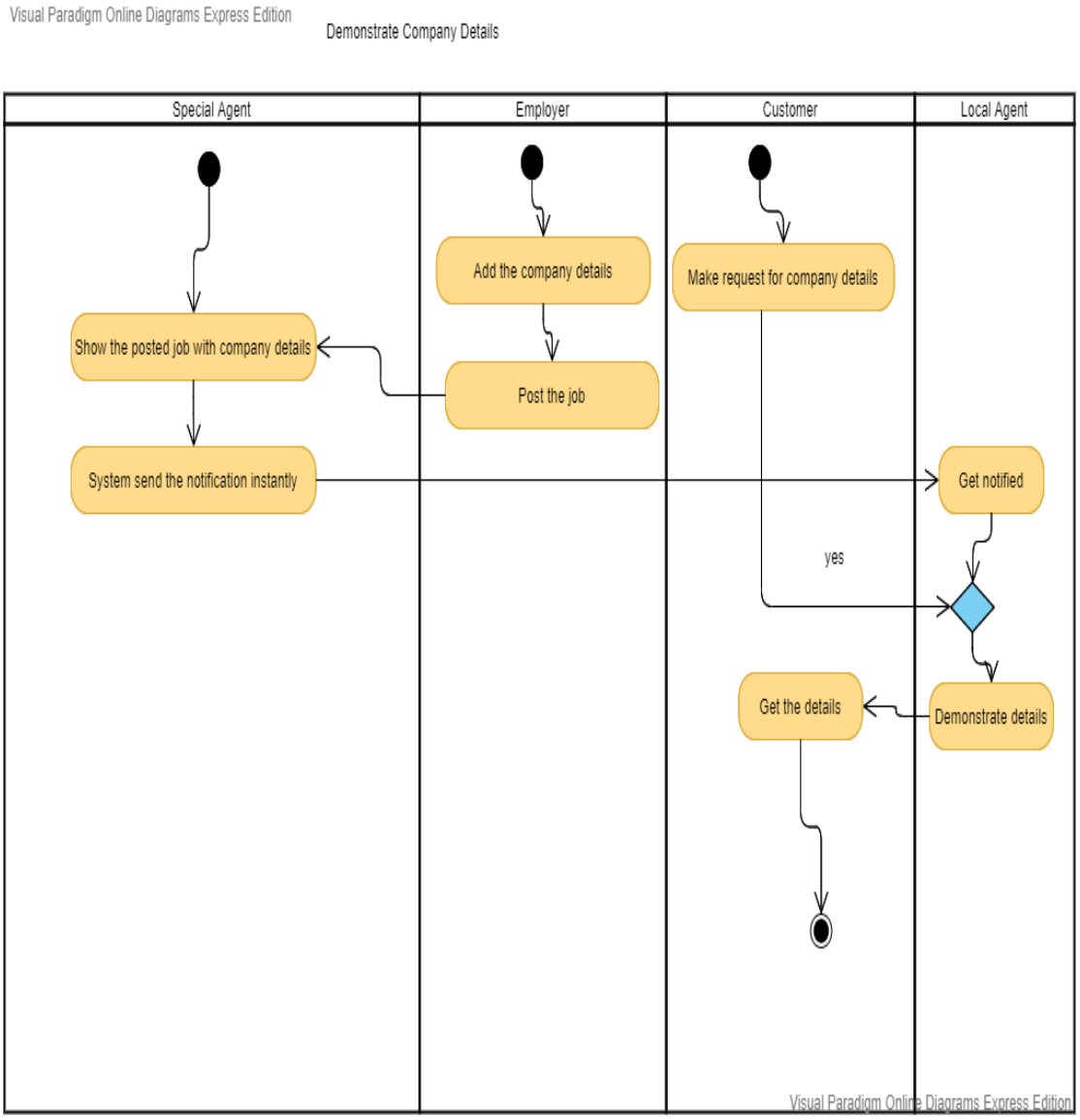


Figure 3. 7: Overview of job details

3.3.7. MANAGE JOB

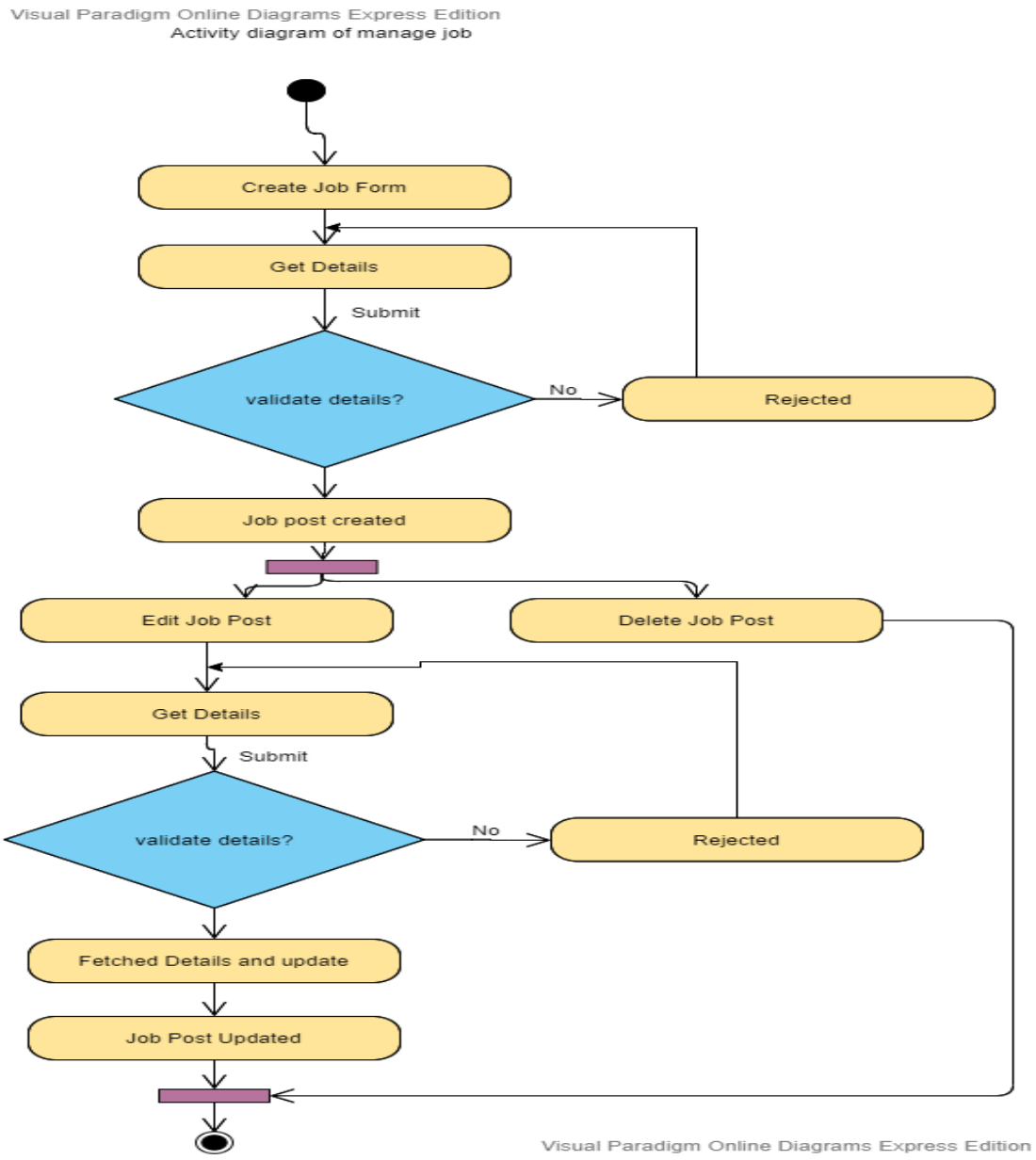


Figure 3. 8: Overview of manage job

3.3.8. SEARCH JOB

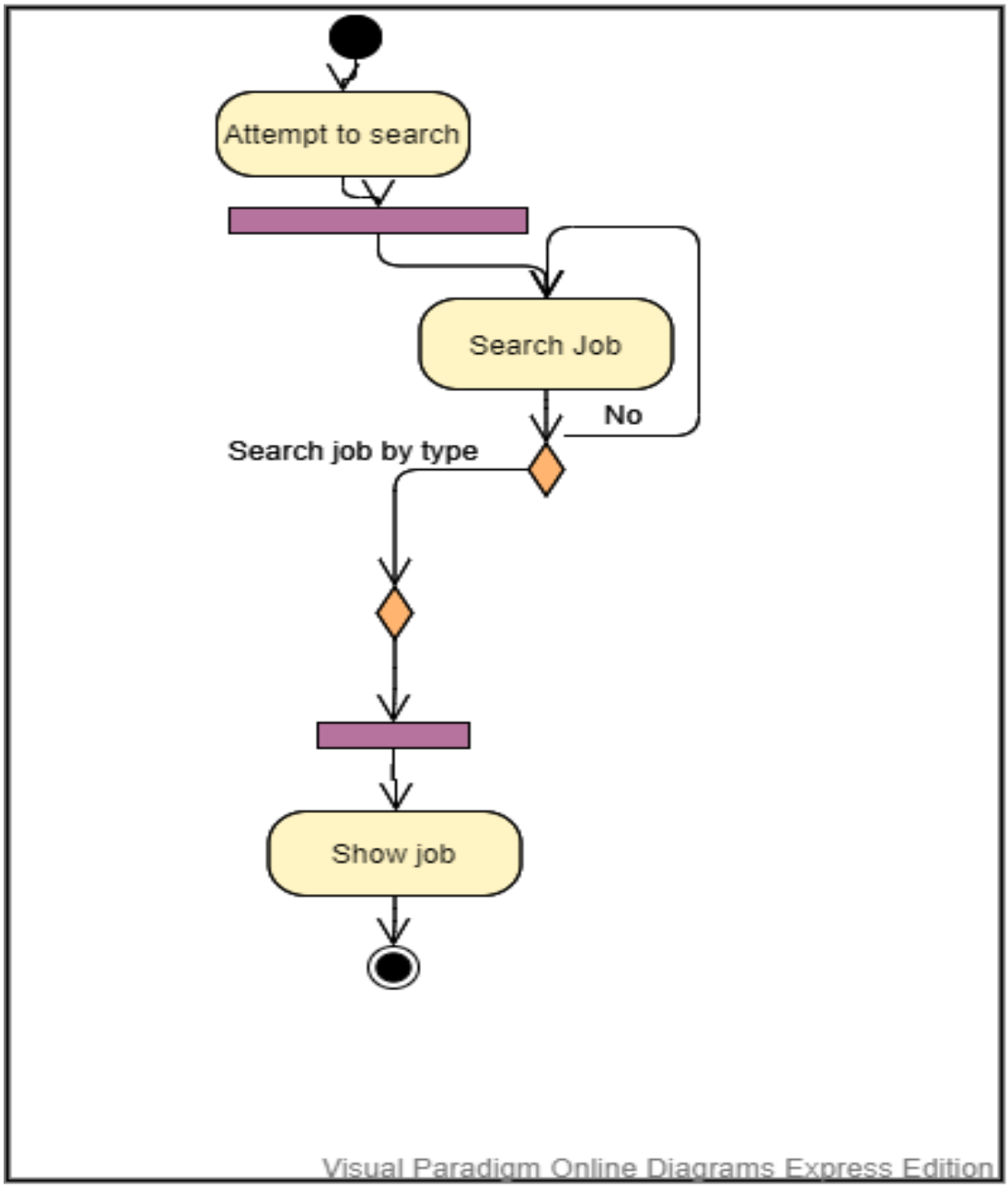


Figure 3. 9: Overview of search job

3.3.9. GET APPROVED VISA

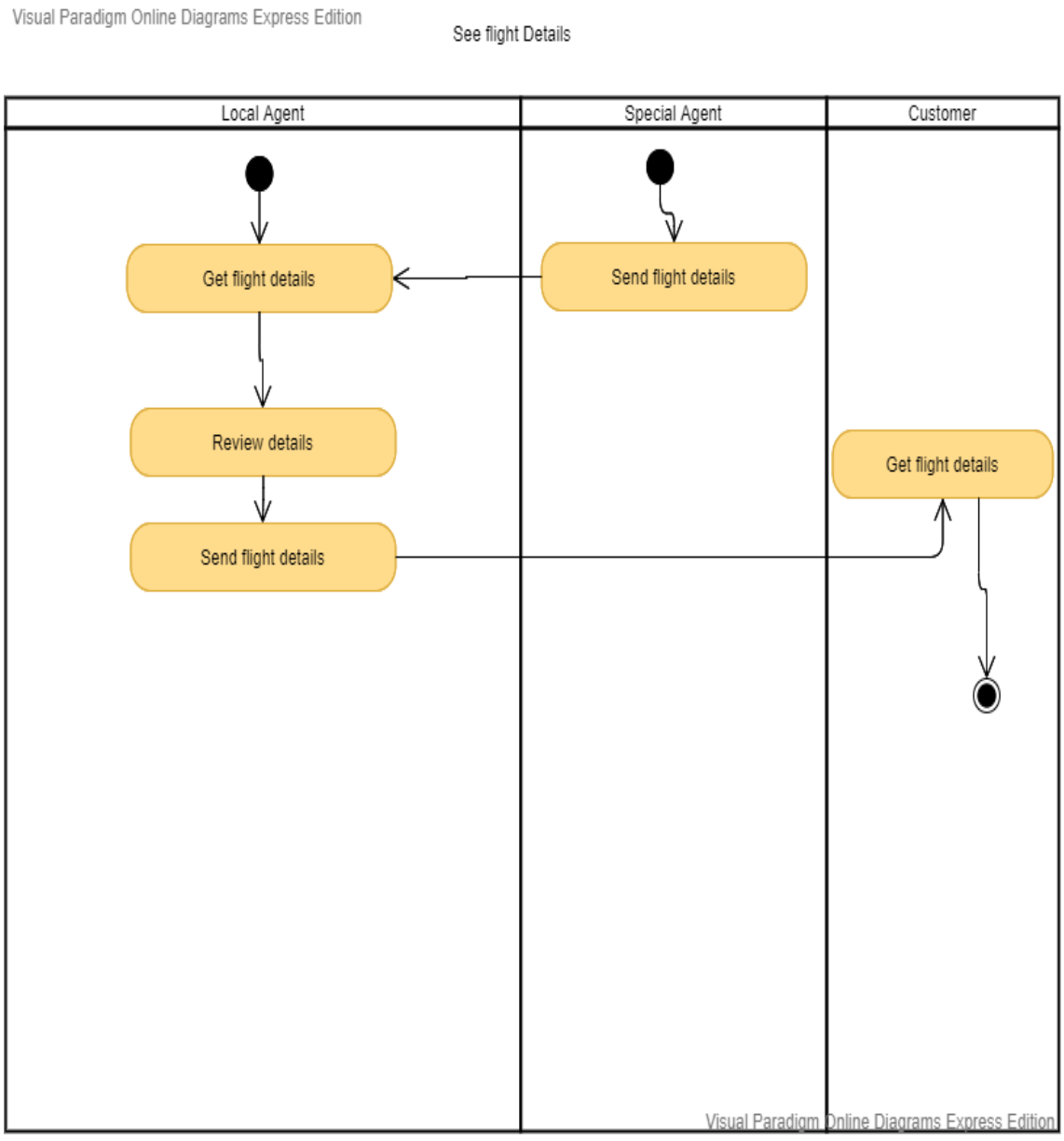


Figure 3. 10: Overview of get approved visa

3.3.10. SENT HIRE REQUESTS

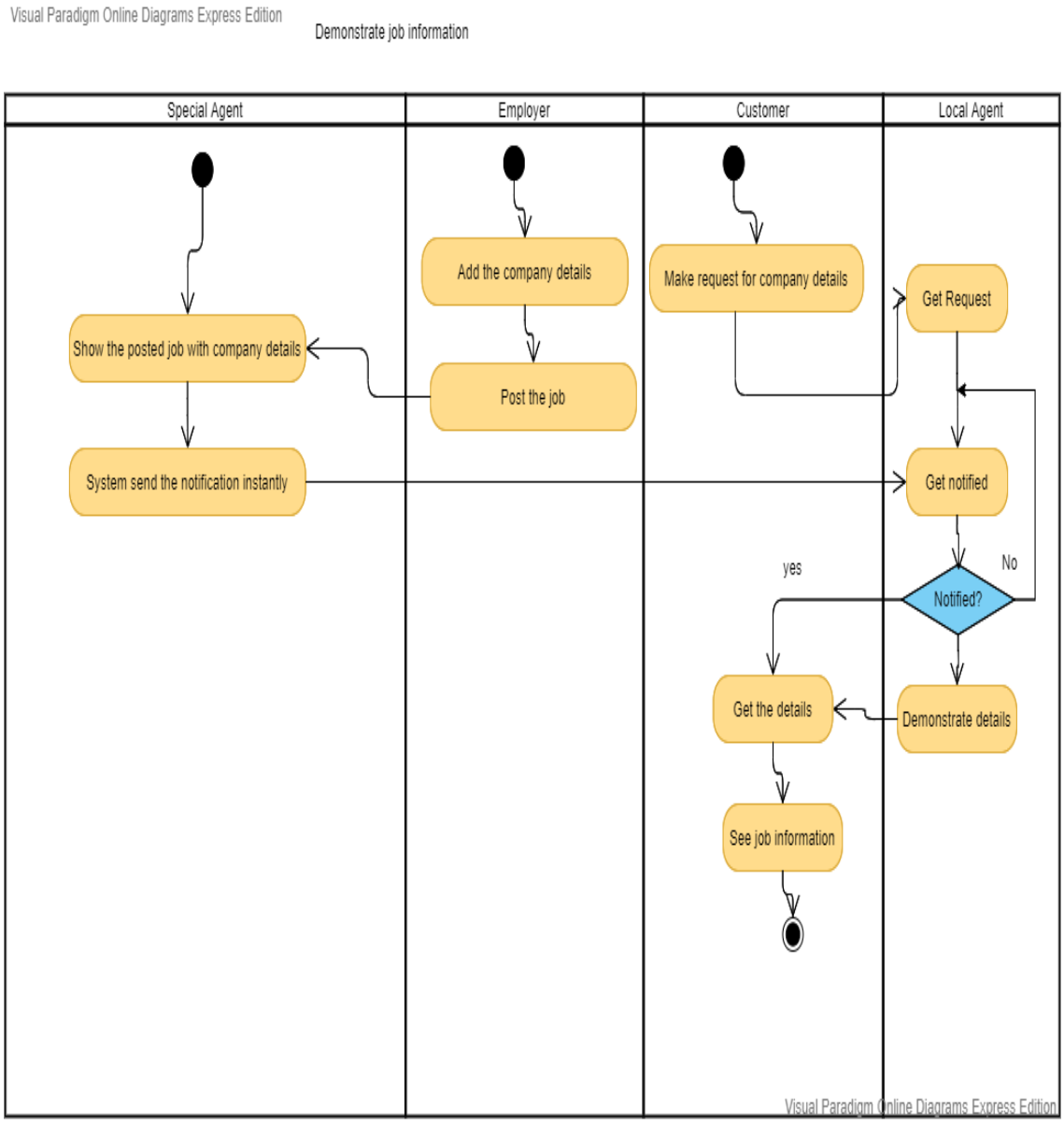


Figure 3. 11: Overview of sent hiring requests

3.4. SEQUENCE DIAGRAM

3.4.1. LOGIN

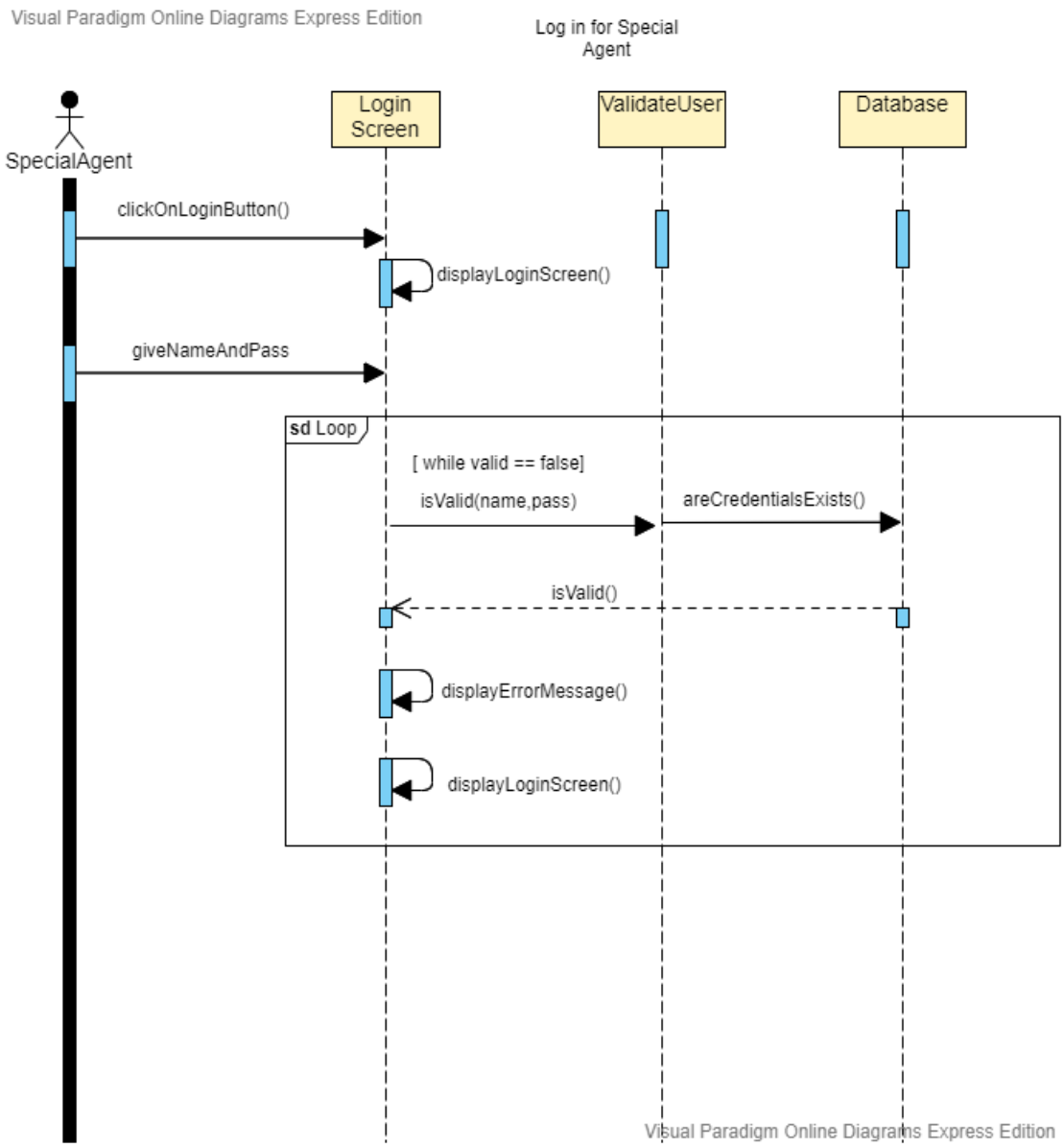


Figure 3. 12: Overview of user login

3.4.2. REGISTRATION

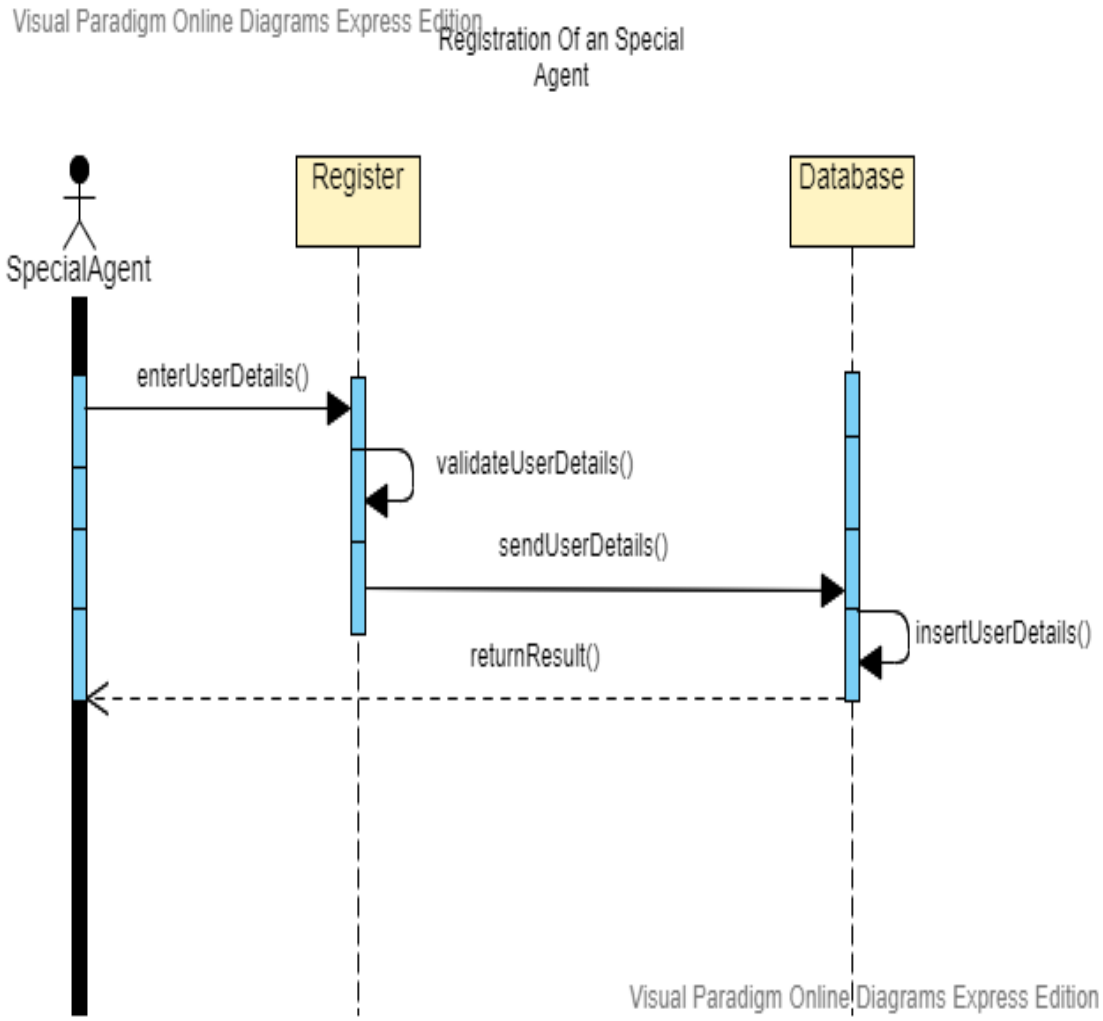


Figure 3. 13: Overview of user registration

3.4.3. MANAGE JOB

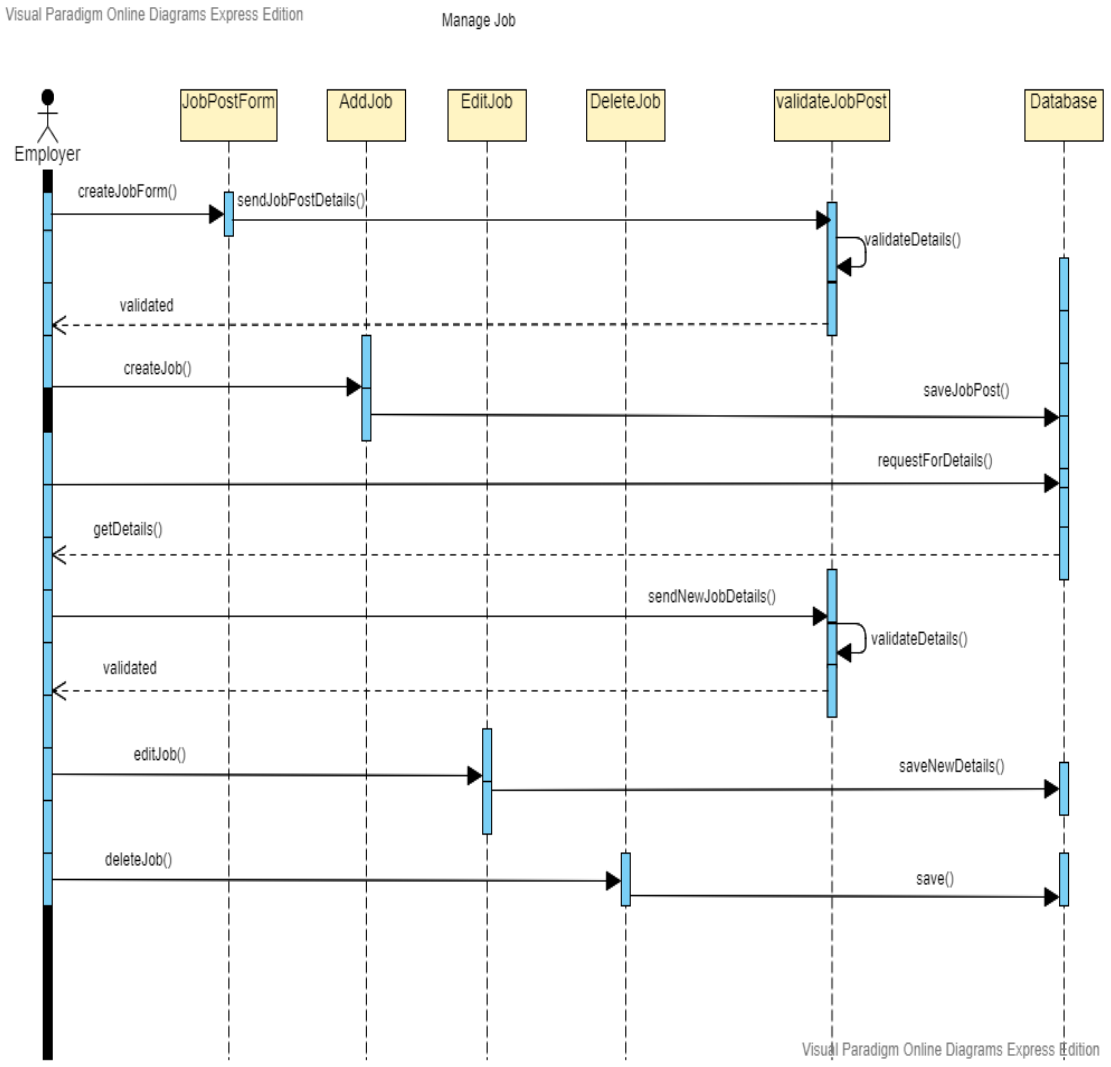


Figure 3. 14: Overview of manage job

3.4.4. LIVE CHAT

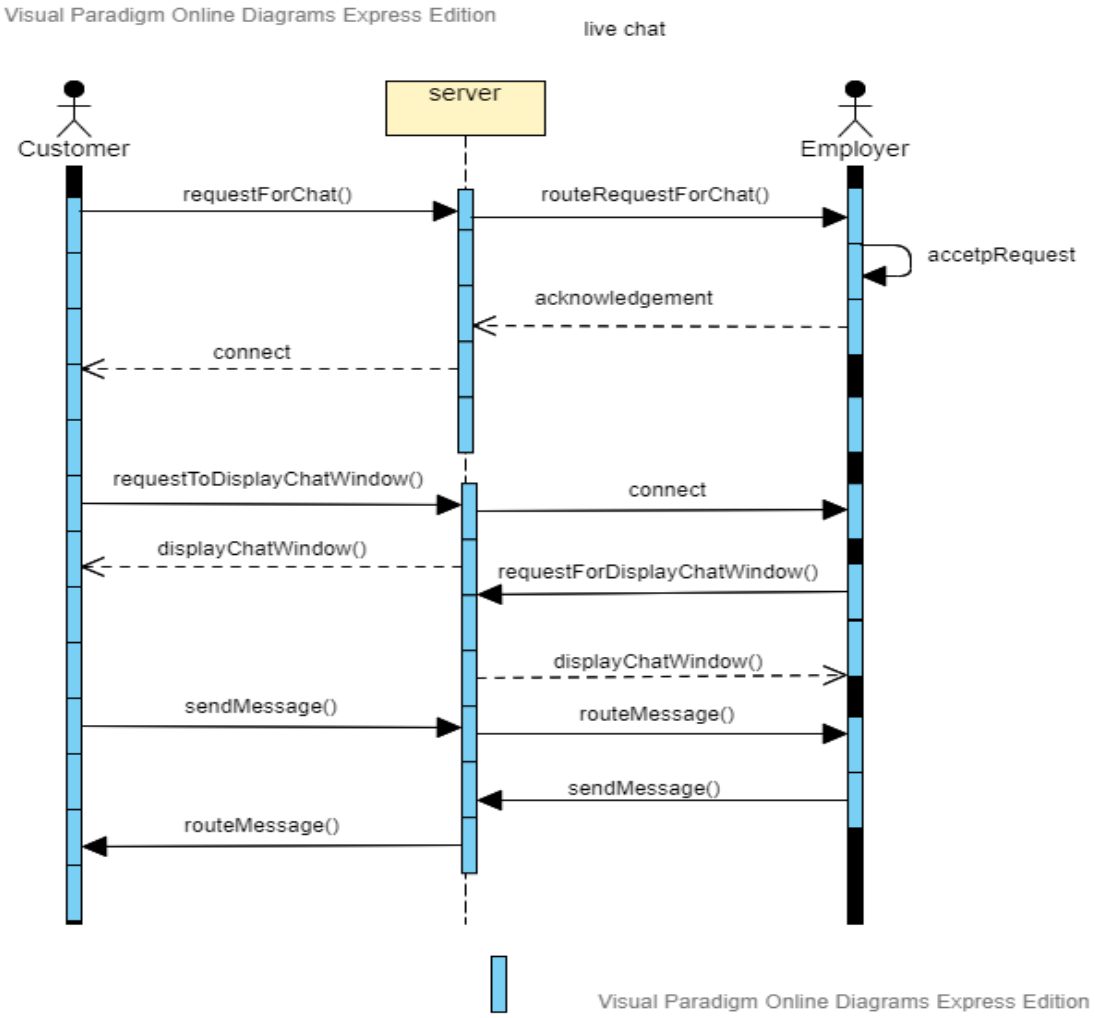


Figure 3. 15: Overview of live chat

3.4.5. MANAGE PROFILE

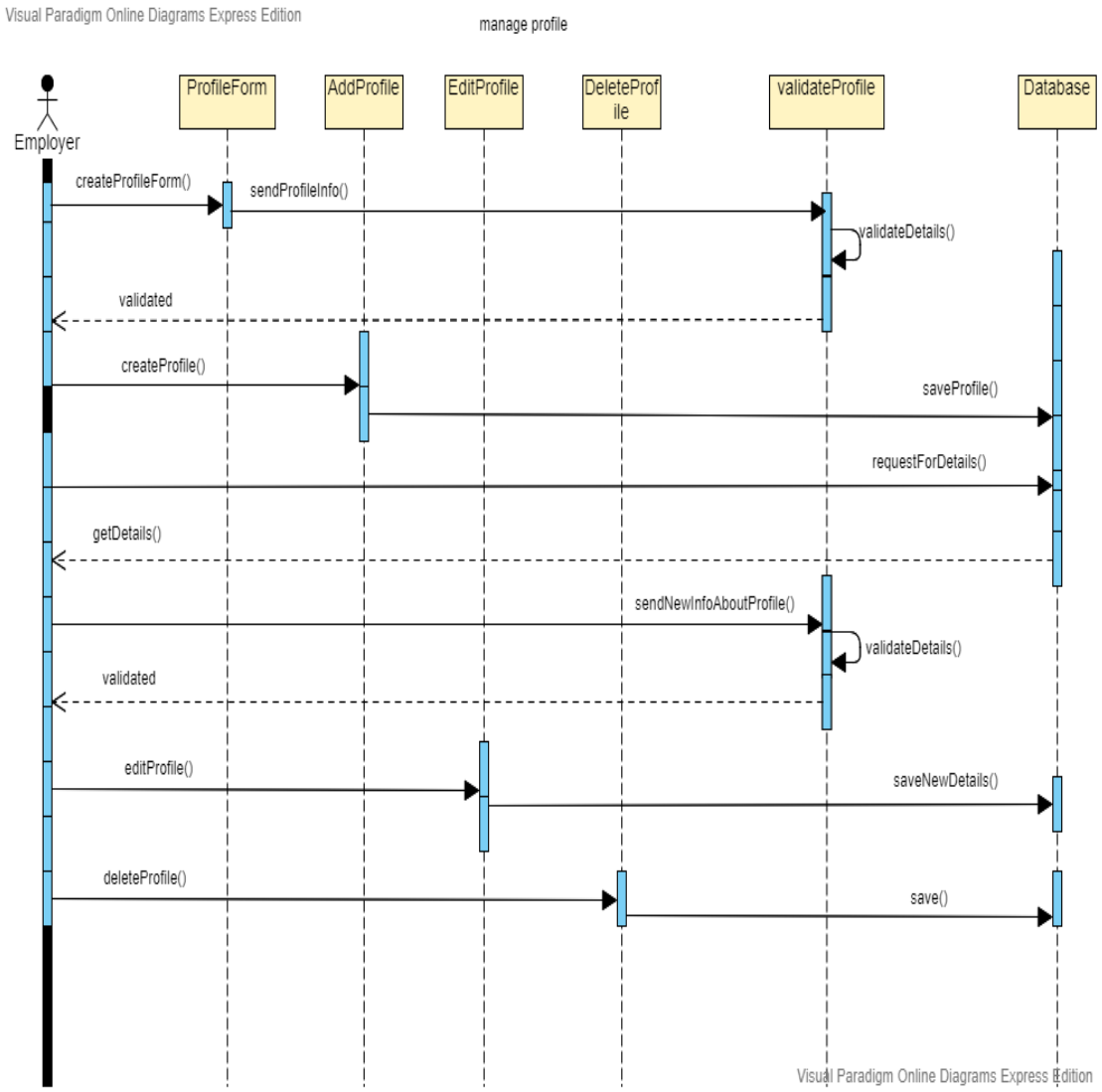


Figure 3. 16: Overview of manage profile

3.4.6. SEARCH JOB

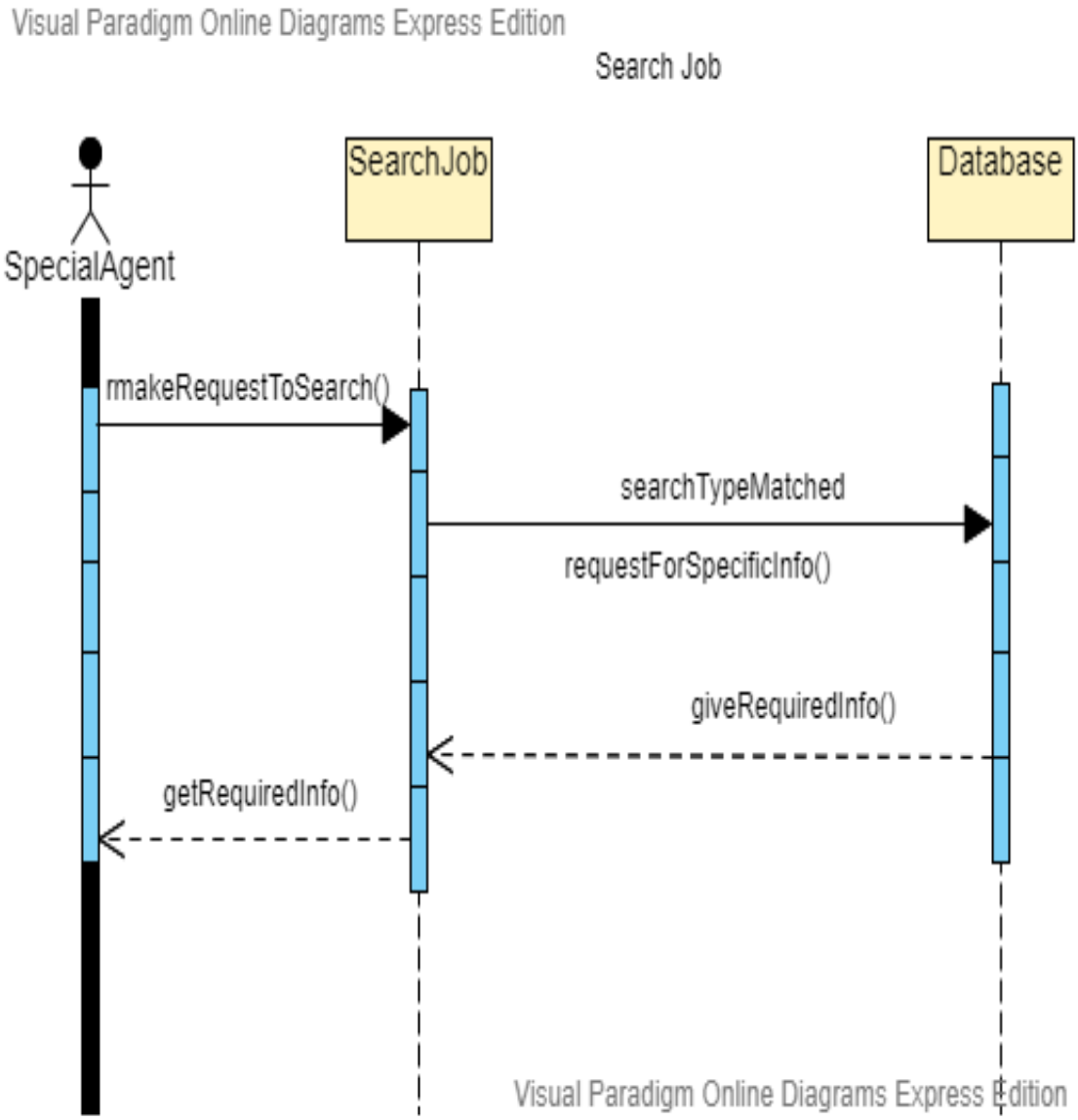


Figure 3. 17: Overview of manage job

3.4.7. CAN SEE CANDIDATES

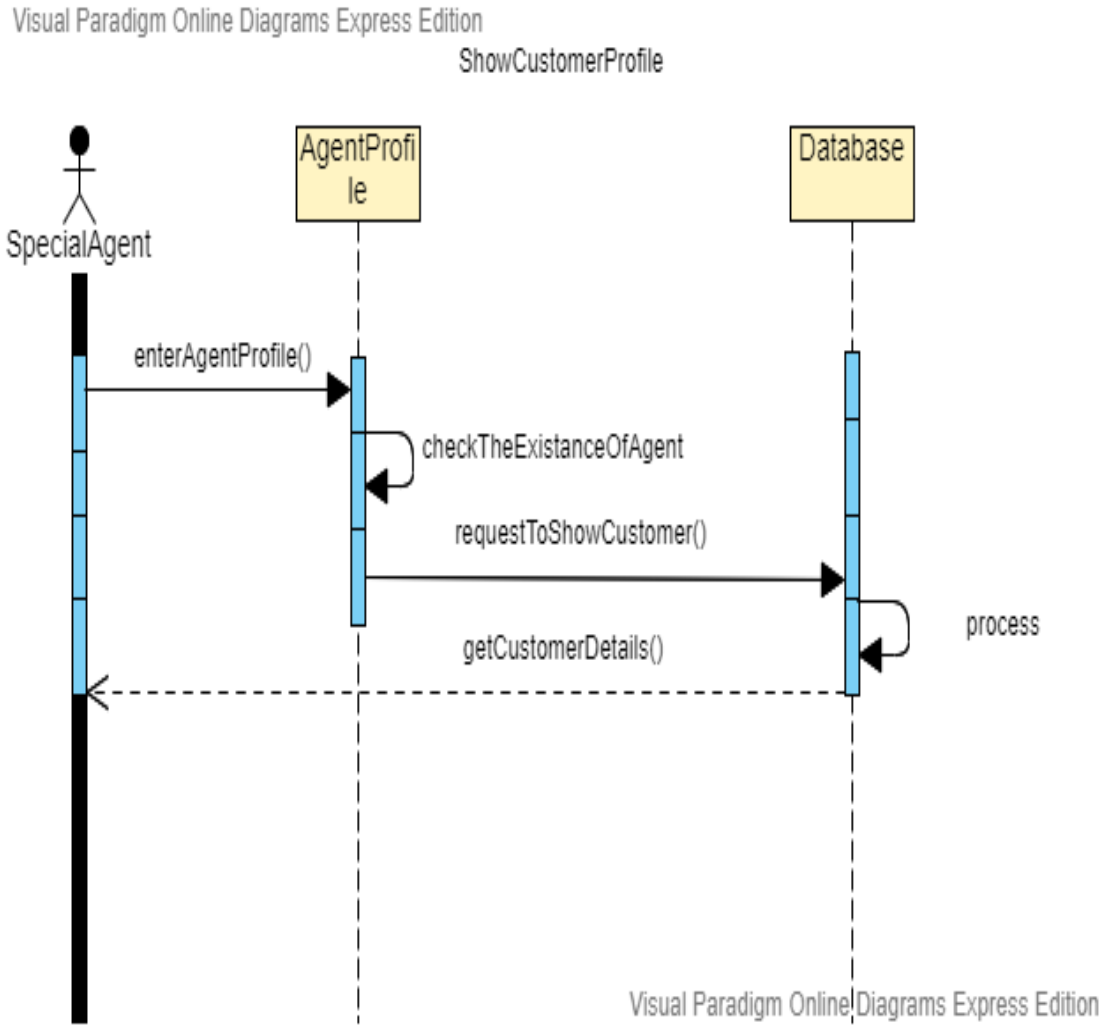


Figure 3. 18: Overview of see candidates

3.4.8. GET APPROVED VISA

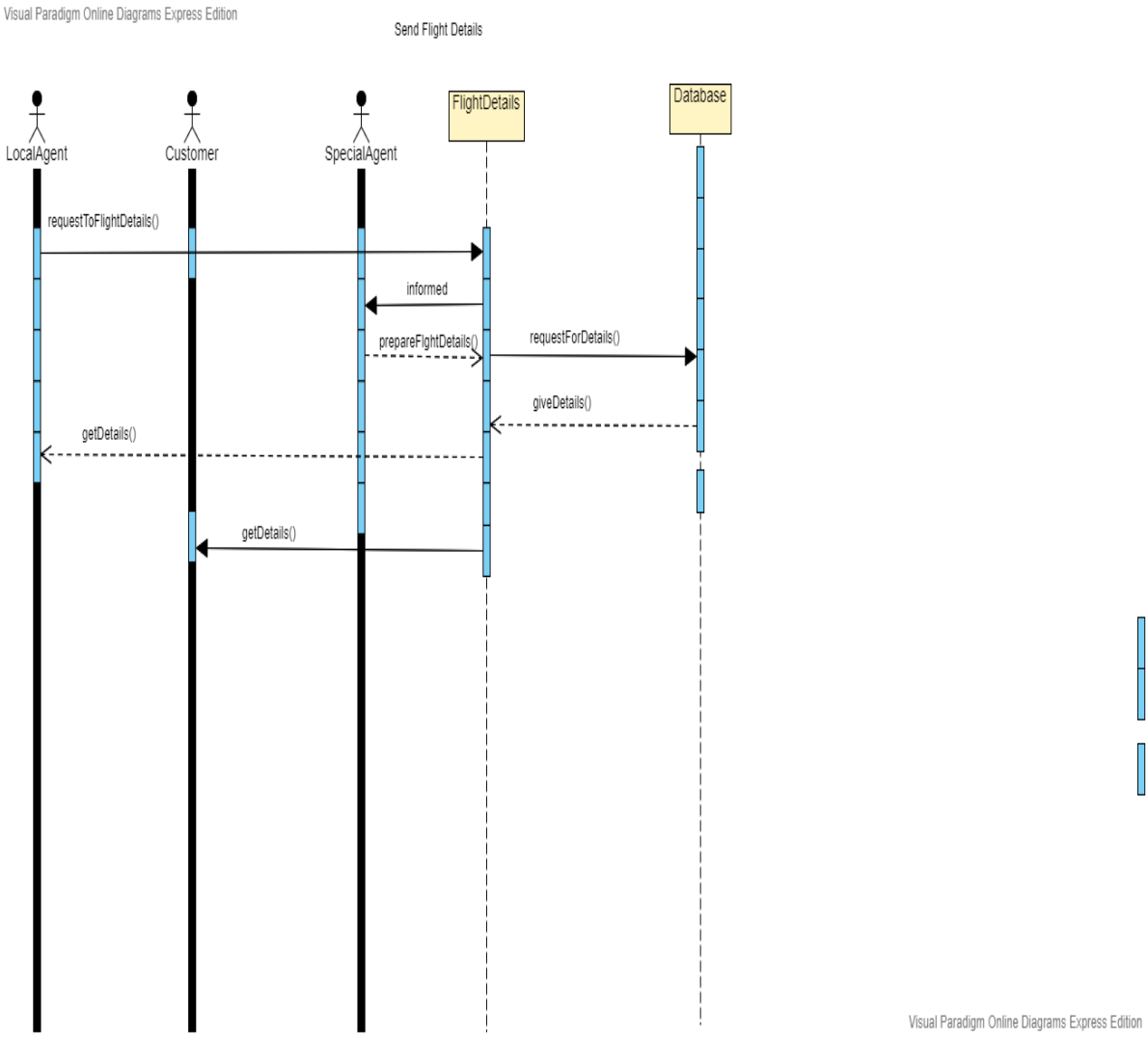


Figure 3. 19: Overview of getting approved visa

3.4.9. JOB DETAILS

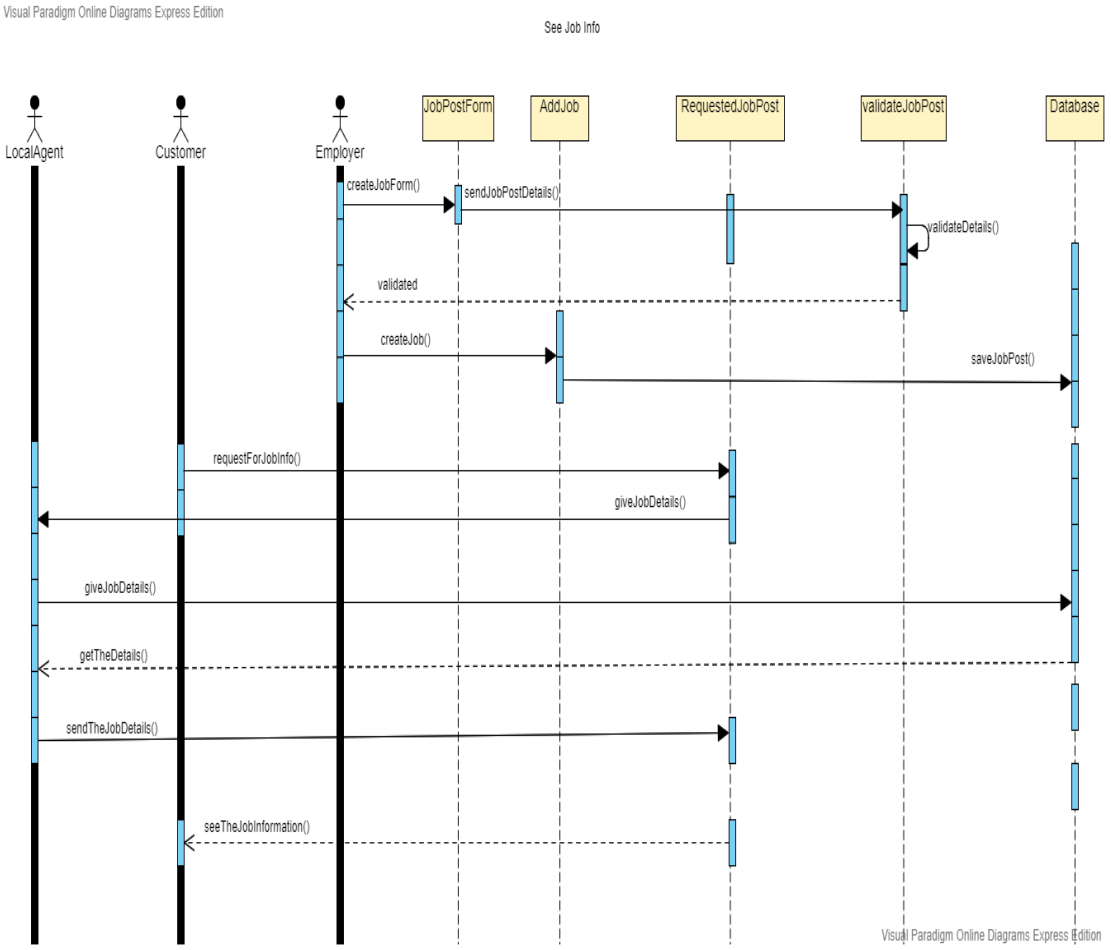


Figure 3. 20: Overview of job details

3.4.10. SENT HIRE REQUESTS

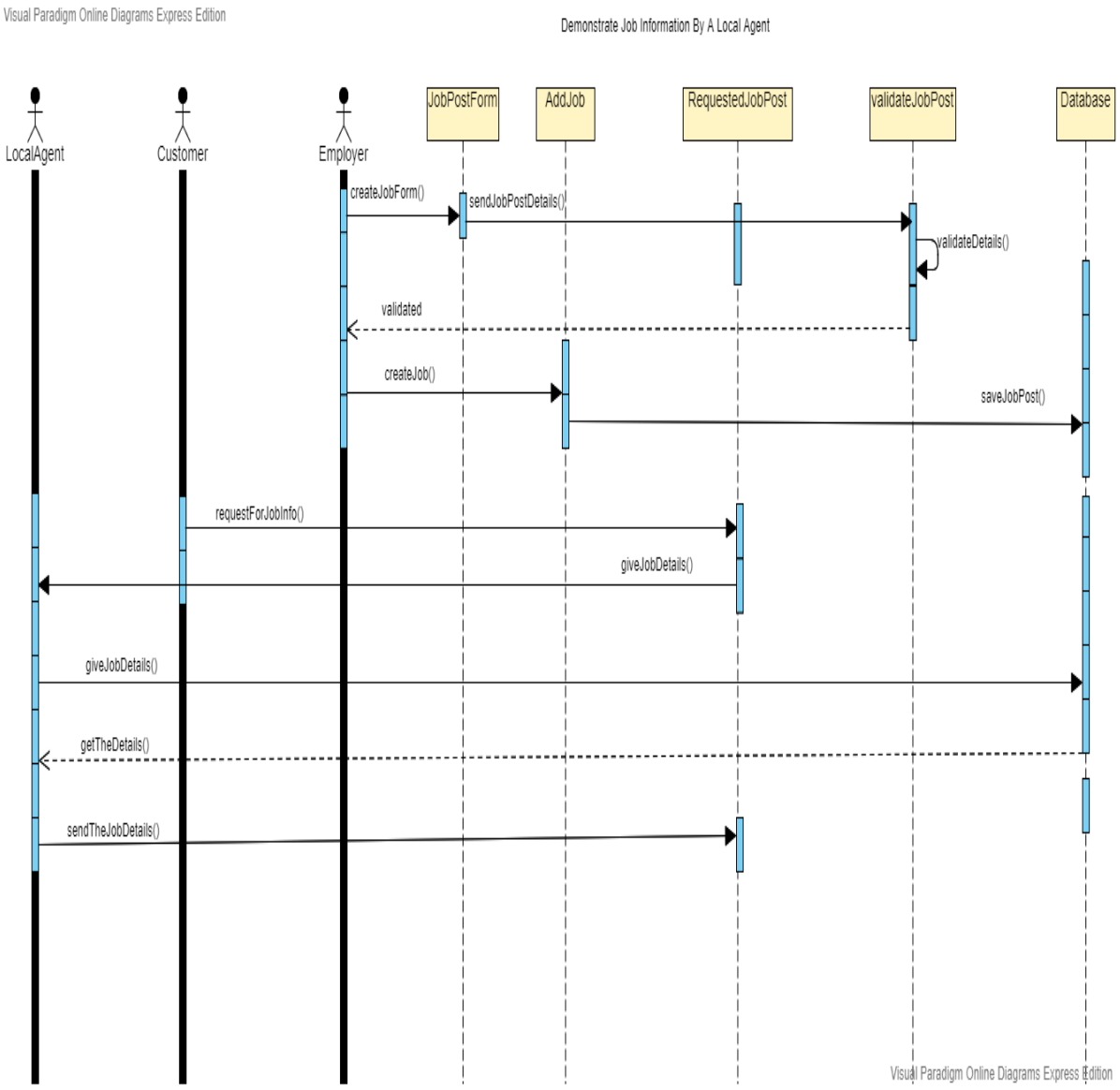


Figure 3. 21: Overview of sent hiring requests

4. CHAPTER-04 SYSTEM DESIGN SPECIFICATION

4.1. ENTITY RELATIONSHIP DIAGRAM

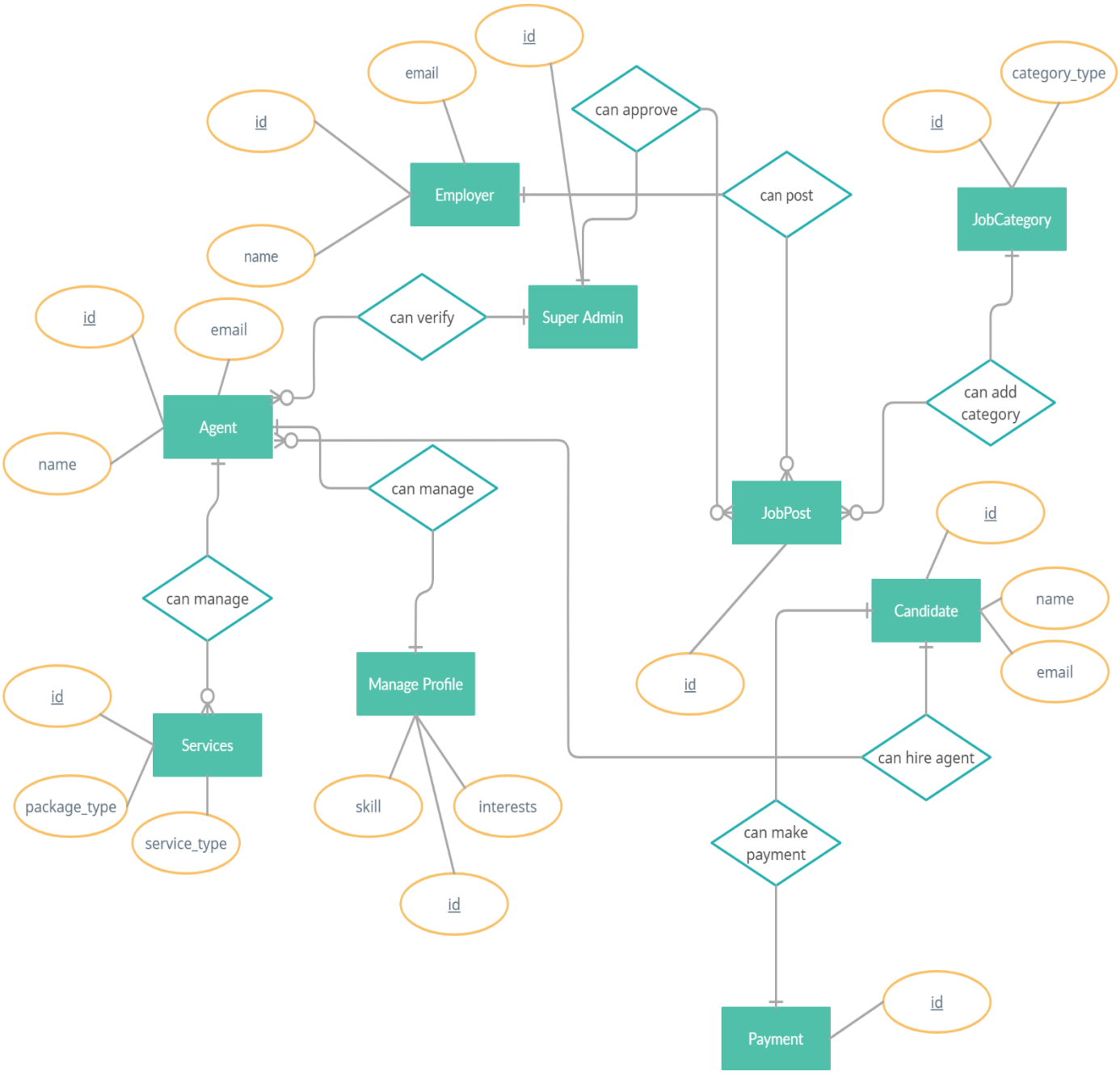


Figure 4. 1: Overview of entity relationship diagram

4.2. CLASS DIAGRAM

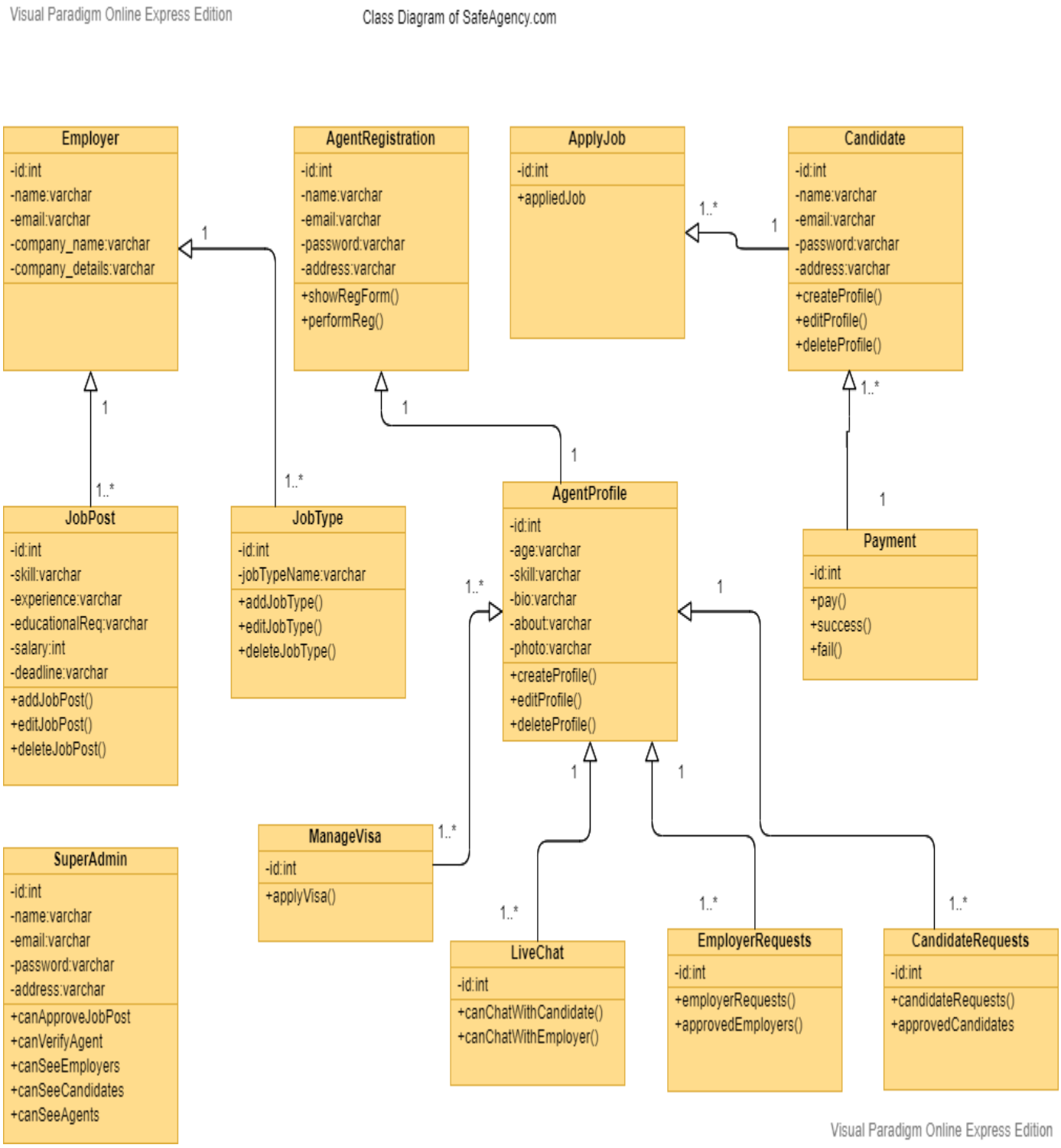


Figure 4. 2: Overview of class diagram

4.3. Development Tools and Technology

4.3.1. User Interface Technology

- HTML5
- CSS3
- JAVASCRIPT
- VUE JS

4.3.2. Implementation Tools and Platforms

- PHP7
- LARAVEL6
- XAMPP
- PHP STORM

5. CHAPTER-05 SYSTEM TESTING

5.1. Testing Features

5.1.1. Feature to be tested

- Login
- Registration
- Manage profile
- Manage job
- Manage service
- Verify agent
- Payment

5.1.2. Feature not to be tested

- Subscription

5.2. Testing Strategies

5.2.1. Test Approach

- Testing approaches are done by the side of author of this system.
- Testing approaches are going to apply in the above-mentioned features.
- To complete the test approach, we required to meet some criteria specifically.

5.2.2. Pass or Fail Criteria

- If the test condition matches then we are going to marked the status as Pass.
- If the test condition does not match then we are going marked the status as Fail.

5.2.3. Suspension and Resumption

Suspension criteria says that the criteria to be used to suspend all or a specific portion of the testing activities whereas resumption criteria says when testing can pause after it has been suspended.

- It specifies the unavailability of external dependent systems during the time of execution.
- When a defect is caught that cannot allow any further testing.
- Critical path schedule or deadline is missed so that the client will not accept delivery even if all testing is completed.
- It tells that a specific holiday shuts down both development and testing.

5.2.4. Testing Schedule

Here I break out the whole project into small tasks and add the schedule for each task as below

Table 5. 1: Testing schedule

Test Phase	**Time**
To create test plan	2 Week
To create the test Specification	1 Week
Test Specification Team	1 Week
Component Test	2 Week
Integration Test	1 Week
System Testing	4 Week

5.3. TEST CASE

5.3.1. TEST CASE 1

Table 5. 2: Test Case1 (Login)

TEST CASE ID	TEST CASE NAME	TEST SCENARIO	TEST STEPS	TEST DATA	EXPECTED RESULTS	PASS/FAIL
TU01	Login	Check user login with valid data	1) Go to site 2) Enter email 3) Then enter password 4) After that click submit	Email=’princeashifulislam@gmail.com’ Password=12345678	The user should log into the system successfully	Pass
TU02	Login	Check user login with invalid data	1) Go to site 2) Enter email 3) Then enter password 4) After that click submit	Email=’princeashifulislam@gmail.com’ Password=1234567	The user should not log into the system successfully	Pass

5.3.2. TEST CASE 2

Table 5. 3: Test Case2 (Registration)

TEST CASE ID	TEST CASE NAME	TEST SCENARIO	TEST STEPS	TEST DATA	EXPECTED RESULTS	PASS/FAIL
TU01	Registration	Check user registration with valid data	1) Go to site 2) Enter name 3) Then enter email 4) Enter contact 4) After that click submit	Email=’princeashifulislam@gmail.com’ name=’prince’ contact=’01944234532	Registration has been complete successfully	Pass
TU02	Registration	Check user registration with invalid data	1) Go to site 2) Enter name	Email=’princeashifulislam@gmail.com’ Password=	Password failed is required	Pass

			3) Then enter email 4) Enter contact 4) After that click submit			

5.3.3. TEST CASE 3

Table 5. 4: Test Case 3 (Manage Profile)

TEST CASE ID	TEST CASE NAME	TEST SCENARIO	TEST STEPS	TEST DATA	EXPECTED RESULTS	PASS/FAIL
TU01	Manage Profile	Check manage profile with valid data	1) Go to site 2) Enter skill 3) Enter interests 4) Enter bio 4) After that click submit	skill=java interests=programming bio=passionate about programming	Profile has been created successfully	Pass
TU02	Manage Profile	Check manage profile with invalid data	1) Go to site 2) Enter skill 3) Enter interests 4) Enter bio 4) After that click submit	skill=java interests= bio=passionate about programming	Interests fields are required	Pass

5.3.4. TEST CASE 4

Table 5. 5: Test Case 4 (Manage Service)

TEST CASE ID	TEST CASE NAME	TEST SCENARIO	TEST STEPS	TEST DATA	EXPECTED RESULTS	PASS/FAIL
TU01	Manage Services	Check manage service with valid data	1) Go to site 2) Enter service type 3) Enter name 4) Enter package 4) After that click submit	Type=Can search Name= search Package = complete	Service has been created successfully	Pass
TU02	Manage Services	Check manage service with invalid data	1) Go to site 2) Enter service type 3) Enter name 4) Enter package 4) After that click submit	Type=Can search Name= search Package =	Check that whether the package is added or not in the add package page	Pass

5.3.5. TEST CASE 5

Table 5. 6: Test Case 5 (Payment)

TEST CASE ID	TEST CASE NAME	TEST SCENARIO	TEST STEPS	TEST DATA	EXPECTED RESULTS	PASS/FAIL
TU01	Payment	Check payment with valid data	1) Go to site 2) Enter amount 3) Enter name 4) Enter email 5) Enter Package info 4) After that click submit	Amount = 40000 Name = prince Email = prince@gmail.com Package=complete	Payment has been successfully completed	Pass
TU02	Payment	Check payment with invalid data	1) Go to site 2) Enter amount 3) Enter name 4) Enter email 5) Enter Package info 4) After that click submit	Amount = 40000 Name = prince Email = Package=complete	Before click to the payment button make sure that you are an authenticated user	Pass

5.3.6. TEST CASE 6

Table 5. 7: Test Case 6

TEST CASE ID	TEST CASE NAME	TEST SCENARIO	TEST STEPS	TEST DATA	EXPECTED RESULTS	PASS/FAIL
TU01	Manage Job	Check manage job with valid data	1) Go to site 2) Enter job_type 3) Enter desc 4) Enter vacancy 4) After that click submit	Job_type = full=time Desc= minimum 2 years of experience Vacany=3	Job has been posted successfully	Pass
TU02	Manage Job	Check manage job with invalid data	1) Go to site 2) Enter service type 3) Enter name 4) Enter package 4) After that click submit	Job_type = Desc= minimum 2 years of experience Vacany=3	Make sure that you have added a job type before you post something	Pass

5.3.7. TEST CASE 7

Table 5. 8: Test Case 7 (Verify Agent)

TEST CASE ID	TEST CASE NAME	TEST SCENARIO	TEST STEPS	TEST DATA	EXPECTED RESULTS	PASS/FAIL
TU01	Verify Agent	Check verify agent with valid data	1) Go to site 2) check license 3) Check email 4) After that click approve or reject	License= a2455 Email= agent1@gmail.com	Agent has been approved successfully	Pass
TU02	Verify Agent	Check verify agent with invalid data	1) Go to site 2) check license 3) Check email	License=	Agent has been rejected successfully	Pass

			4) After that click approve or reject	Email= agent1@gmail.com		

5.3.8. Testing Environment (software requirements)

- Browser: - Firefox, Google Chrome, Explorer

6. CHAPTER-05 USER MANUAL

6.1. EMPLOYER PANEL

1. An employer can create his profile by giving these required information

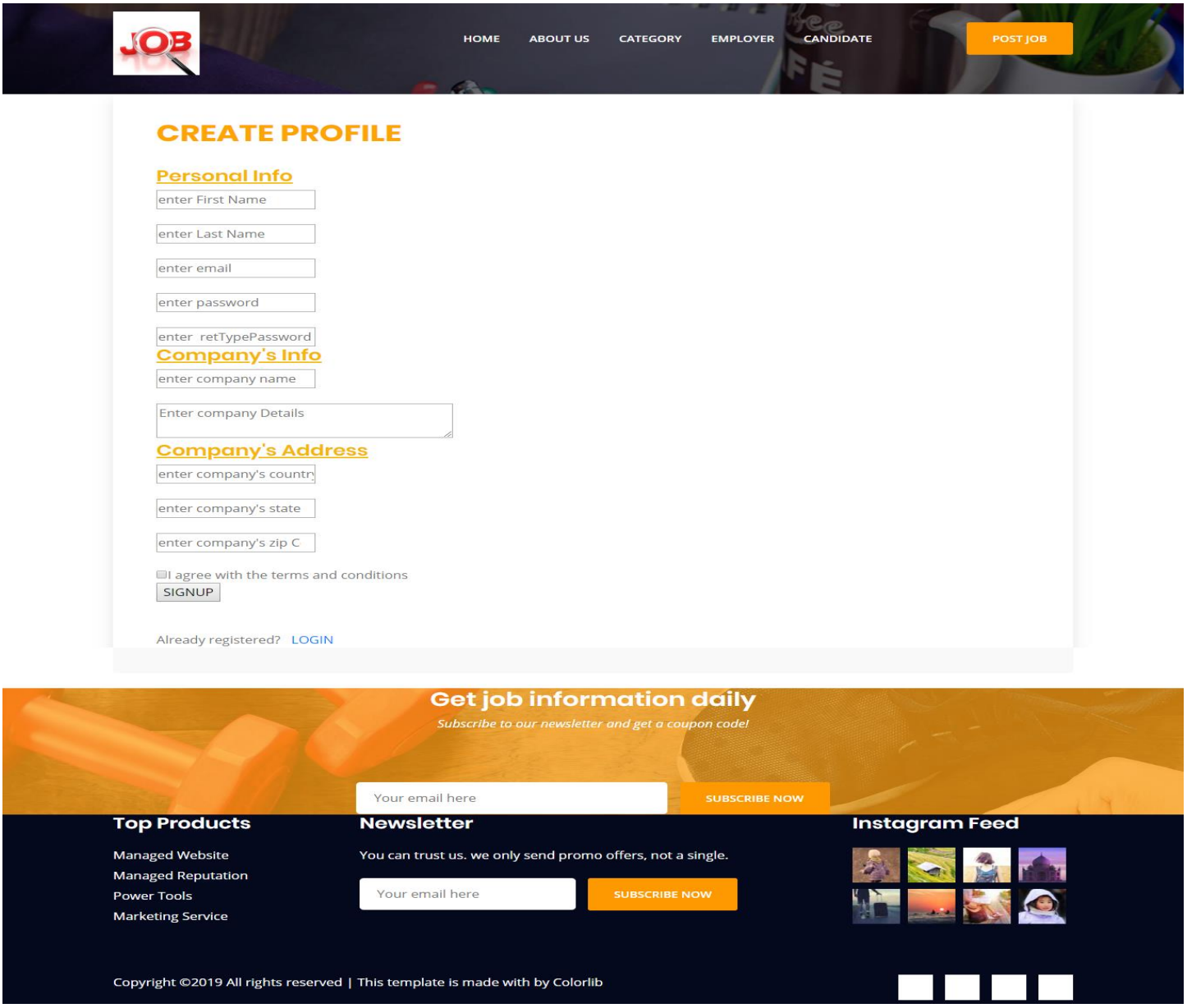


Figure 6. 1: Employer can create profile

2. An employer can update his profile easily by giving new information into this form

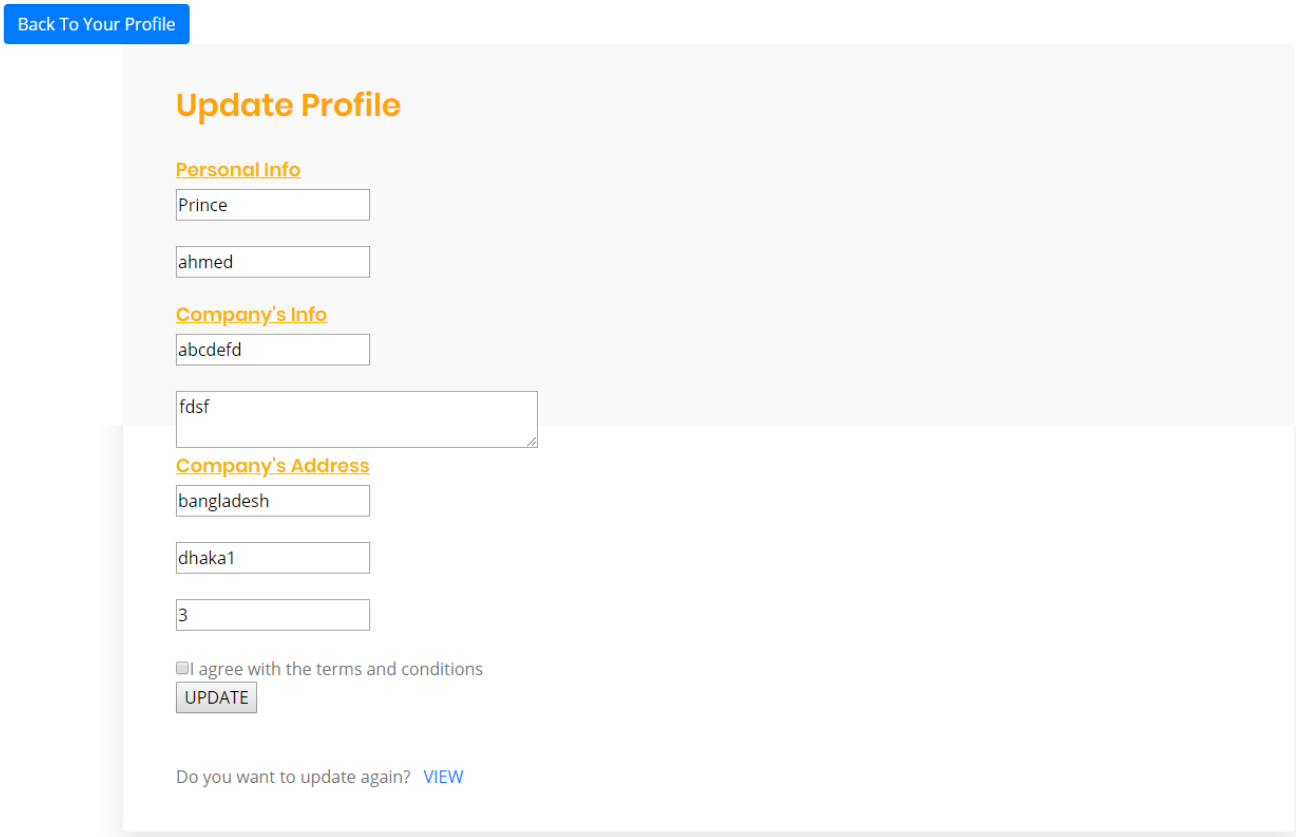


Figure 6. 2: Employer can update profile

3. An employer can add job category

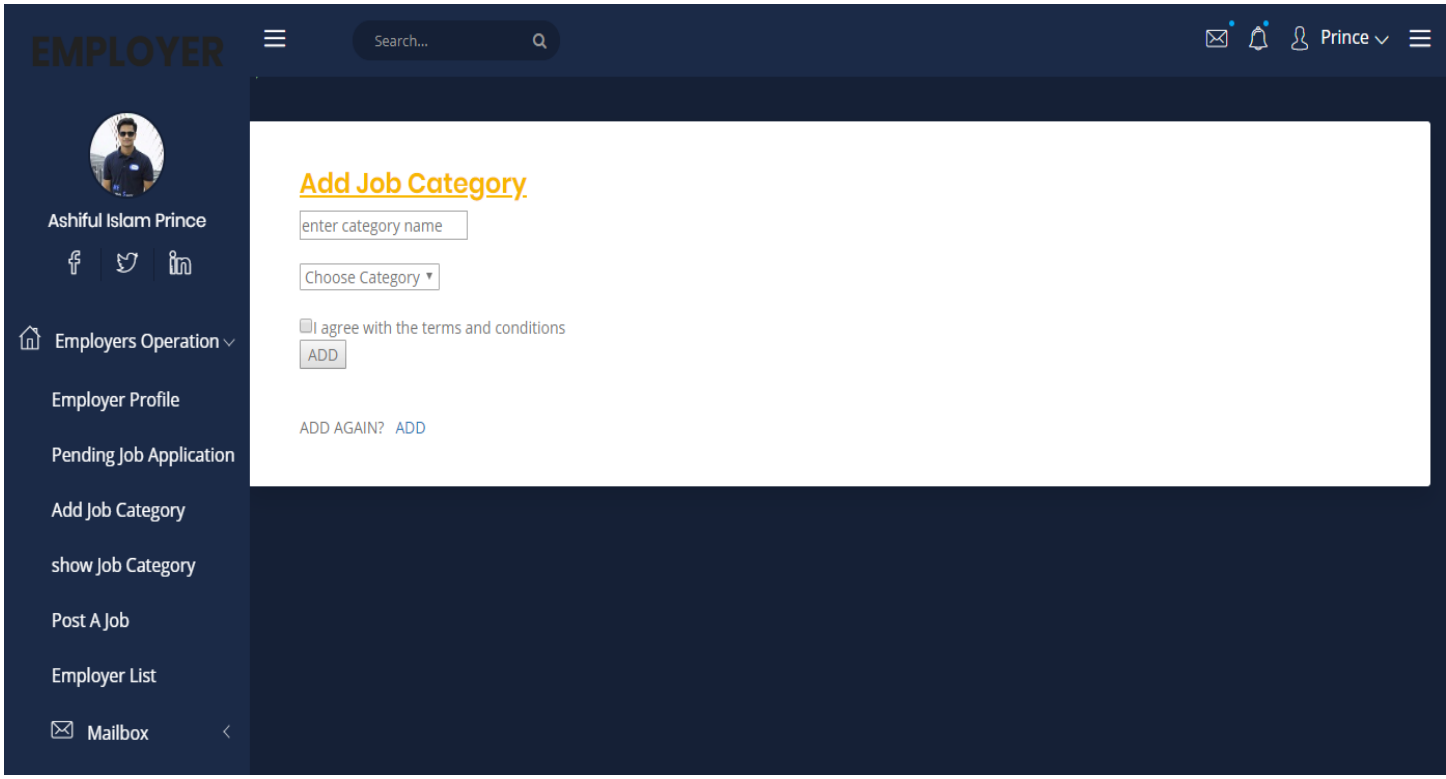


Figure 6. 3: Employer can add job category

4. An employer can post a job according to the job category

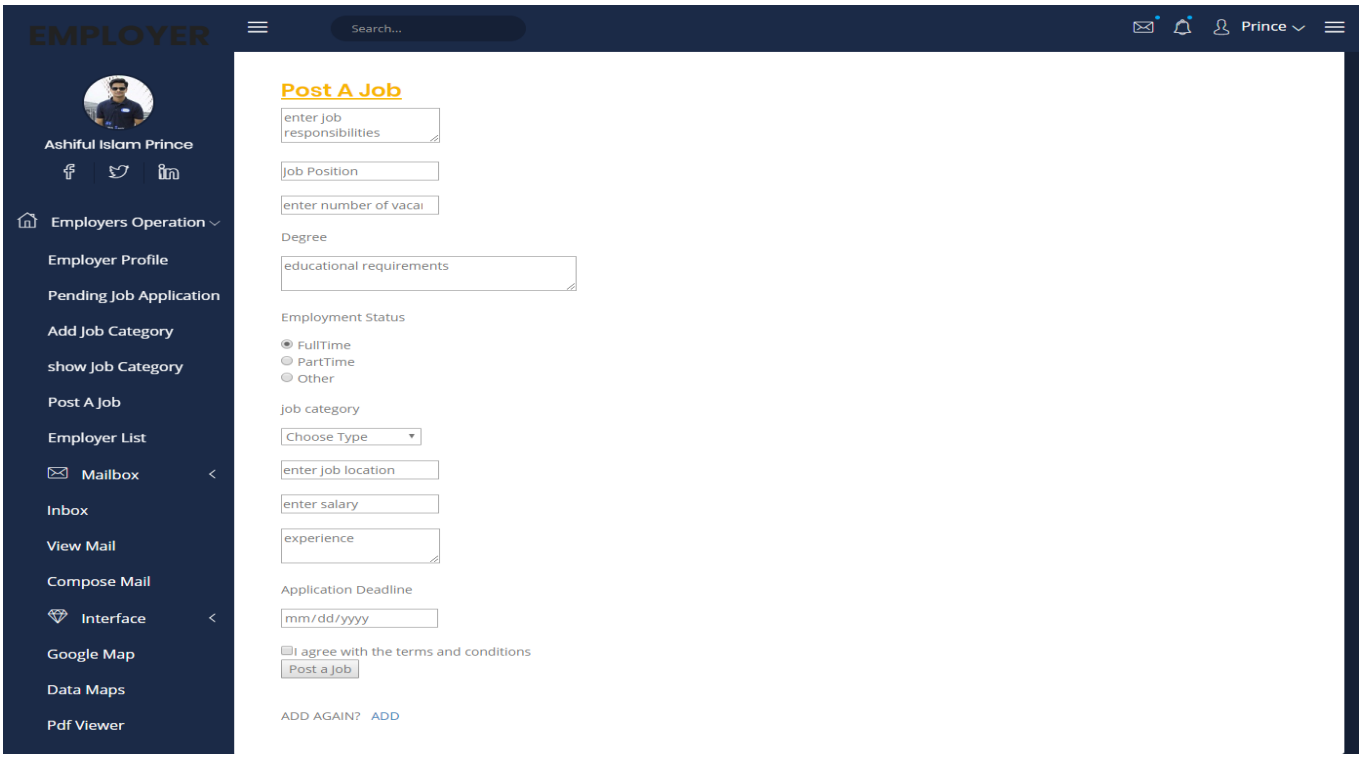


Figure 6. 4: Employer can post a job

5. An employer can easily approve the job information by seeing candidate requests

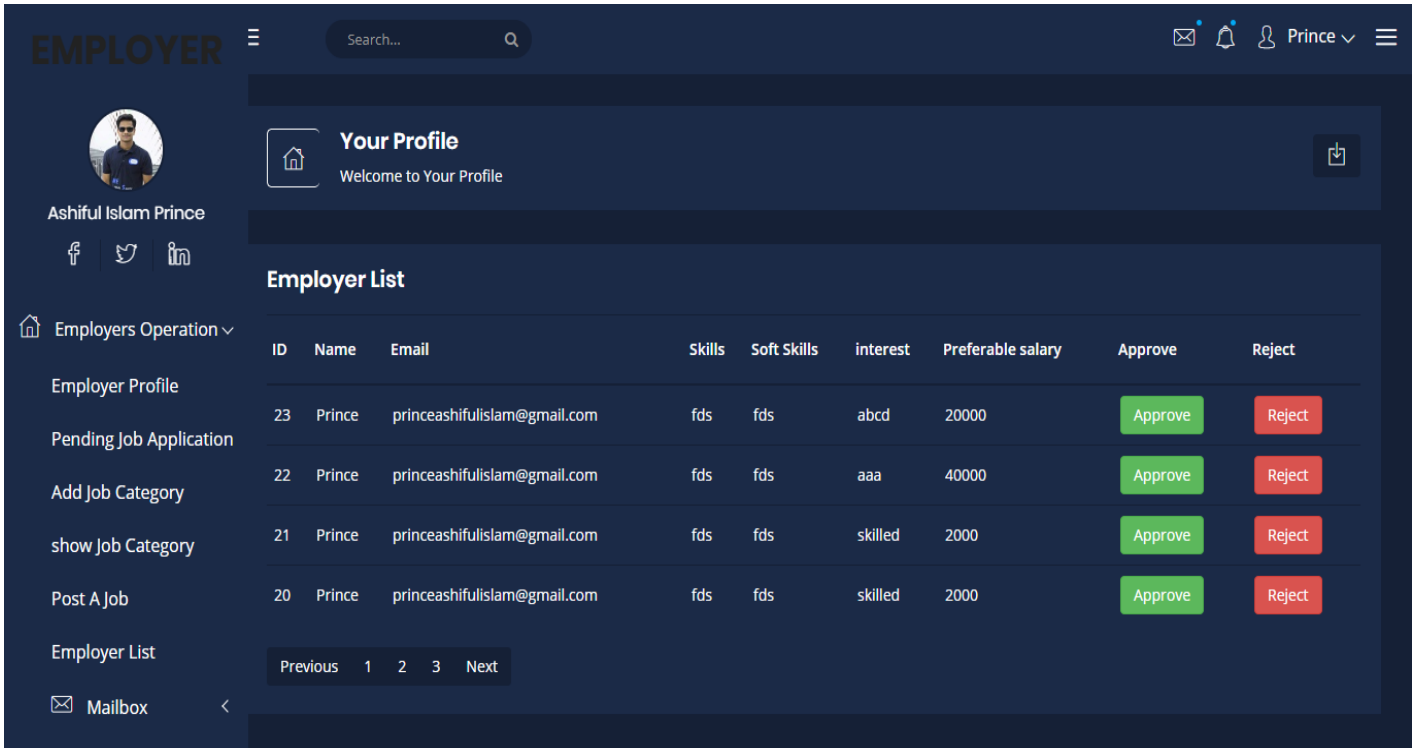


Figure 6. 5: Employer can approve a job

6. An employer can approve or reject the agent requests

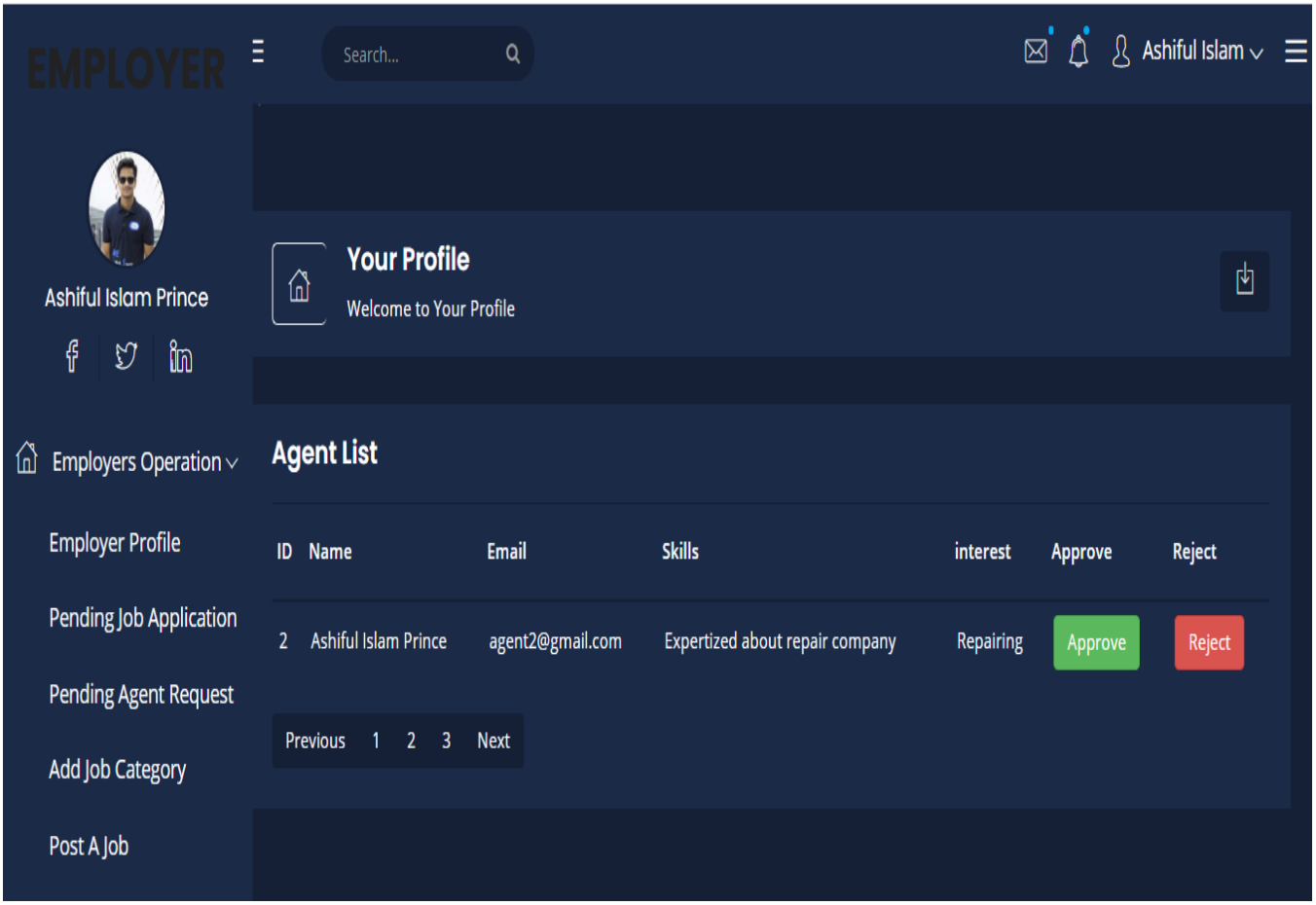


Figure 6. 6: Employer can approve agent's hiring requests

6.2. AGENT PANEL

7. An agent can sign up by giving correct information

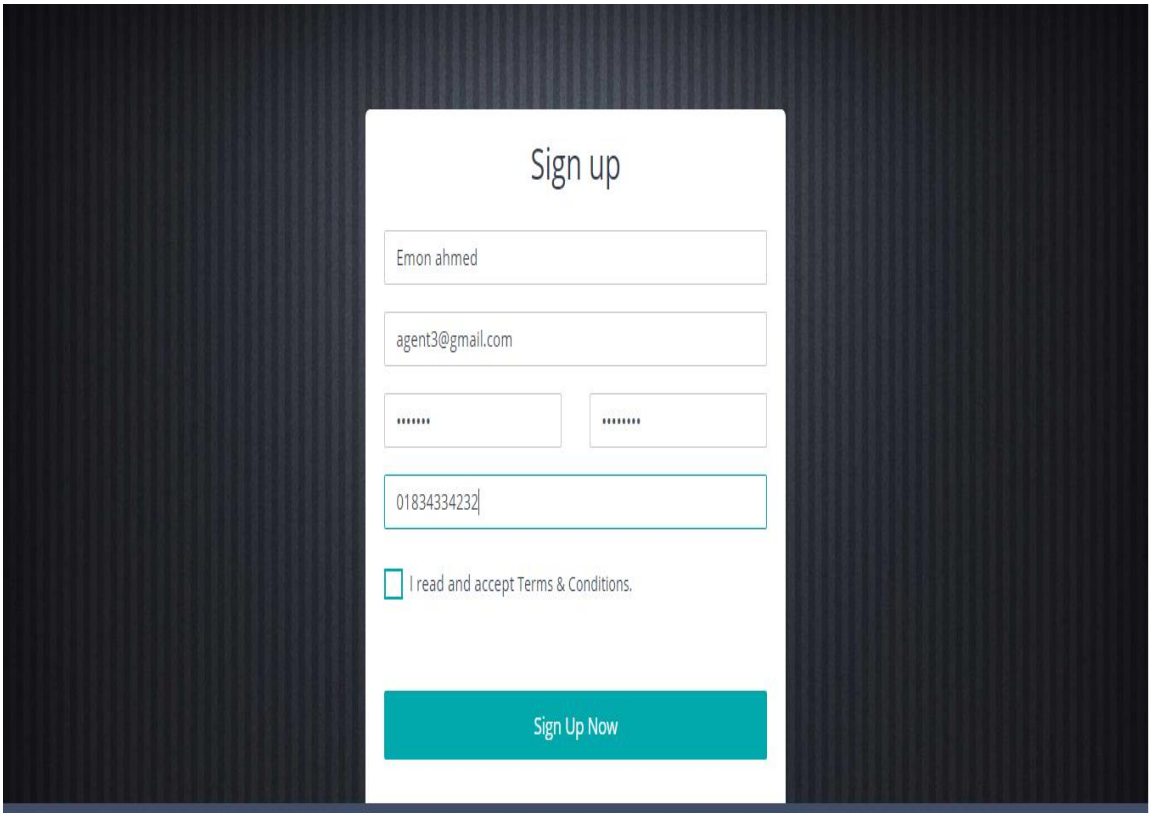


Figure 6. 7: Agent can sign up

8. An agent can log in to enter his panel

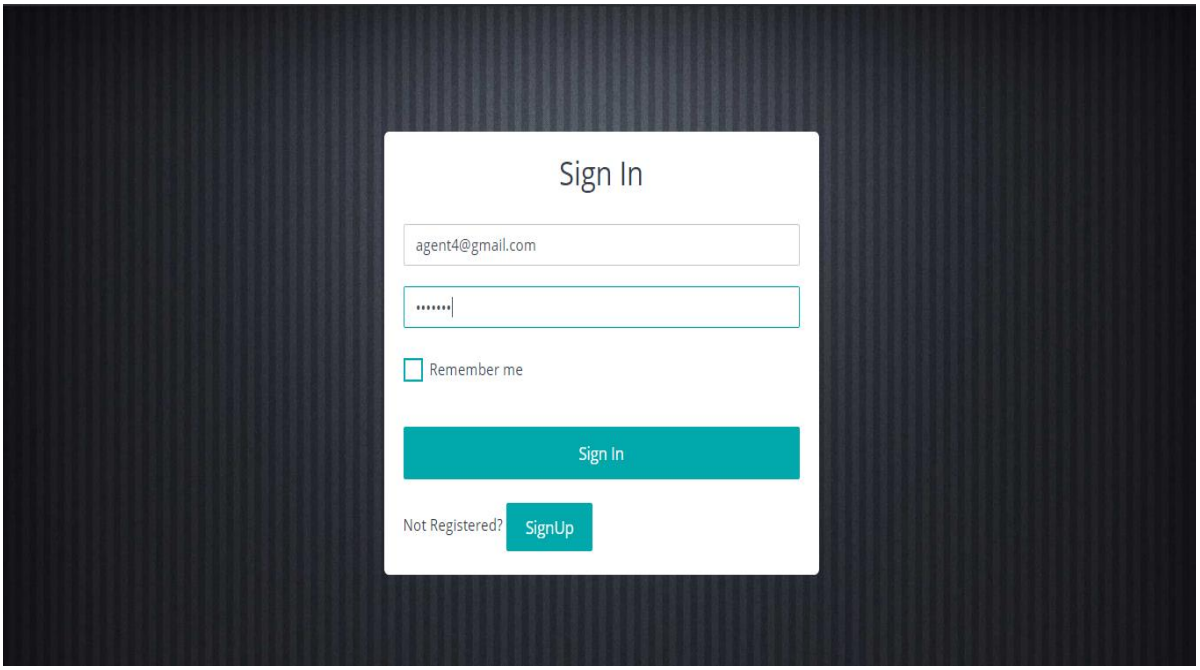


Figure 6. 8: Agent can log in

9. An agent can create, update and delete his profile

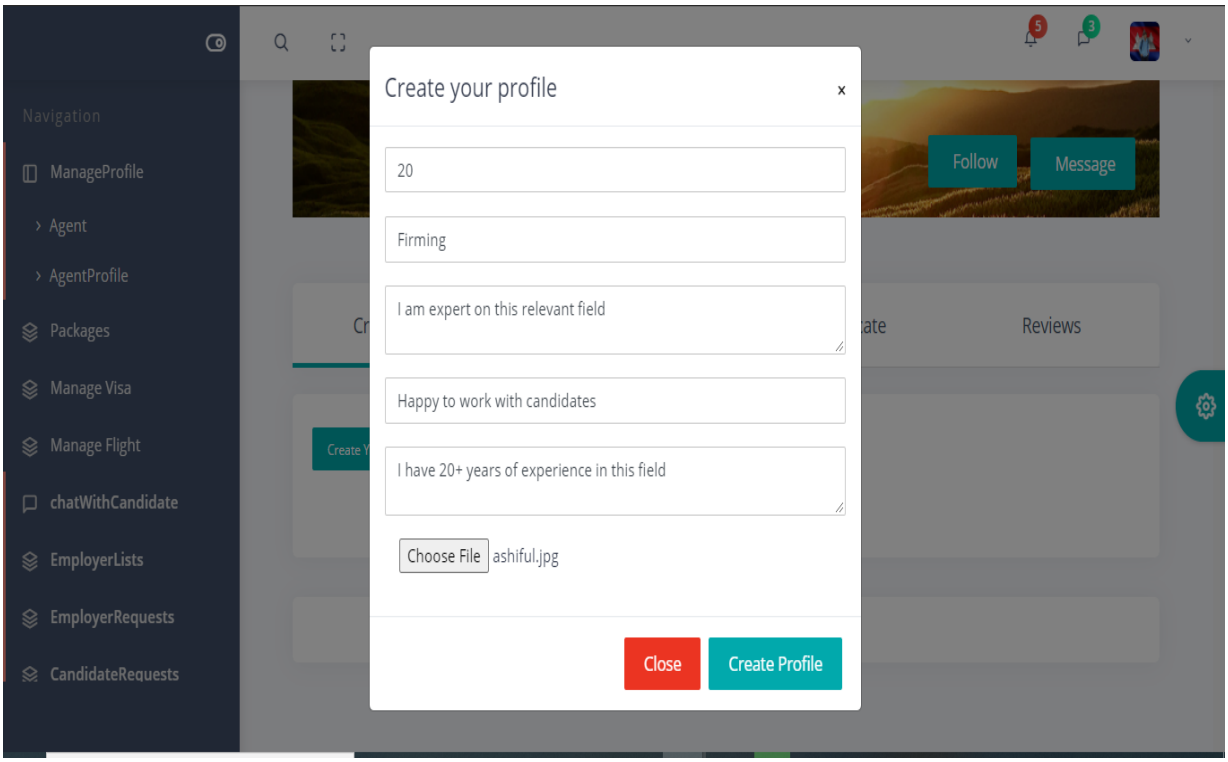


Figure 6. 9: Agent can manage profile

10. An agent can navigate his own profile

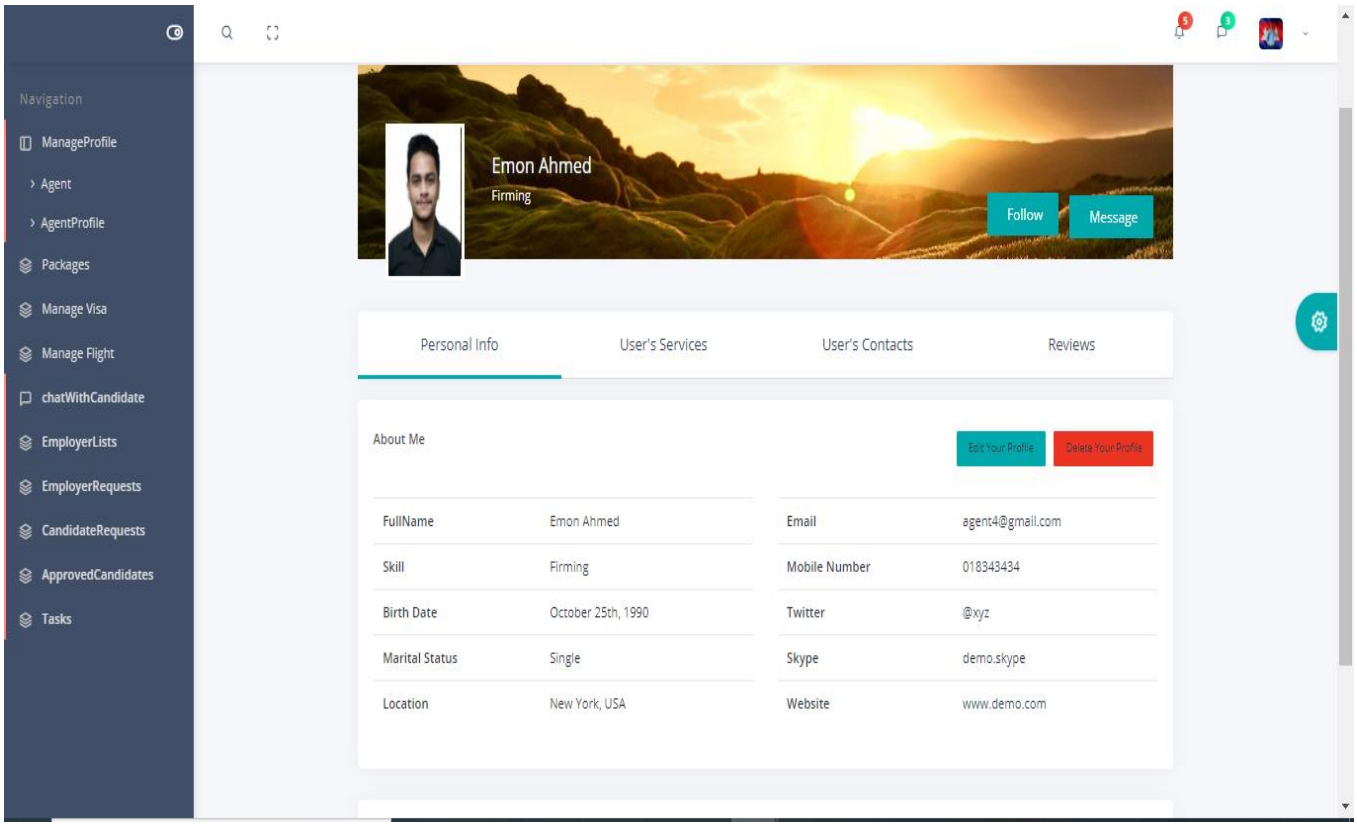


Figure 6. 10: Agent can navigate his own profile

11. An agent can add, update, delete service type

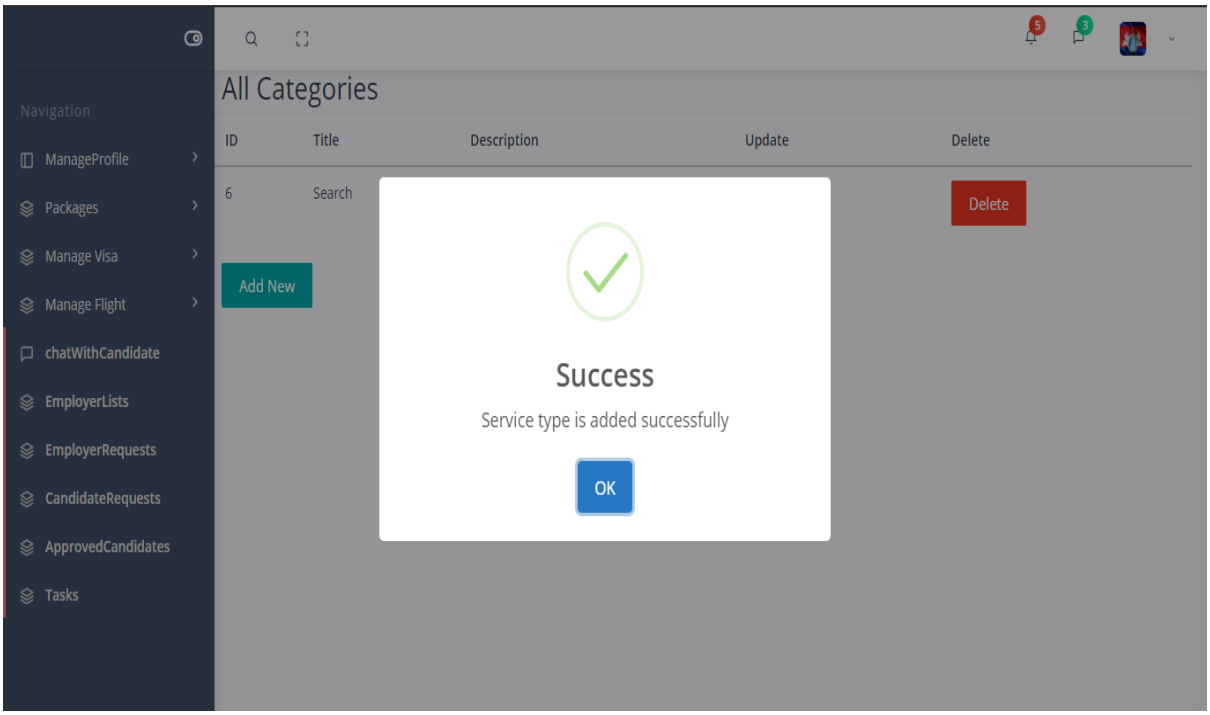


Figure 6. 11: Agent can manage service

12. An agent can add, update, delete package info

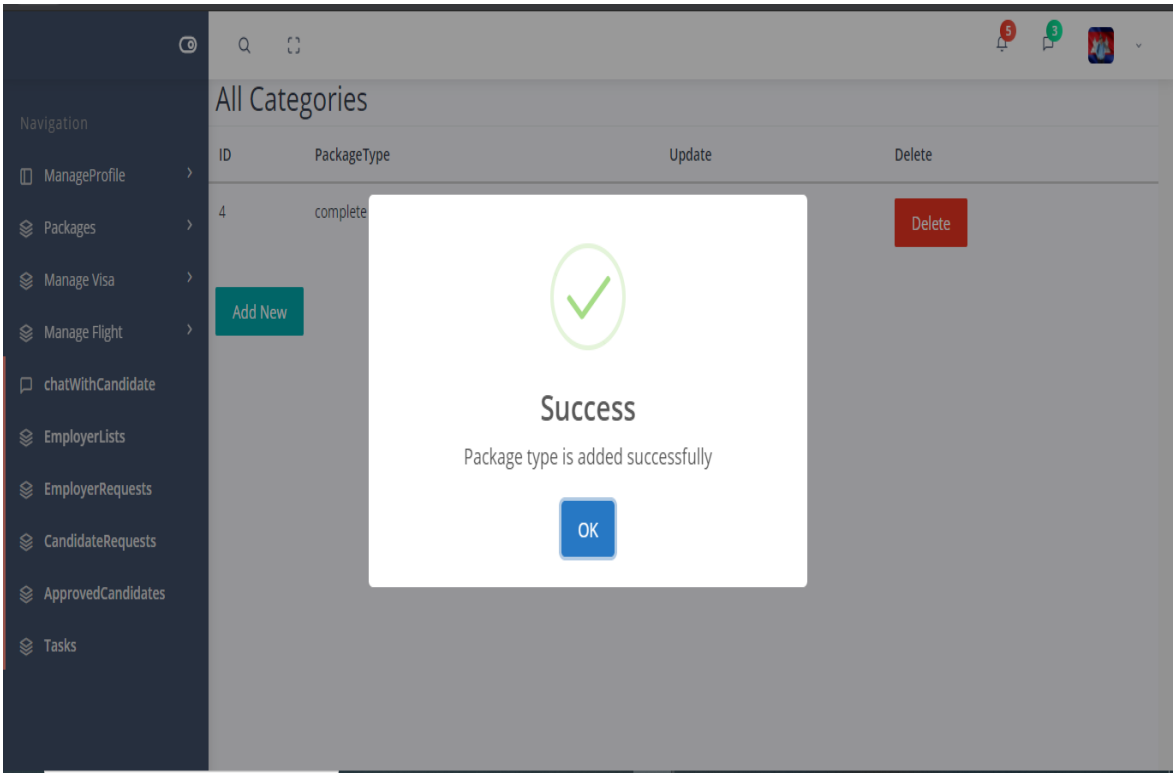


Figure 6. 12: Agent can manage package

13. An agent can post service depends of service type and package info

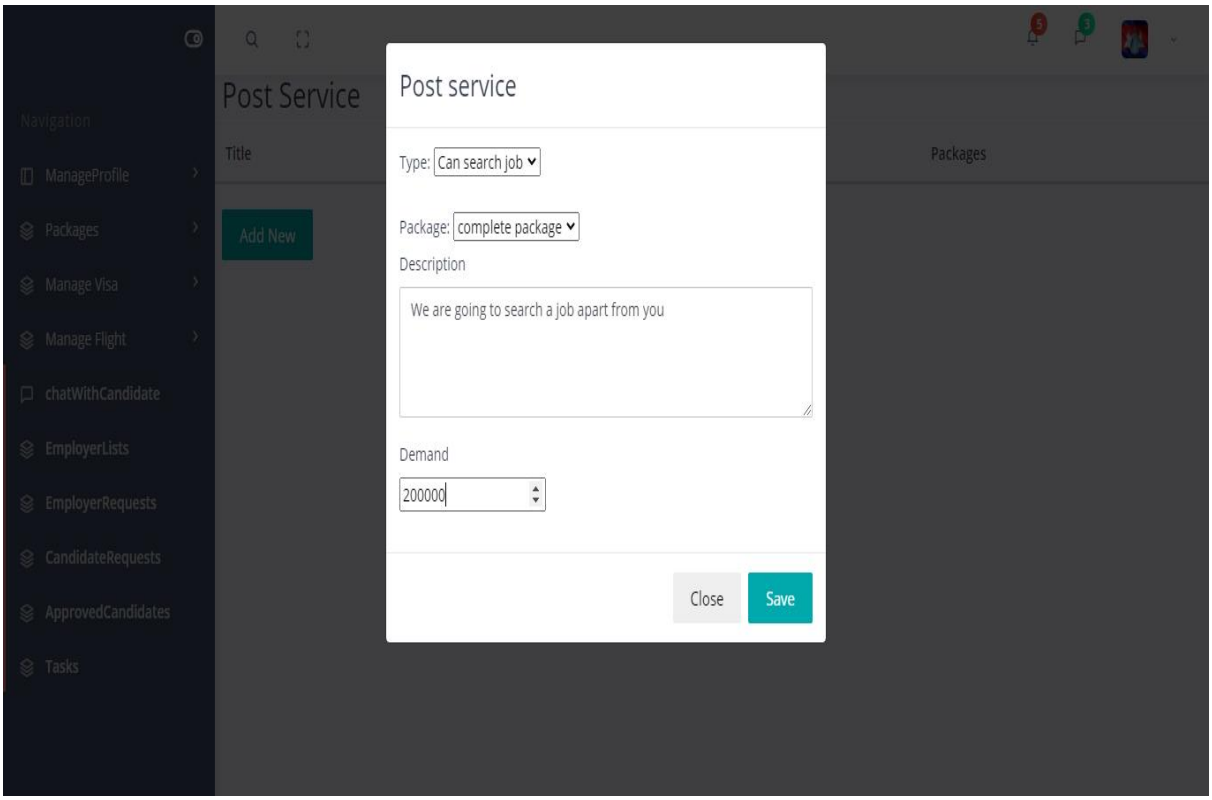


Figure 6. 13: Agent can post service

14. An agent can send request to the employer

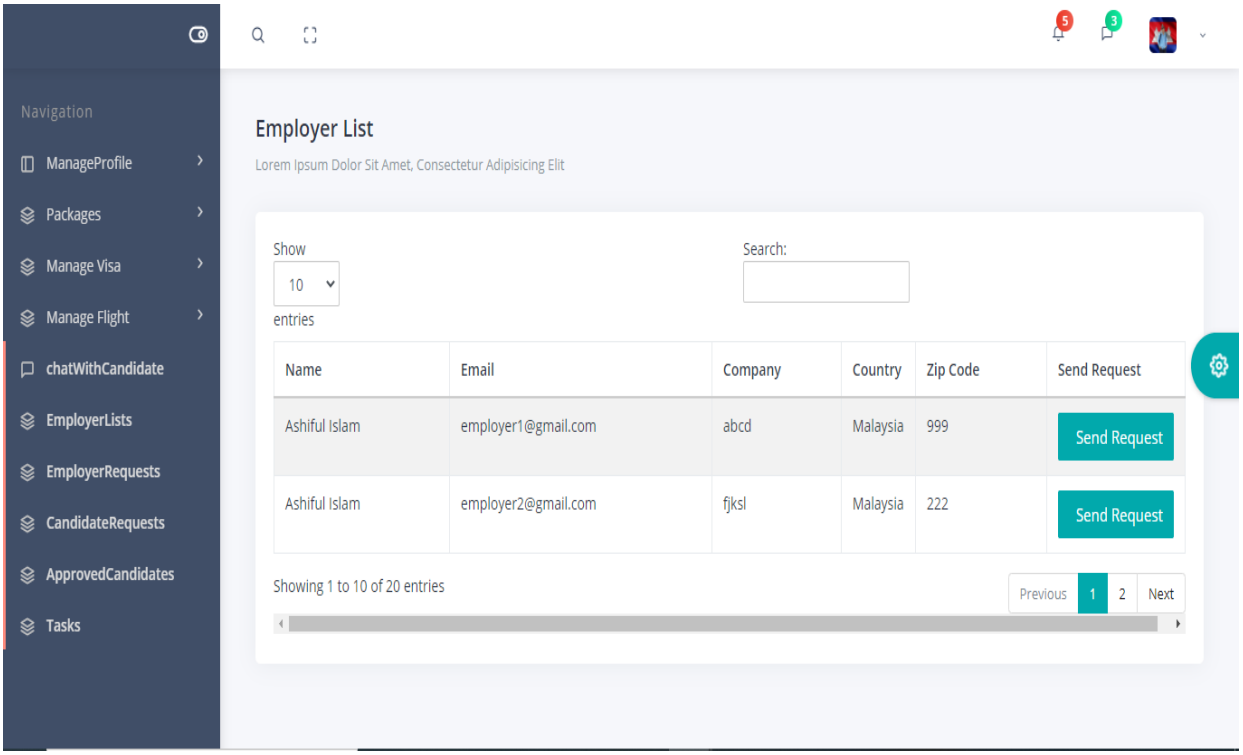


Figure 6. 14: Agent can send requests to the employer

15. An agent can approve or reject the candidate after the candidate makes the hiring requests

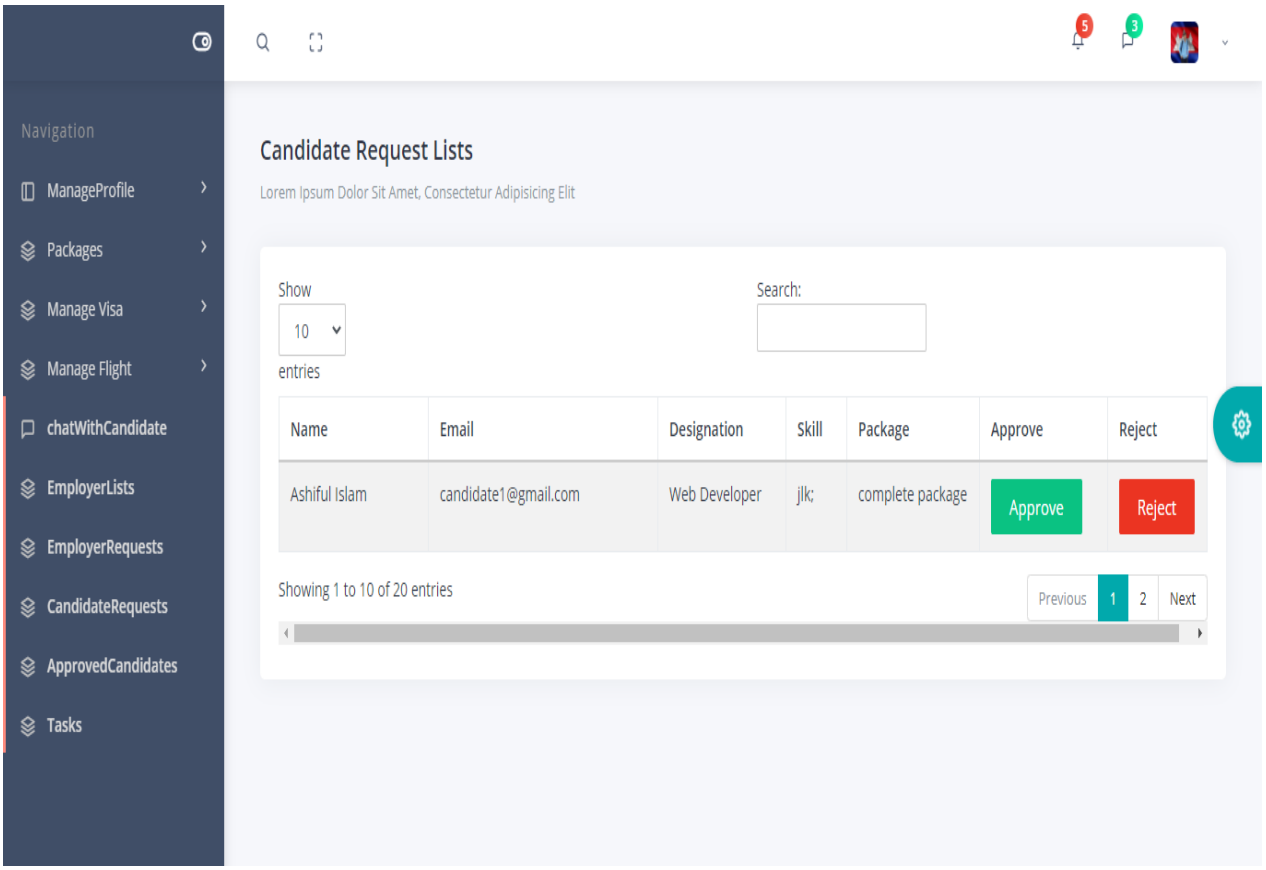


Figure 6. 15: Agent can approve candidates

16. An agent can see approved candidates

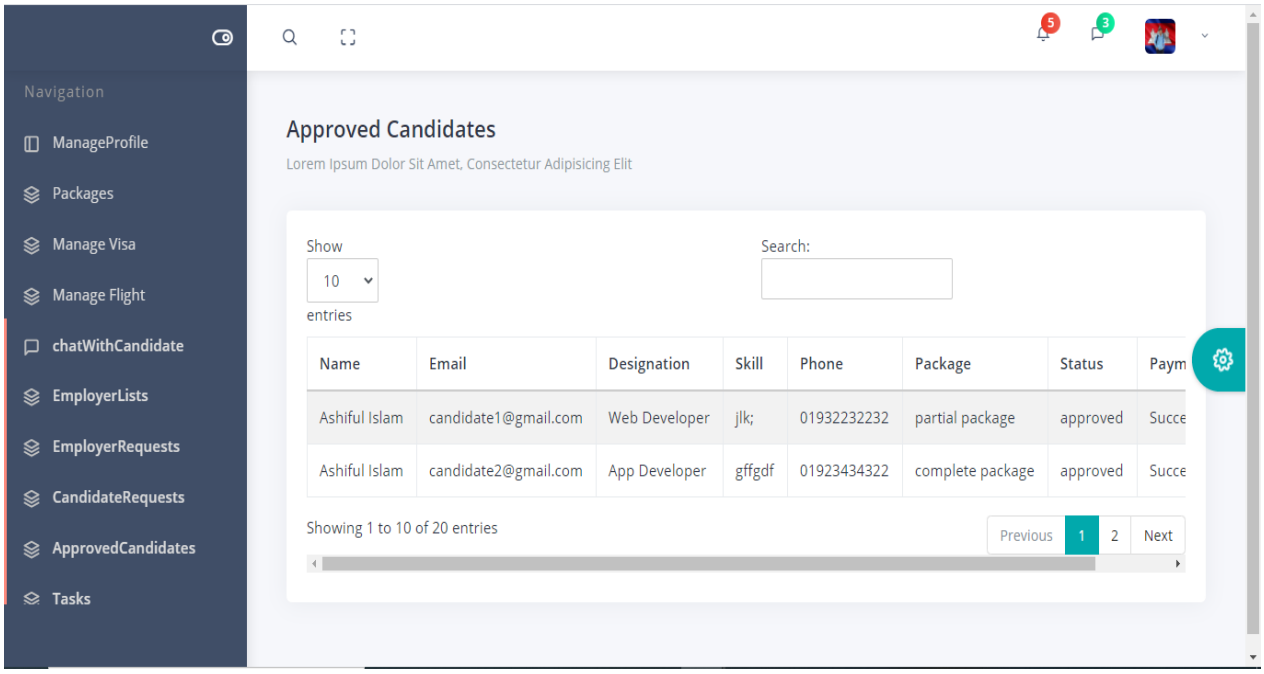


Figure 6. 16: Agent can see approved candidates

17. An agent is going to his required tasks

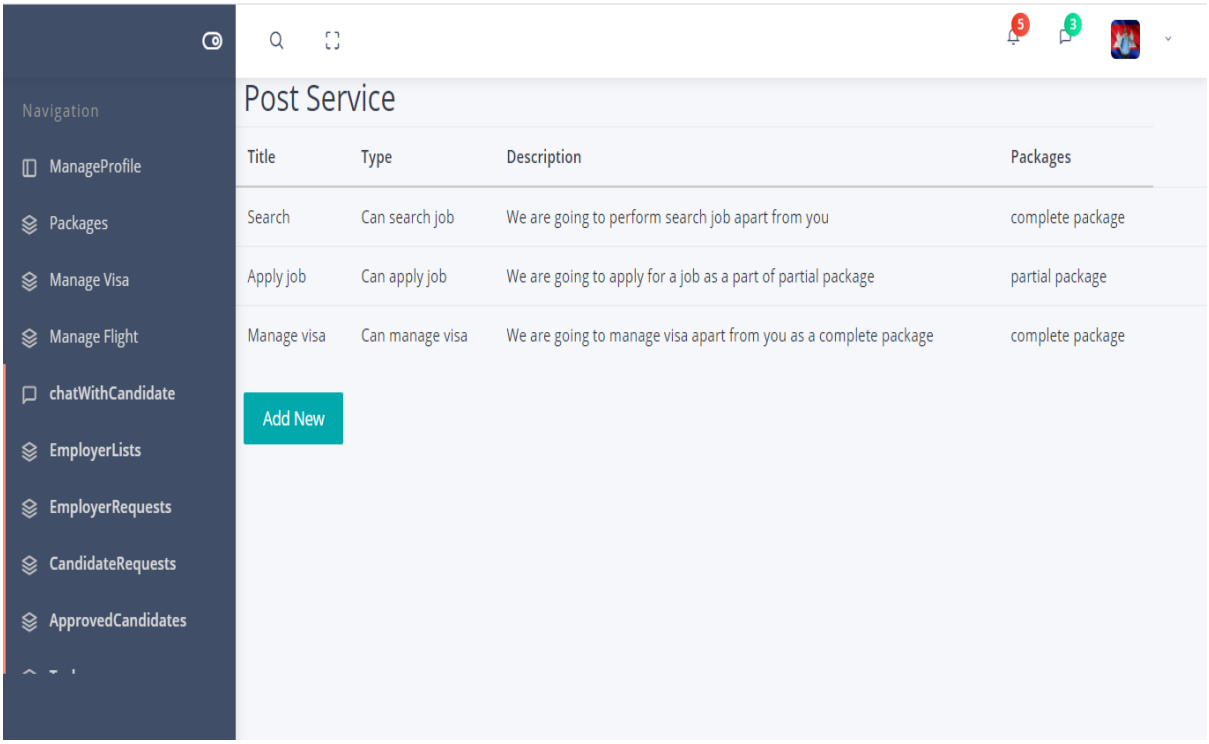


Figure 6. 17: Agent can do required tasks

18. An agent can make a communication through the live chat with the candidate after he approves the hiring requests

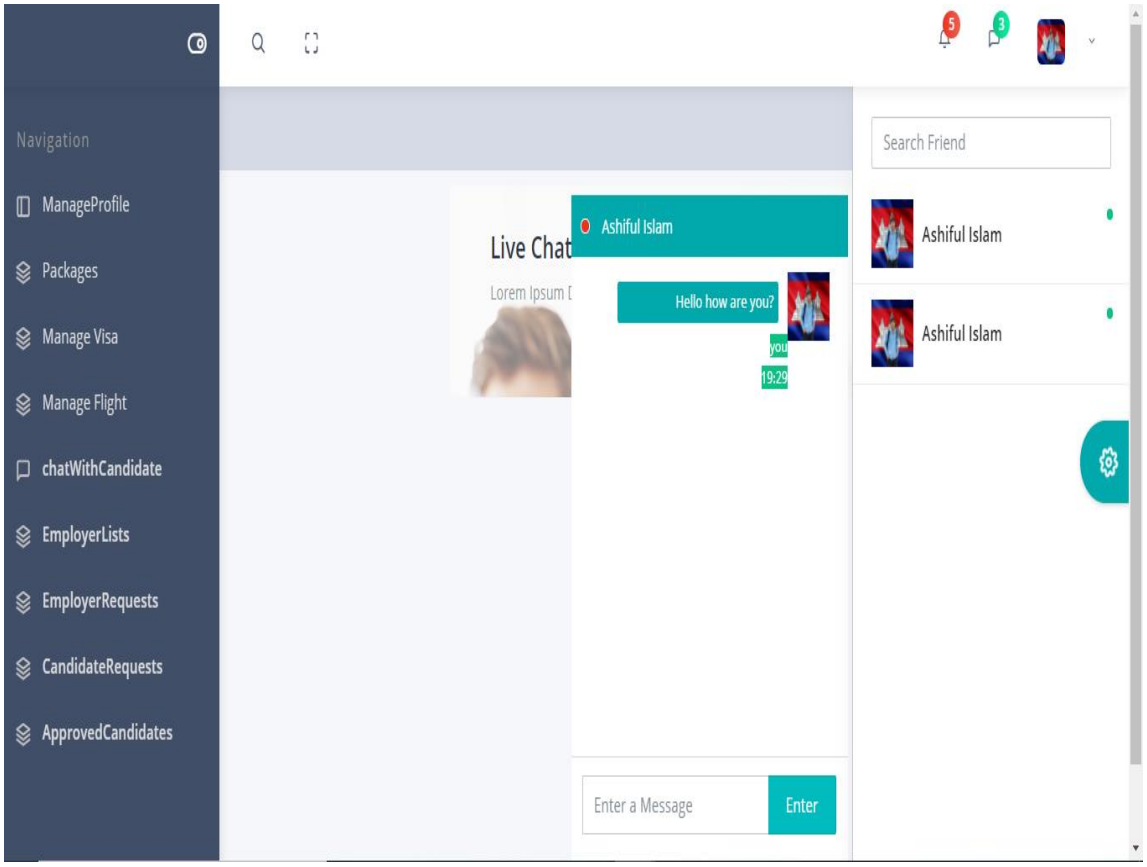


Figure 6. 18: Agent can make communication

6.3. SUPER ADMIN PANEL

19. An admin requires to log in to enter his panel

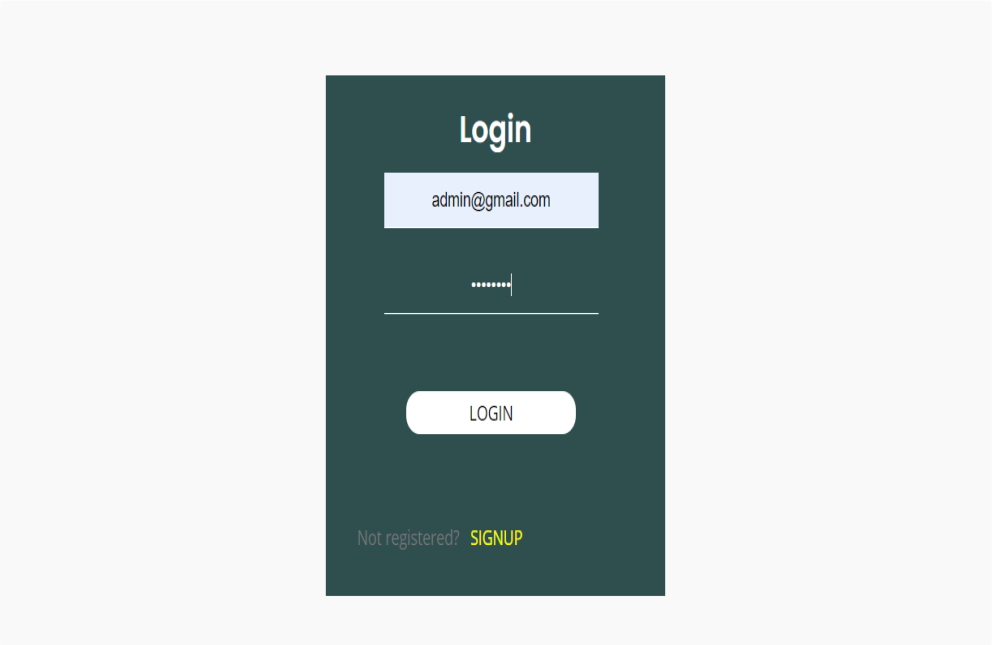


Figure 6. 19: Super admin login

20. An admin can approve job posts by seeing job information

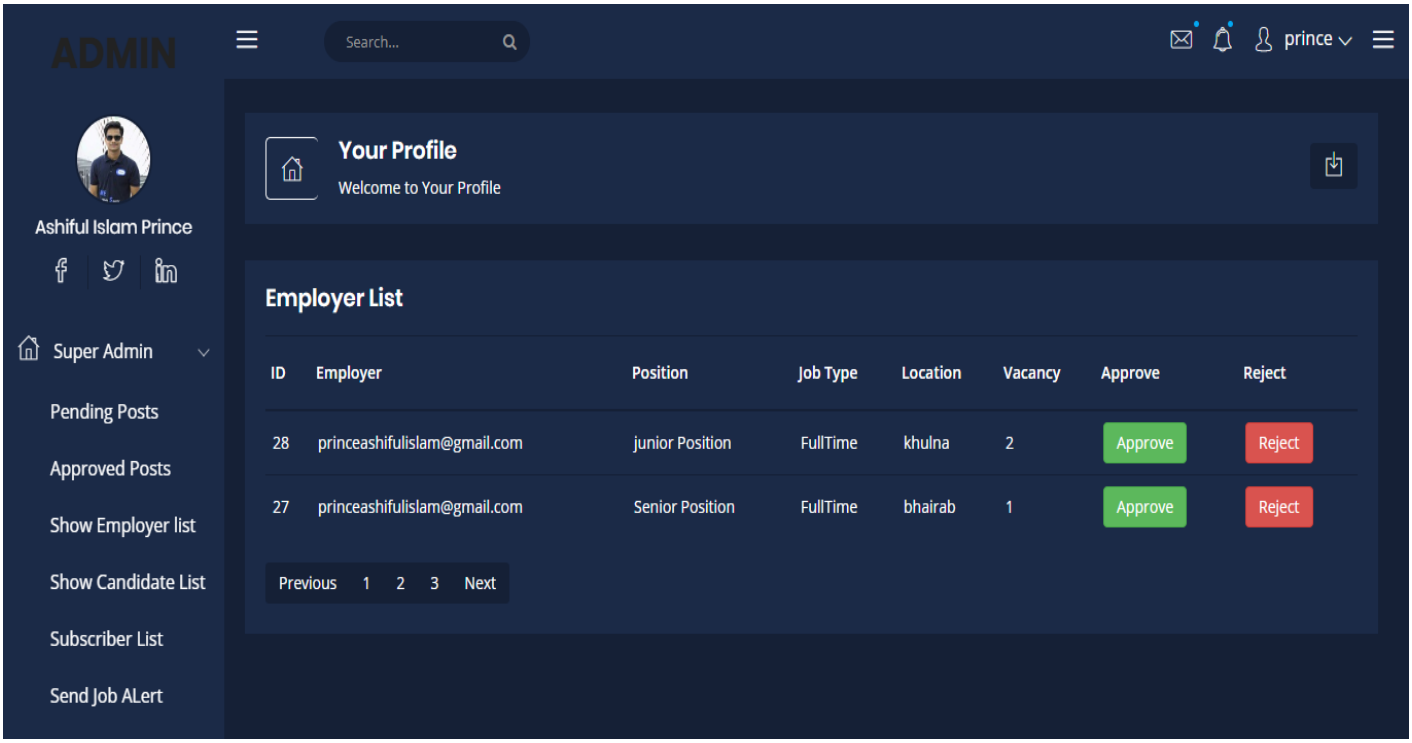


Figure 6. 20:Super admin Registration

21. An admin can see the subscriber's list

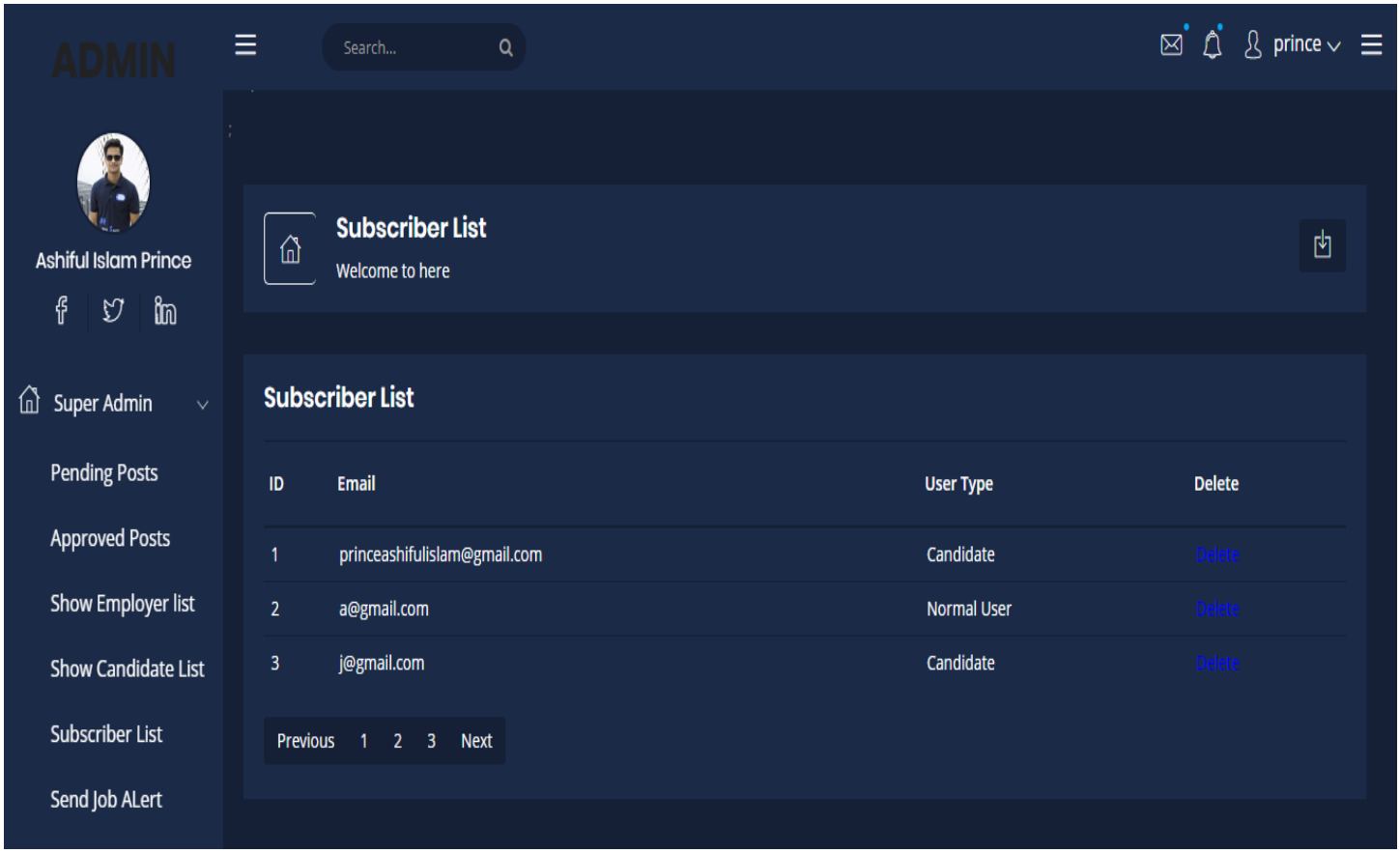


Figure 6. 21:Super admin can see subscriber list

22. An admin can see candidate lists

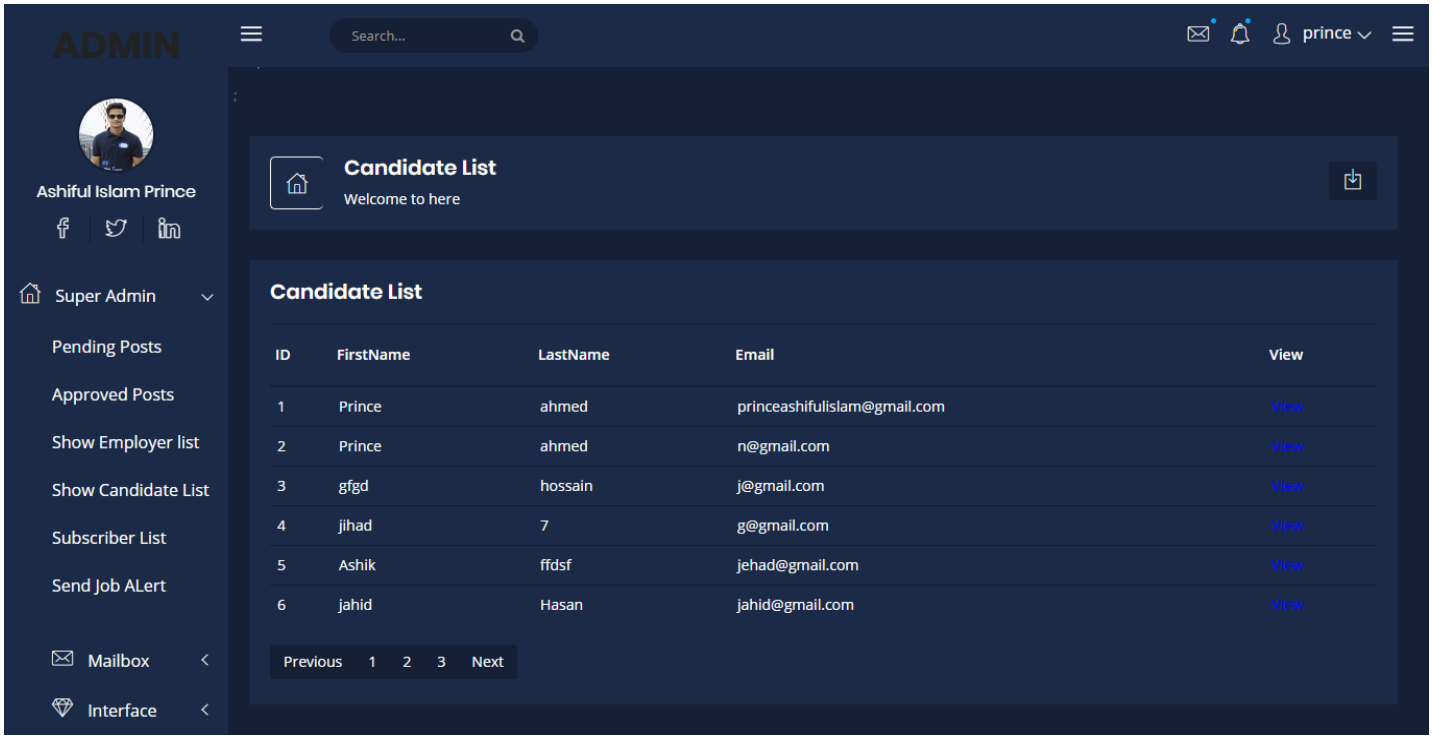


Figure 6. 22: An admin can see candidate list

6.4. CANDIDATE PANEL

23. Candidate must log in to enter his panel (Lending pages)

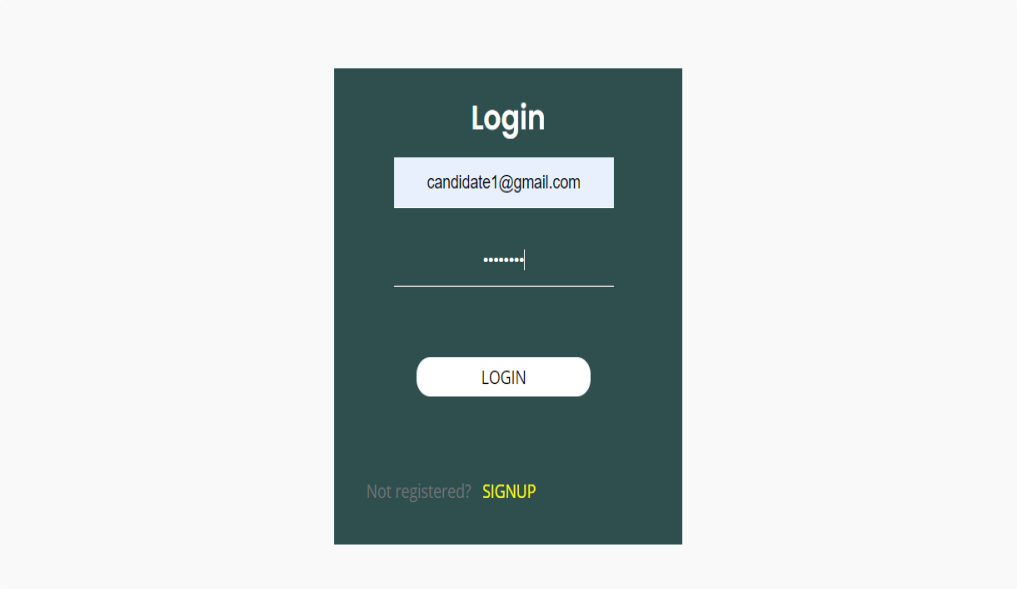


Figure 6. 23: Candidate login

24. The candidate can sign up by giving his information

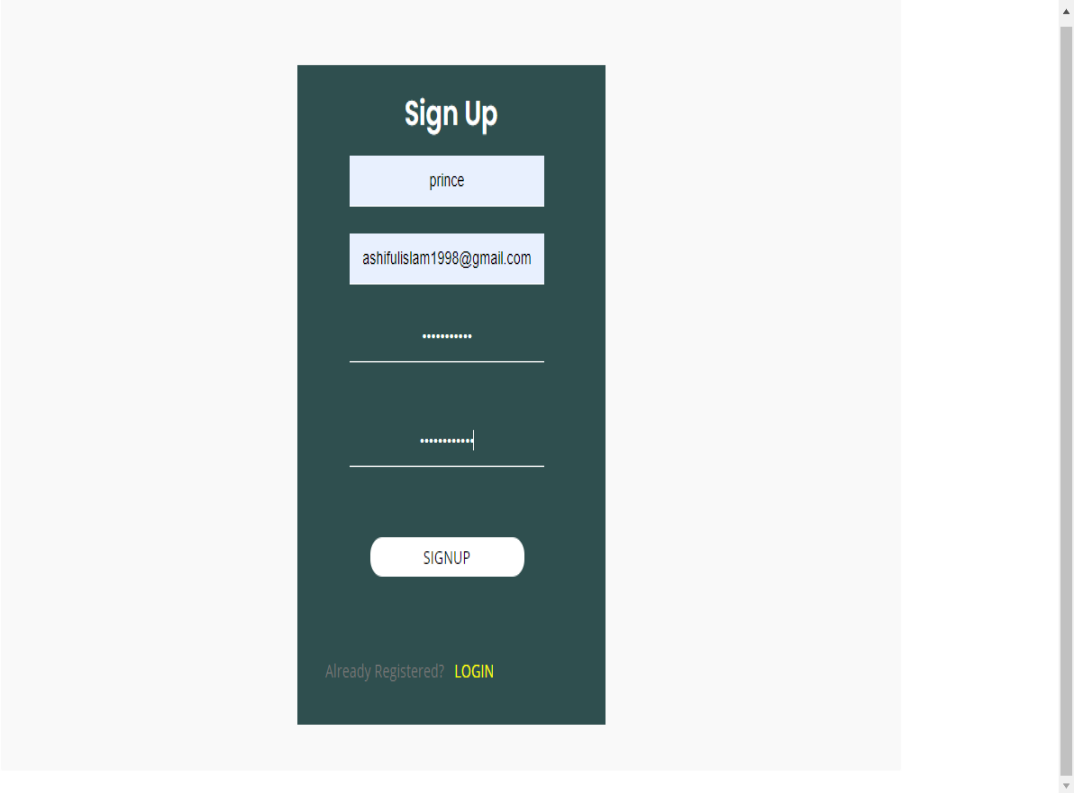


Figure 6. 24: Candidate sign up

25. Candidate can manage his own profile by creating, updating and deleting his account

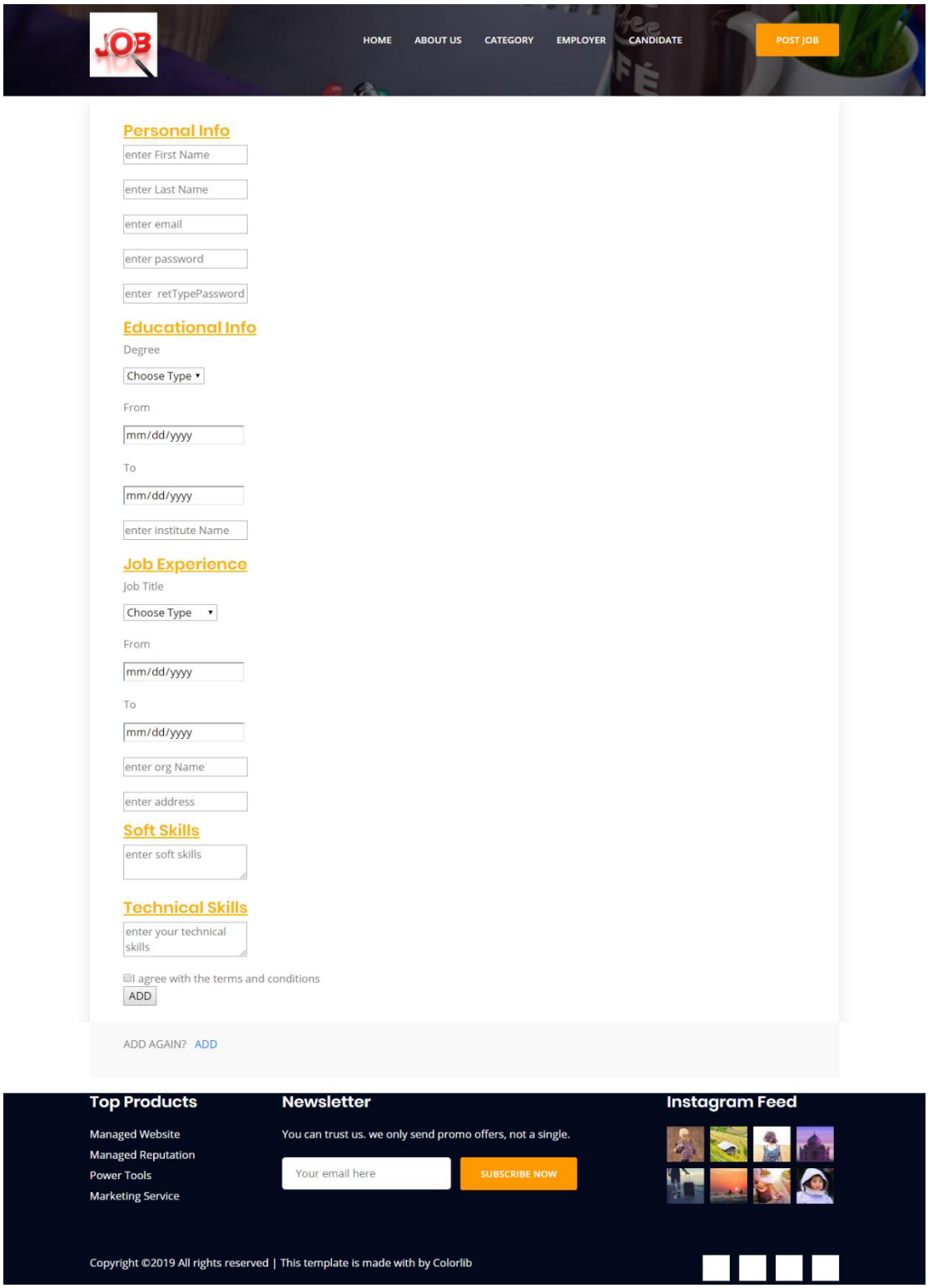


Figure 6. 25: Candidate can manage his own profile

26. A candidate can search job by the job type, location

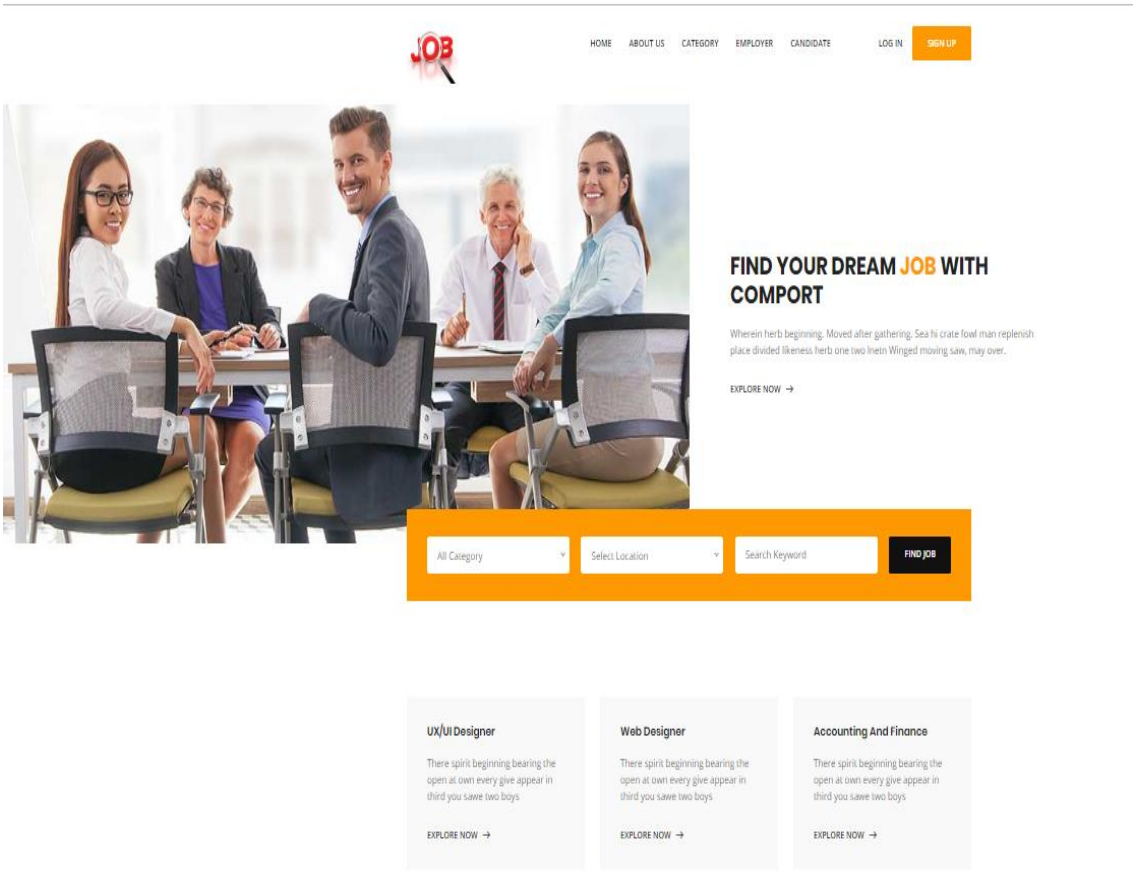


Figure 6. 26: Candidate can search a job

27. A candidate can see searched result

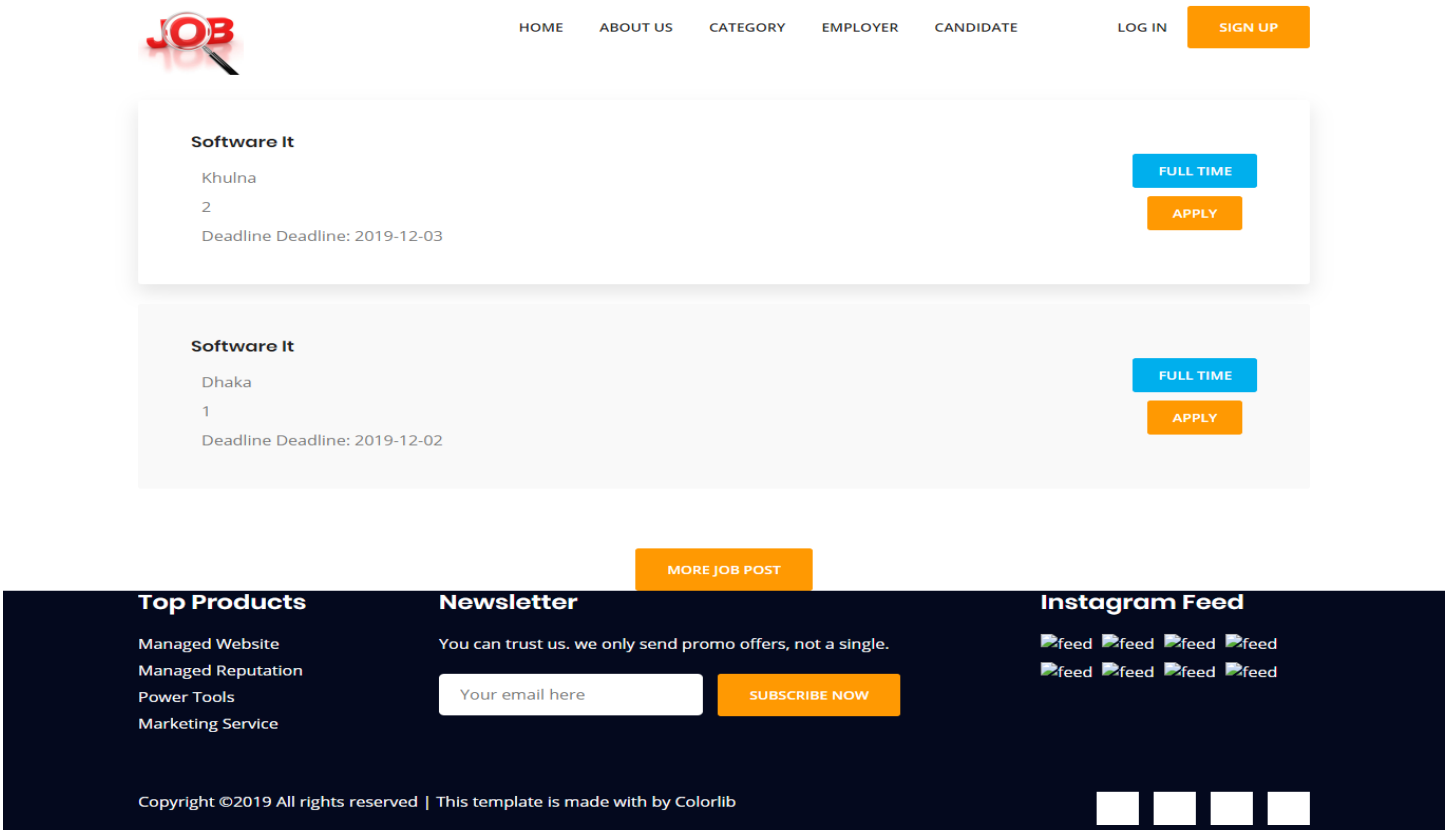


Figure 6. 27: Candidate can see searched result

28. A candidate can find their desired job through category wise

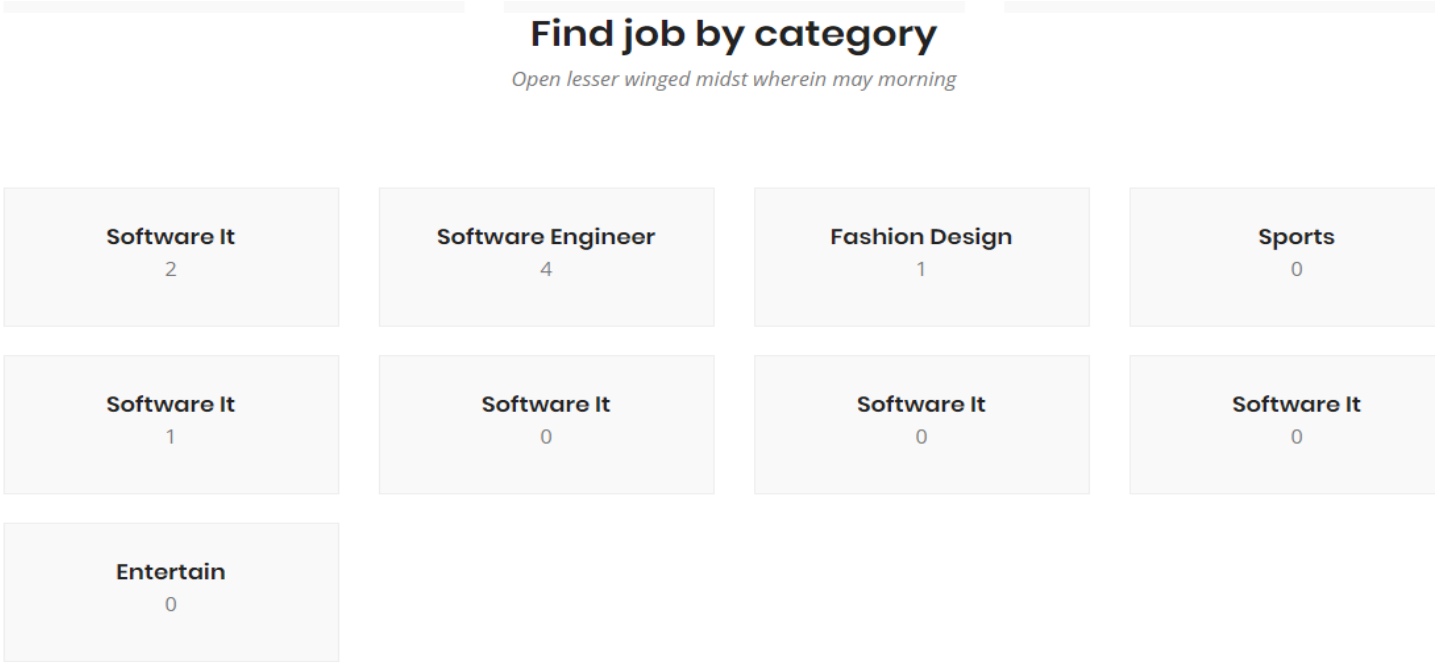


Figure 6. 28: Candidate can see job by it's category

29. A candidate can fill up the job application form

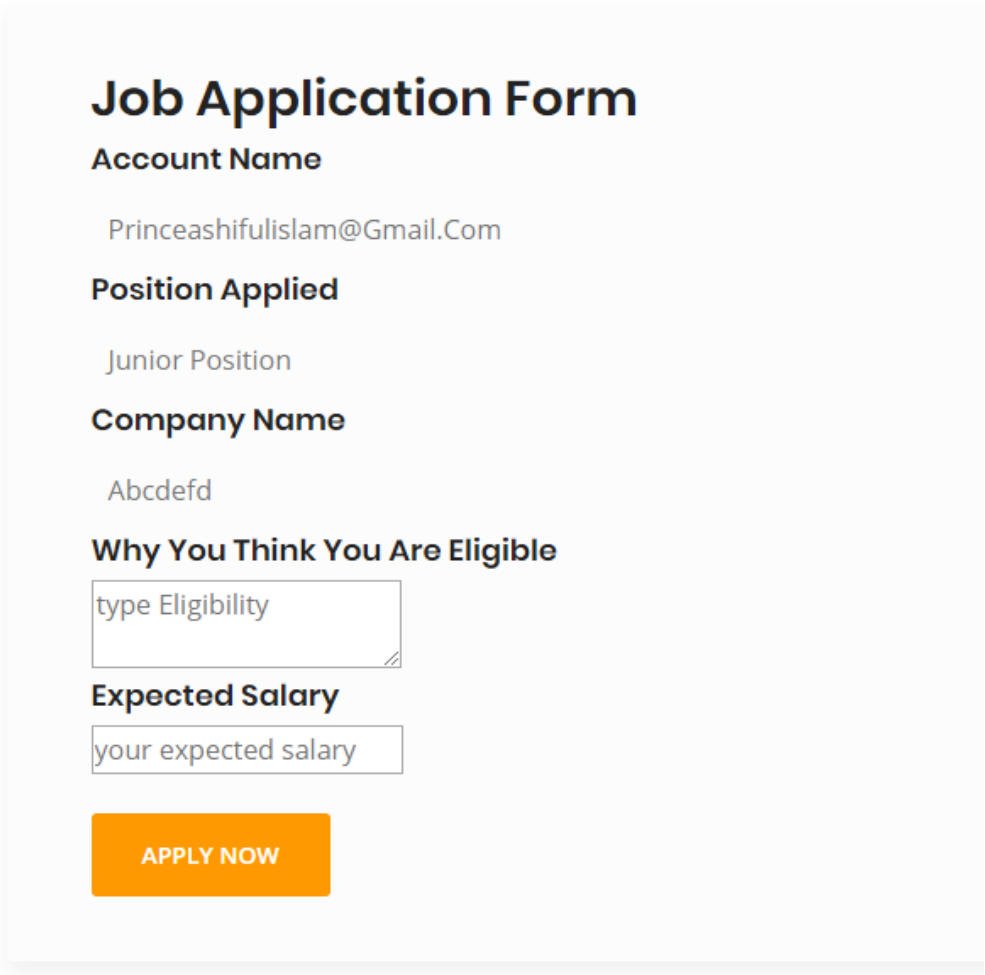


Figure 6. 29: Candidate can apply for a job

30. A candidate can see his performed agent hiring request status

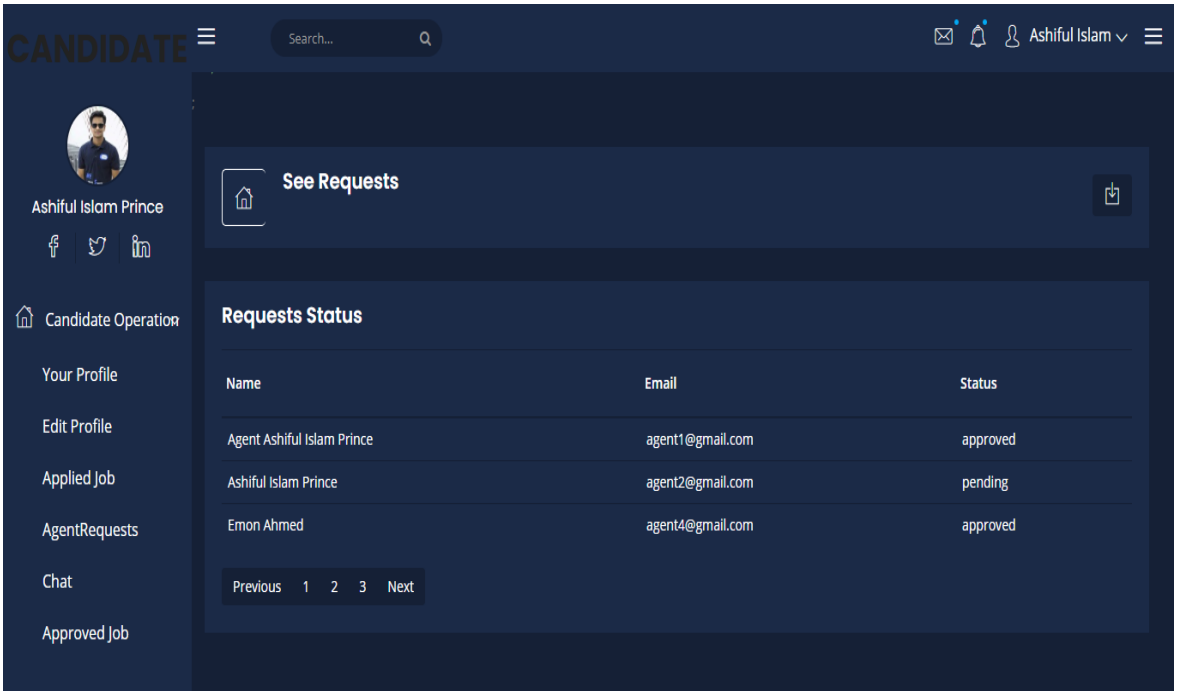


Figure 6. 30: Candidate can see agent hiring request status

31. A candidate can see his applied job status

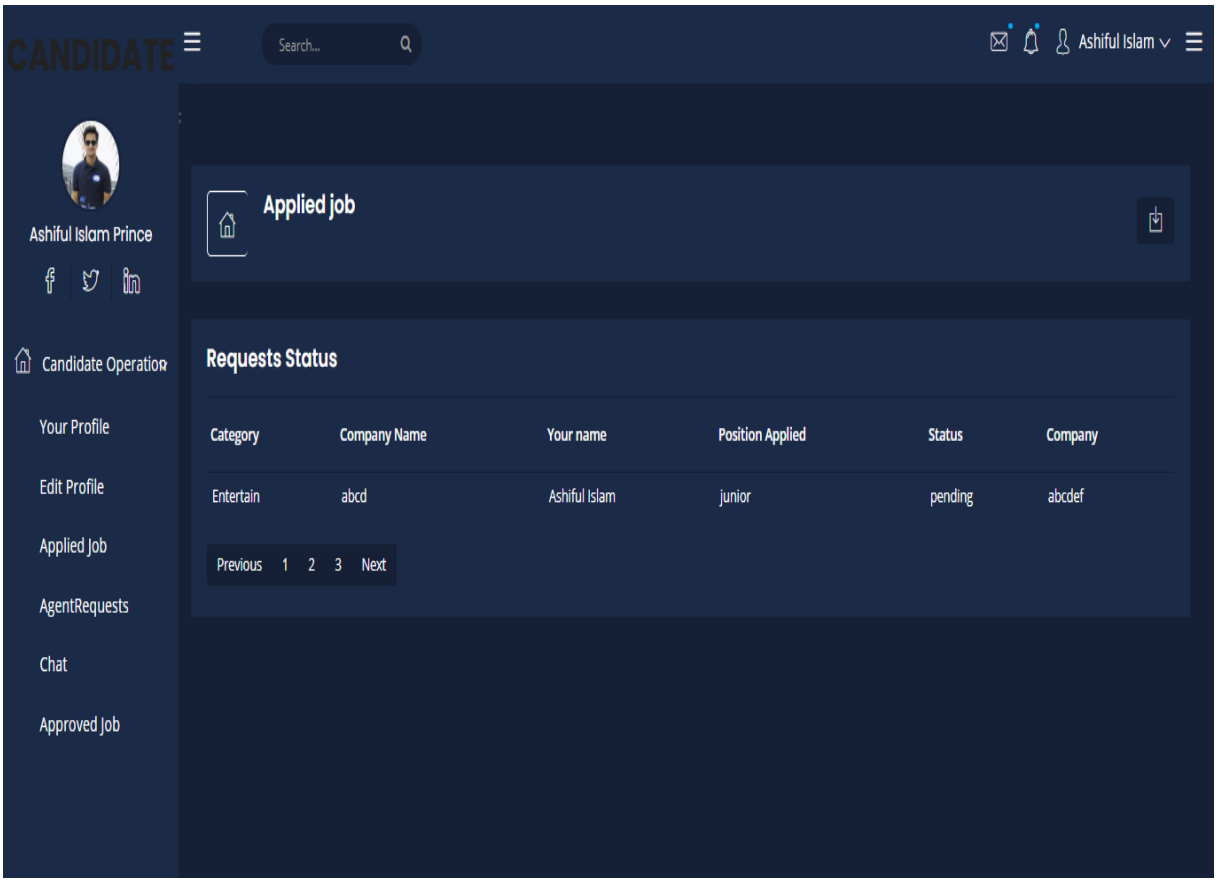


Figure 6. 31: Candidate can see his applied job status

32. A candidate can see experienced agent and can explore them

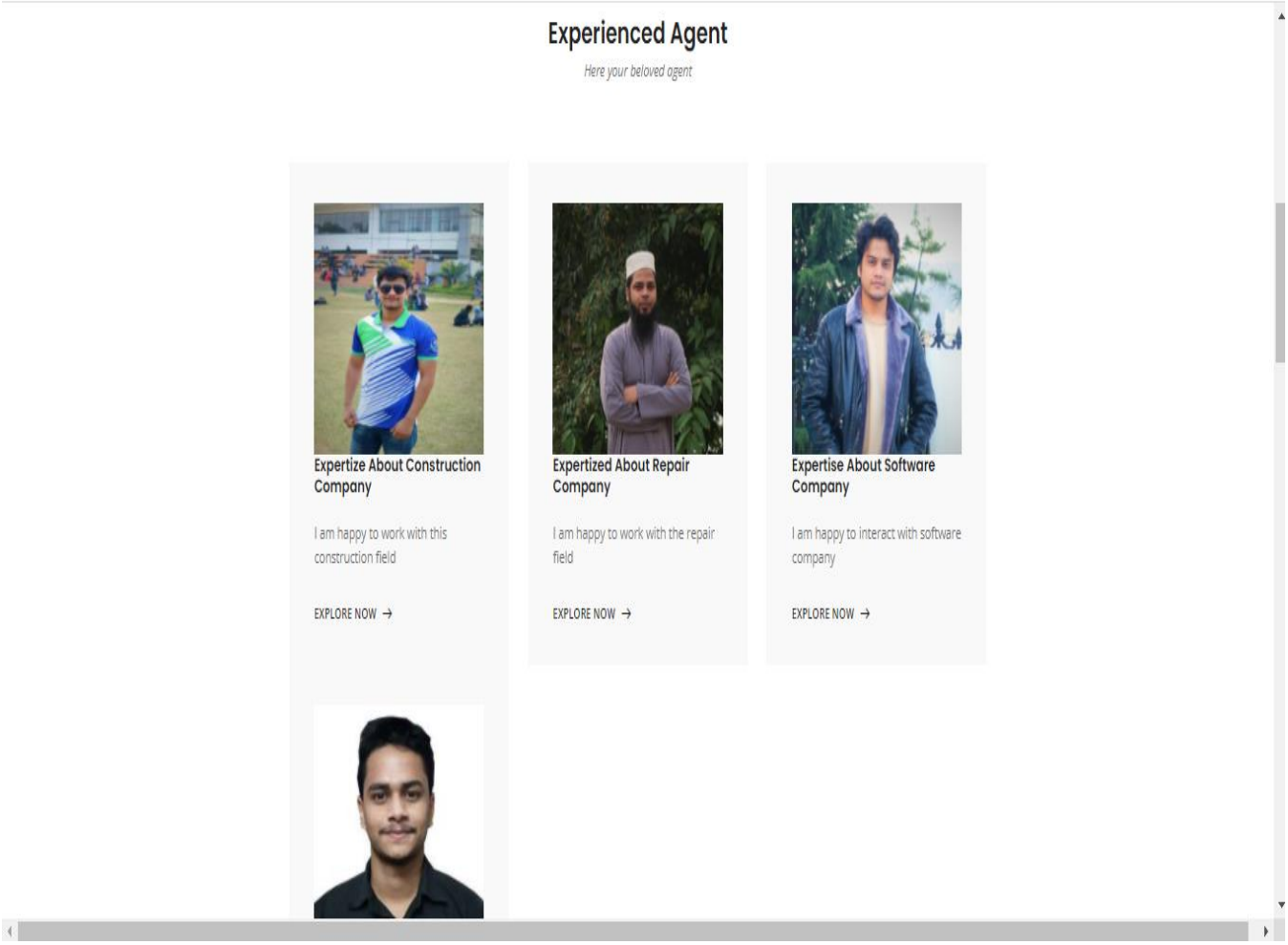


Figure 6. 32: Candidate can see the experienced agent

33. After exploring an agent, he can view the agent profile in detail as well as his services

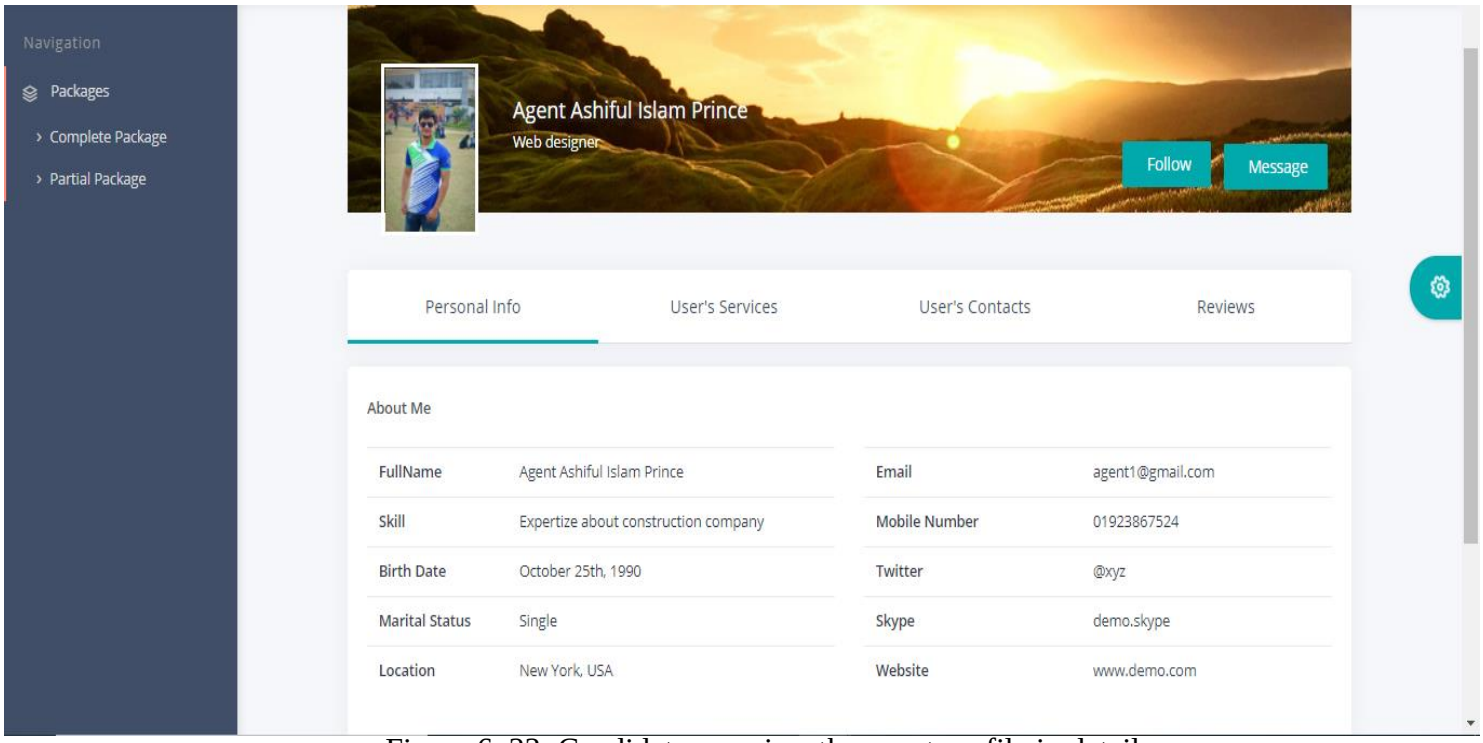


Figure 6. 33: Candidate can view the agent profile in detail

34. A candidate can see packages in detail

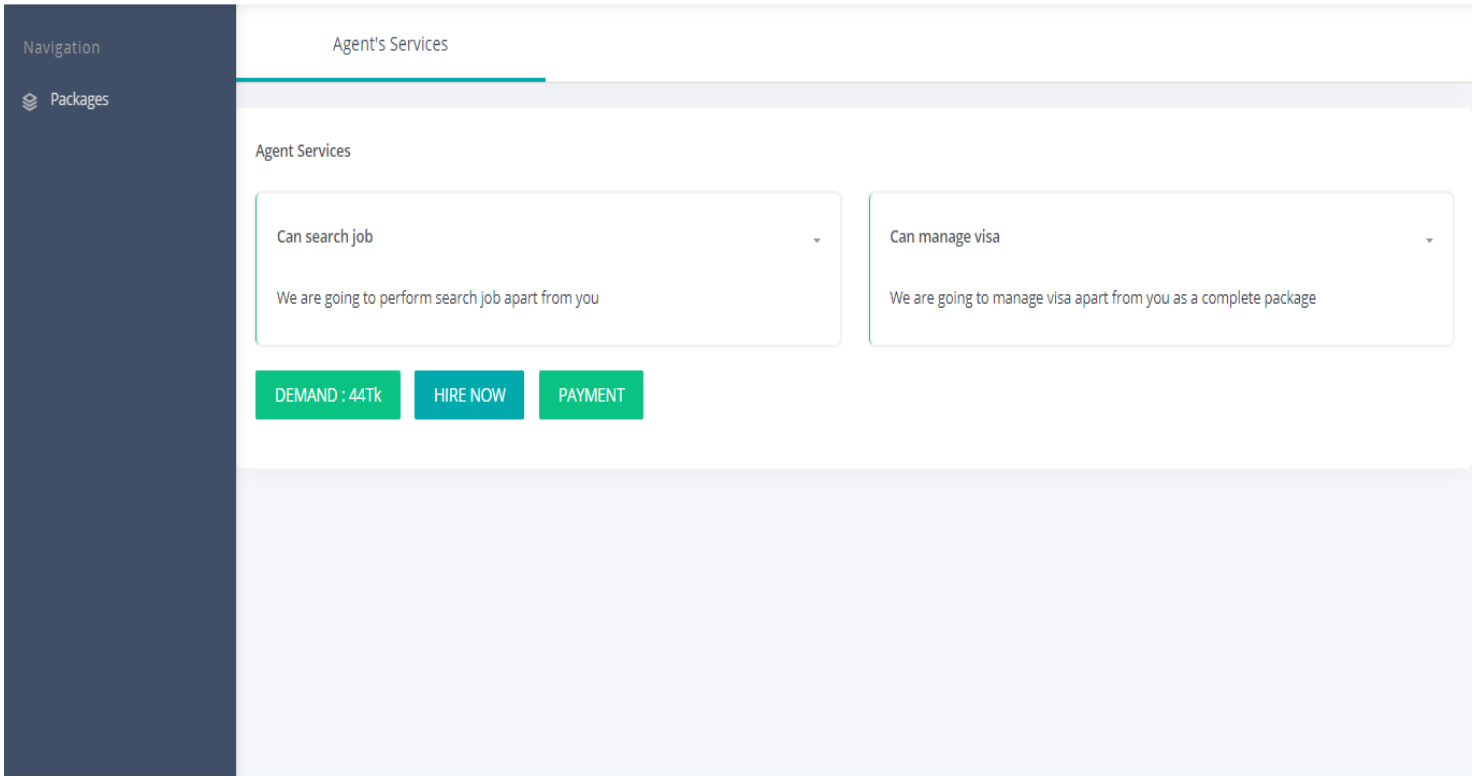


Figure 6. 34: Candidate can see package in detail

35. A candidate can make a hire request for getting the agent's services and see the status

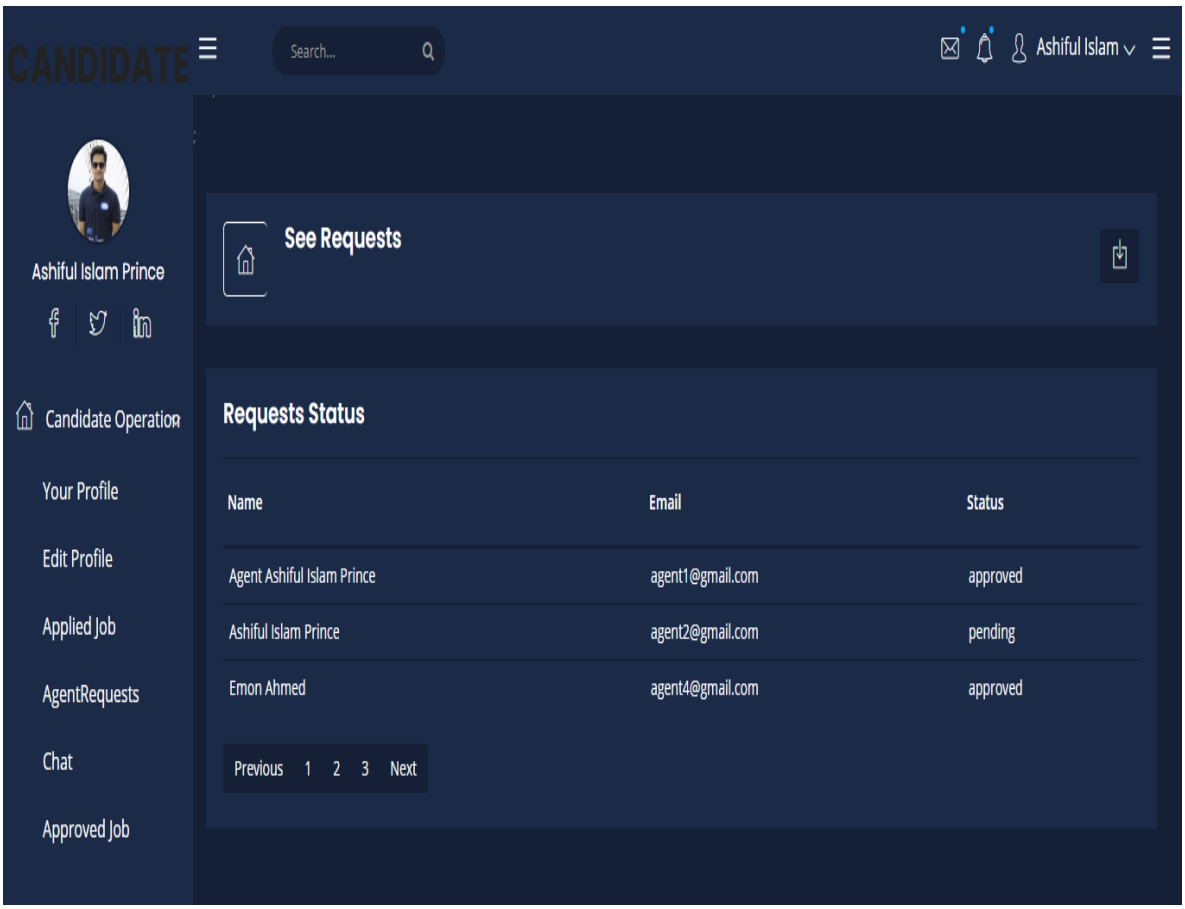


Figure 6. 35: Candidate can make a hire request

36. A candidate can make a payment after getting the request approval from the agent by giving billing information

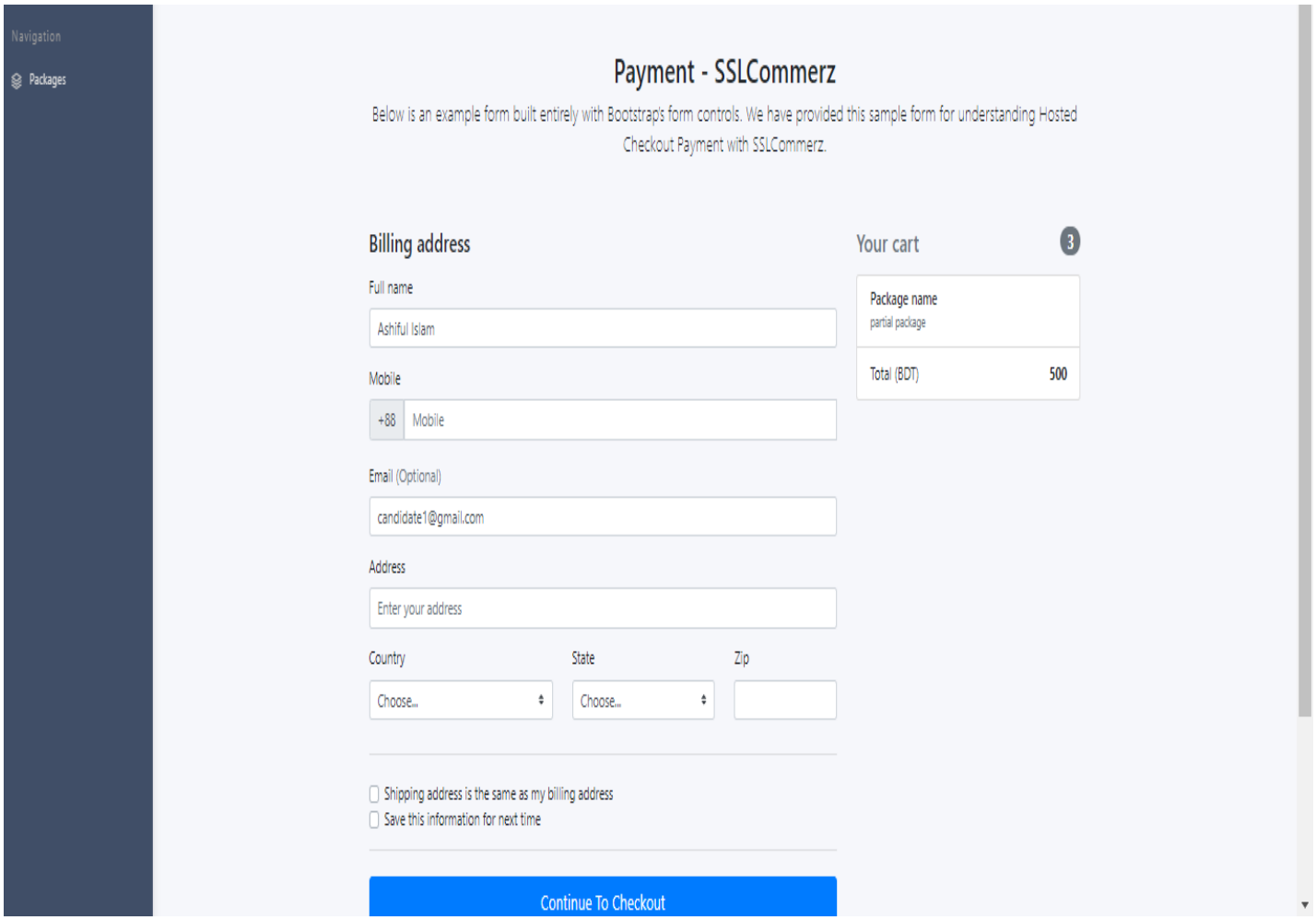


Figure 6. 36: Candidate can pay

37. A candidate can make an easy payment by choosing any payment method like bkash, rocket etc.

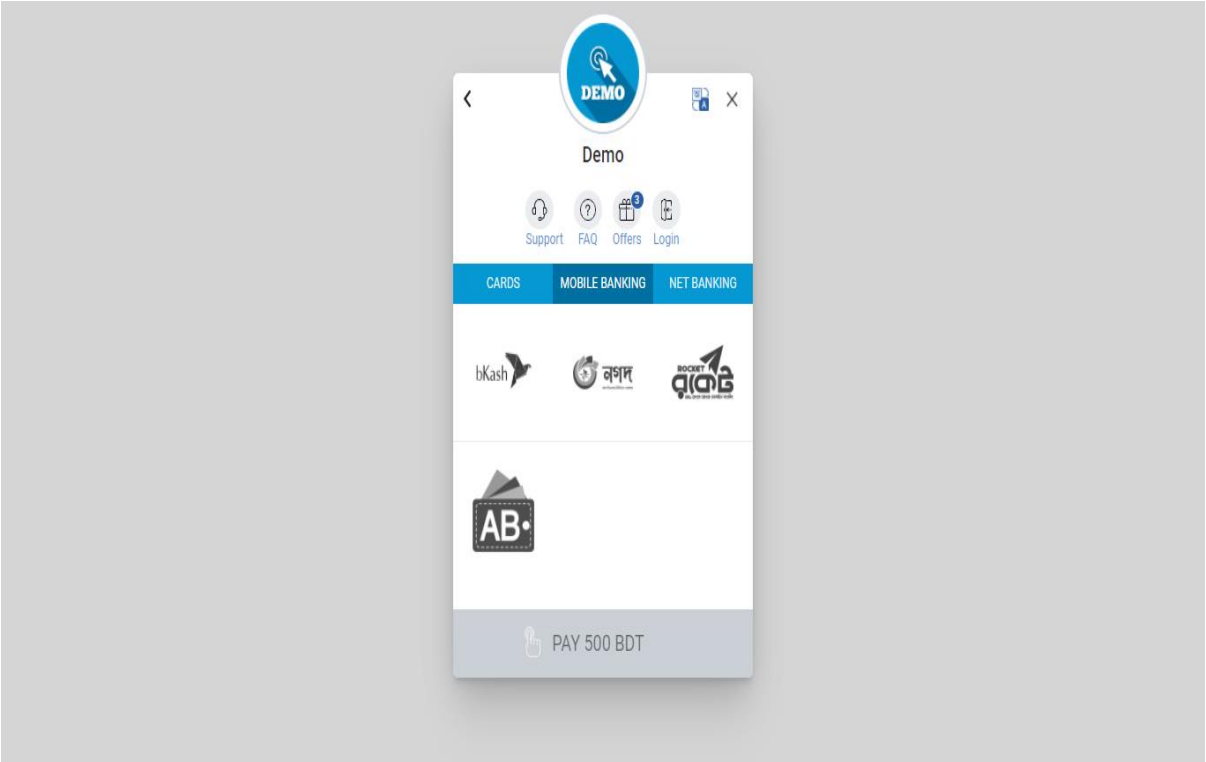


Figure 6. 37: Candidate can pay through bkash, rocket.

38. Here is the OTP page

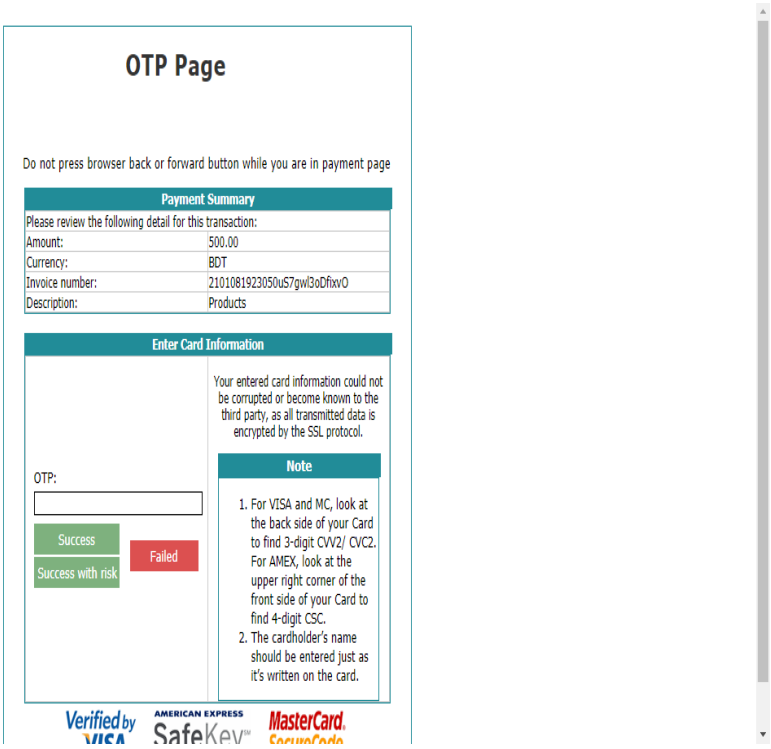


Figure 6. 38: OTP page

39. A candidate can make a communication with the agent

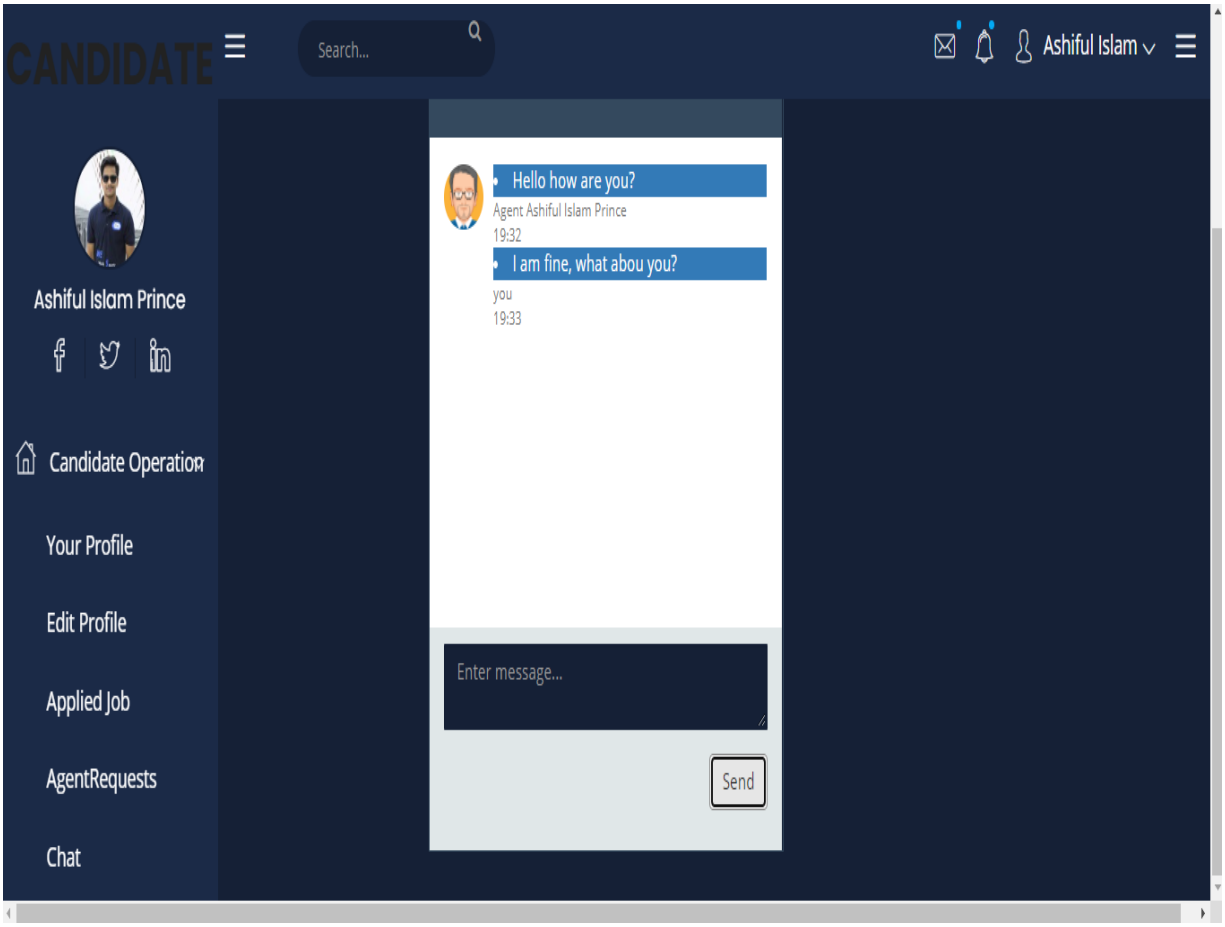


Figure 6. 39: Candidate can make communication

7. CHAPTER-07 PROJECT SUMMARY

7.1. GIT HUB LINK: https://github.com/ashifulislam/safeAgency.com

7.2. LIMITATIONS

Table 6. 1: Limitations of this systems

Performance	1) Less Performance
Speed	2)Less Speed
Features	1)User cannot Rate on this system 2)Monthly Subscription System has not been done yet. 3)Any candidate applies for the job. Because of this reason an employer could not find their desired candidate too early as possible. 4)There is no video chatting system among employer and candidate.
Ability	1)Million users can not hit on this system at a time.

7.3. FUTURE WORK PLAN

- User Rating System
- Monthly Subscription System
- Only Eligible Candidate Can Apply for the Job.
- Live Video Chat Among Employers and Candidates

APPNEDIX A

```
<?php

namespace App\Http\Controllers;

use App\AgentRequest;
use App\AgentsProfile;
use App\Candidate;
use App\CandidateRequests;
use App\Employer;
use Illuminate\Http\Request;
use Auth;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Redirect;
use RealRashid\SweetAlert\Facades\Alert;
use Symfony\Component\Console\Input\Input;

class LocalAgentController extends Controller
{
   /**
    * Create a new controller instance.
    *
    * @return void
    */
   protected $data = [];

   public function __construct()
   {
      $this->middleware('auth:localAgent');
   }
```



```
   public function approvedCandidates()
   {

      $current_agent_id=Auth::user()->id;
     $approvedCandidates=DB::table('candidate_requests')
         ->select(

'candidates.id','candidates.email','candidate_requests.agent_reg_id','candidates.firstName',
            'candidates.title','candidates.skill_name',
'candidate_requests.candidate_id','candidate_requests.status',
            'package_lists.package_type','orders.payment_status','orders.phone')


         ->join('candidates','candidates.id','=','candidate_requests.candidate_id')
         ->join('package_lists','package_lists.id','=','candidate_requests.package_type_id')
         ->join('orders','candidates.id','=','orders.candidate_id','left outer')
         ->where('candidate_requests.agent_reg_id',$current_agent_id)
         ->where('orders.agent_reg_id',$current_agent_id)
         ->where('candidate_requests.status','=','approved')
         ->orderBy('candidates.id','ASC')
         ->get();


      //get candidate id from above
      //and match to the order table then we can get the payment status of the specific
candidate
      return
view('local_agent.approvedCandidates',['approvedCandidates'=>$approvedCandidates]);
   }
```



```
public function approveCandidates(Request $request,$current_candidate_id){
    $current_agent_id=Auth::user()->id;

     if(CandidateRequests::all()
        ->where('status','=','approved')
        ->where('agent_reg_id',$current_agent_id)
        ->where('candidate_id',$current_candidate_id)
        ->count()>0
        ){
        return back()->with('error_message','Already approved');
     }
     else{
        CandidateRequests::where('candidate_id',$current_candidate_id)->
        where('agent_reg_id',$current_agent_id)
           ->update(array('status'=>$request->get('status')));

}
return back()->with('success_message','The request is approved');

 }
 public function rejectCandidates(Request $request,$current_candidate_id){
    $current_agent_id=Auth::user()->id;

    if(CandidateRequests::all()
          ->where('status','=','rejected')
          ->where('agent_reg_id',$current_agent_id)
          ->where('candidate_id',$current_candidate_id)
          ->count()>0
    ){
       return back()->with('error_message','Already rejected');
    }
```



```
        else{
            CandidateRequests::where('candidate_id',$current_candidate_id)->
            where('agent_reg_id',$current_agent_id)
                ->update(array('status'=>$request->get('status')));
        }
        return back()->with('success_message','The request is rejected');

    }

    public function seeCandidateRequests(){
        $current_agent_id=Auth::user()->id;

        $candidates=DB::table('candidates')
            -
>select('candidates.id','candidates.email','candidate_requests.agent_reg_id','candidates.first
Name',

'candidates.title','candidates.skill_name','candidate_requests.candidate_id','candidate_requ
ests.status','package_lists.package_type')
            ->join('candidate_requests','candidates.id','=','candidate_requests.candidate_id')
            ->join('package_lists','package_lists.id','=','candidate_requests.package_type_id')

            ->where('candidate_requests.agent_reg_id',$current_agent_id)

            ->get();

                if(session('success_message')){
                Alert::success('Success',session('success_message'))->autoClose(3000);
            }
        else if(session('error_message')){
            Alert::error('Error',session('error_message'))->autoClose(3000);
        }
```



```
        return view('local_agent.candidateRequestLists',['candidates'=>$candidates]);

    }
    public function seeEmployers(){
        $employer=Employer::all();

        if(session('success_message')){

            Alert::success('Success', session('success_message'))->autoClose(3000);

        }
        else if(session('error_message')){
            Alert::error('Error', session('error_message'))->autoClose(3000);



        }

        return view('local_agent.seeEmployerList')->with(['employer'=>$employer]);
}
    public function seeRequests(){

        $current_agent_id=Auth::user()->id;

        $data['employers']=  DB::table('employers')

            ->select('employers.firstName','employers.email','employers.companyName','employers.companyCountry','agent_requests.status','agent_requests.emp_id')

            ->join('agent_requests', 'agent_requests.emp_id', '=', 'employers.id')
```



```
        ->where(['agent_requests.agent_reg_id'=>$current_agent_id])
         ->get();
   return view('local_agent.requestLists',$data);
  }
  public function sendRequestToEmployer(Request $request,$emp_id){
  $id = Auth::user()->id;
    if( AgentRequest::where('agent_reg_id', '=',$id)
         ->where('emp_id','=',$emp_id)
         ->count()>0
       ){
         return redirect()->back()->with('error_message','Request already sent');
}
      else{
         //Requests does not exists
         $agentReq = new AgentRequest();
         $agentReq->status  = $request->get('status');
         $agentReq->agent_reg_id = $id;
         $agentReq->emp_id=$emp_id;

         $agentReq->save();
        return redirect()->back()->with('success_message','The request is sent');

      }
  }
  /**
   * Show the application dashboard.
   *
   * @return \Illuminate\Contracts\Support\Renderable
   */
```



```
    public function index()
    {
        $current_agent_id=Auth::user()->id;
        $data['agents']=AgentsProfile::select('photo')-
>where('agent_reg_id',$current_agent_id)->get();
        return view('home',$data);
    }

}
<?php

namespace App\Http\Controllers;

use App\AgentRequest;
use App\Events\ChatEvent;
use App\JobApplication;
use App\JobPost;
use Illuminate\Http\Request;
use App\Employer;
use App\JobCategory;
use Illuminate\Queue\Jobs\Job;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Redirect;
use RealRashid\SweetAlert\Facades\Alert;
use Sentinel;
use Auth;

class EmployerController extends Controller
{

    public function chat(){
```



```
        return view('employer.liveChat');

    }


    public function send(Request $request){
        // return $request->all();
        $user = Auth::user()->firstName;



        event(new ChatEvent($request->message,$user));

    }
    public function approveRequest(Request $request,$id){


        $current_employer_id=Auth::user()->id;
         AgentRequest::where('agent_reg_id', $id)->
           where('emp_id',$current_employer_id)
           ->update(array('status' => $request->get('status')));

        return redirect()->back()->with('success_message','The request is approved');

    }
   public function rejectRequest(Request $request,$id){


        $current_employer_id=Auth::user()->id;
         AgentRequest::where('agent_reg_id', $id)->
```



```
            where('emp_id',$current_employer_id)
            ->update(array('status' => $request->get('status')));

        return redirect()->back()->with('success_message','The request is rejected');

    }


    public function createJobCategory(Request $request){

        if(session('success_message')){
            Alert::success('Success', session('success_message'))->autoClose(3000);

        }


        return view('employer/jobCategory')->with('email',$request->session()->get('user'));
    }
    public function createJobPost(Request $request){
        return view('employer/employerJobPost');
    }
    public function addJobCategory(Request $request){

//          $this->validate($request,[
//           'categoryName'=>'required',
//           'categoryType'=>'required',
//             'myCheck'=>'required'
//       ]);
        $rules=[
            'categoryName'=>['required'],
            'categoryType'=>['required'],
```



```
        'myCheck'=>['required'],

    ];
    $request->validate($rules);


    $emp_id=Auth::user()->id;
    $addJobCategory=new JobCategory();
    $addJobCategory->categoryName=$request->input('categoryName');
    $addJobCategory->categoryType=$request->input('categoryType');
    $addJobCategory->employerId=$emp_id;
    $addJobCategory->save();
    return redirect('/jobCategory')->with('success_message','The category is added');
    }

public function showEmployerList()
{
    //
    $user_id = Auth::user()->id;
    $data['data'] = DB::table('employers')->where('id' ,'=', $user_id)->get();
    if(count ($data)>0){
        return view('employer/show')->with('showEmployer',$data['data']);       }

}


public function updateEmployer($id){
     $addJobCategory=Employer::find($id);
```



```
        return
view('employer/updateEmployerProfile',['updateEmployerProfile'=>$addJobCategory]);


    }
    public function editEmployer(Request $request,$id){
        $this->validate($request,[
            'FirstName'=>'required',
            'LastName'=>'required',
            'CompanyName'=>'required',
            'CompanyDetails'=>'required',
            'CompanyCountry'=>'required',
            'CompanyState'=>'required',
            'CompanyZipCode'=>'required',
            'myCheck'=>'required'


        ]);



        $update=array('firstName'=>$request->input('FirstName'),'lastName'=>$request-
>input('LastName'),'companyName'=>$request->input('CompanyName'),
                    'companyDetails'=>$request->input('CompanyDetails'),
                    'companyCountry'=>$request->input('CompanyCountry'),
                    'companyState'=>$request->input('CompanyState'),
                    'companyZipCode'=>$request->input('CompanyZipCode'));
        $updateEmployer=Employer::where('id',$id);
        $updateEmployer->update($update);
        return redirect('/show')->with('success','Updated Successfully');


    }
    public function deleteEmployer($id){


        $delete=Employer::where('id',$id);
```



```
    $delete->delete();
    return redirect('/show')->with('DeleteSuccess','Deleted Successfully');
}


public function showSingleInfo($id){
    $showEmployer=Employer::find($id);
   return view('employer/viewSingleInfo',['viewSingleInfo'=>$showEmployer]);
}

public function __construct()
{
    $this->middleware('auth:employer');
}
/**
 * show dashboard.
 *
 * @return \Illuminate\Http\Response
 */
public function index()
{
    return view('employer/admin_home');
}
public function showPendingAgentRequest(){

    $current_employer_id=Auth::user()->id;
    $data = [];

        $data['agents']=  DB::table('local_agents')
```



```
            -
>select('agents_profiles.bio','agents_profiles.agent_reg_id','agents_profiles.photo','agents_
profiles.about','agents_profiles.interest','agents_profiles.skill','local_agents.name','local_age
nts.email','agents_profiles.age','local_agents.phone','agent_requests.status')


            ->join('agents_profiles','agents_profiles.agent_reg_id','=','local_agents.id')
            ->join('agent_requests', 'agent_requests.agent_reg_id', '=', 'local_agents.id')
            ->where(['agent_requests.emp_id'=>$current_employer_id])
            ->get();


      if(session('success_message')){
         Alert::success('Success', session('success_message'))->autoClose(3000);


      }
      return view('employer.pendingAgentRequest',$data);


   }


   public function showPendingJobApplication(){
      //current employer and emp_id of the specific job post then get the data
      //specific job post id and employer of this specific job post id
//       $data['pendingPosts'] = JobApplication::where('status','pending')->with('candidate')-
>orderBy('id','DESC')->get();
        $current_employer_id=Auth::user()->id;
         $data['pendingPosts']=DB::table('job_applications')
         -
>select('job_applications.id','candidates.firstName','candidates.email','candidates.skill_name
'


,'candidates.softSkills','job_applications.interest','job_applications.salary','job_posts.employ
erId')
         ->join('candidates','candidates.id','=','job_applications.candidateId')
         ->join('job_posts','job_posts.id','job_applications.jobPostId')
         ->where('job_applications.status','=','pending')
```



```
            ->where('job_posts.employerId',$current_employer_id)
            ->get();
            if(session('success_message')){
                Alert::success('Success',session('success_message'))->autoClose(3000);
            }

        return view('employer.pendingJobApplication',$data);
    }
    public function updatePendingJobApplicationStatus(Request $request, $id){
        // dd($request->all());
        $jobApplication = JobApplication::findOrFail($id);
        $jobApplication->status = $request->input('status');
        $jobApplication->save();

        return redirect()->back()->with('success_message','Application is approved
successfully');
    }
}
<?php

namespace App\Http\Controllers;

use App\AgentsProfile;
use App\Events\ChatEvent;
use App\JobApplication;
use App\JobPost;
use Illuminate\Http\Request;
use App\Candidate;
use App\JobCategory;
use Illuminate\Support\Facades\DB;
use RealRashid\SweetAlert\Facades\Alert;
```



```
use Sentinel;
use Auth;

class CandidateController extends Controller
{
   public function __construct()
   {
      $this->middleware('auth:candidate')-
>except('showAllAgent','showPackageList','showPartialPackageList');
   }
   public function chat()
   {

      return view('candidate.liveChat');

   }

   public function send(Request $request)
   {

      $user = Auth::user()->firstName;
      event(new ChatEvent($request->message,$user));

   }

   public function showPartialPackageList($agent_id)
   {

      if(session('success_message'))
      {
```



```
        Alert::success('Success',session('success_message'))->autoClose(4000);


    }
    else if(session('error_message'))
    {
        Alert::error('Error',session('error_message'))->autoClose(4000);
    }
    else if(session('not_approved'))
    {


        Alert::error('Error',session('not_approved'))->autoClose(4000);


    }


    $services=DB::table('service_types')


        -
>select('service_types.service_title','service_types.service_type','manage_services.service_description','manage_services.demand','manage_services.agent_reg_id','package_lists.package_type')
        ->join('manage_services','service_types.id','=','manage_services.service_type_id')
        ->join('package_lists','package_lists.id','=','manage_services.package_type_id')
        ->where(['service_types.agent_reg_id'=>$agent_id])
        ->where(['package_lists.package_type'=>'partial package'])
        ->get();


    //Getting demands
    $demand = DB::table('package_lists')


        ->select('demand')
        ->join('manage_services','package_lists.id','=','manage_services.package_type_id')
        ->where(['manage_services.agent_reg_id'=>$agent_id])
```



```
        ->where(['package_lists.package_type'=>'partial package'])
        ->get();

    $total_amount=0;
    //Calculate demands
    foreach ($demand as $singleDemand)
    {
        $total_amount=$total_amount+$singleDemand->demand;
    }

    $packageTypeId=DB::table('package_lists')

        ->select('manage_services.package_type_id','package_lists.package_type')
        ->join('manage_services','package_lists.id','=','manage_services.package_type_id')
        ->where('manage_services.agent_reg_id',$agent_id)
        ->where('package_lists.package_type','=','partial package')
        ->first();
    return view('candidate.partialPackageList',['services' =>
$services],['demands'=>$total_amount])->with('agent_id', $agent_id)-
>with(['packageTypeId'=>$packageTypeId]);
  }

  public function showPackageList($agent_id)
  {
    if(session('success_message'))
    {

        Alert::success('Success',session('success_message'))->autoClose(4000);

    }
    else if(session('error_message'))
```



```
{
   Alert::error('Error',session('error_message'))->autoClose(4000);
}
else if(session('not_approved')){

   Alert::error('Error',session('not_approved'))->autoClose(4000);

}

//Getting services based on package
$services=DB::table('service_types')

   -
>select('service_types.service_title','service_types.service_type','manage_services.service_description','manage_services.demand','manage_services.agent_reg_id','package_lists.package_type')
   ->join('manage_services','service_types.id','=','manage_services.service_type_id')
   ->join('package_lists','package_lists.id','=','manage_services.package_type_id')
   ->where(['service_types.agent_reg_id'=>$agent_id])
   ->where(['package_lists.package_type'=>'complete package'])
   ->get();

//getting the package_type_id
$packageTypeId=DB::table('package_lists')

   ->select('manage_services.package_type_id','package_lists.package_type')
   ->join('manage_services','package_lists.id','=','manage_services.package_type_id')
   ->where('manage_services.agent_reg_id',$agent_id)
   ->where('package_lists.package_type','=','complete package')
   ->first();
```



```
      //Getting demands
       $demand = DB::table('package_lists')

          ->select('demand')
          ->join('manage_services','package_lists.id','=','manage_services.package_type_id')
          ->where(['manage_services.agent_reg_id'=>$agent_id])
          ->where(['package_lists.package_type'=>'complete package'])
          ->get();

       $total_amount=0;
       //Calculate demands
       foreach ($demand as $singleDemand)
       {
          $total_amount=$total_amount+$singleDemand->demand;
       }



       return view('candidate.packageList',['services' =>
$services],['demands'=>$total_amount])
          ->with('agent_id', $agent_id)
          ->with(['packageTypeId'=>$packageTypeId]);


    }

    public function showAllAgent($agent_id)
    {



       $data['agents']=  DB::table('local_agents')

          -
>select('agents_profiles.bio','agents_profiles.agent_reg_id','agents_profiles.photo','agents_
```



```
profiles.about','agents_profiles.interest','agents_profiles.skill','local_agents.name','local_agents.email','agents_profiles.age','local_agents.phone')

        ->join('agents_profiles','agents_profiles.agent_reg_id','=','local_agents.id')
        ->where(['local_agents.id'=>$agent_id])

        ->get();



    return view('candidate.showAllAgentProfile',$data);
  }

  public function candidateHome()
  {
    return view('candidate/candidateHome');
  }


  public function showCandidate()
  {
    //
    $user_id = Auth::user()->id;
    $data['data'] = DB::table('candidates')->where('id', '=', $user_id)->get();
    if (count($data) > 0) {


        return view('candidate/viewYourProfile')->with('showCandidate', $data['data']);
    }
  }
```



```
    public function showCandidateForUpdate()
    {
        //
        $user_id = Auth::user()->id;
        $data['data'] = DB::table('candidates')->where('id', '=', $user_id)->get();
        if (count($data) > 0) {


            return view('candidate/show')->with('showCandidate', $data['data']);
        }
    }

    public function updateCandidate($id)
    {

        $returnValue=Candidate::find($id);
        return
view('candidate/updateCandidateProfile',['updateCandidateProfile'=>$returnValue]);


    }

    public function editCandidate(Request $request,$id)
    {

        $this->validate($request,[
            'firstName'=>'required',
            'lastName'=>'required',
            'email'=>'required',
            'institute'=>'required',
            'org'=>'required',
            'address'=>'required',
```



```
            'softSkills'=>'required',
            'skill_name'=>'required',
            'myCheck'=>'required'

        ]);

        $update=array('firstName'=>$request->input('firstName'),'lastName'=>$request->input('lastName'),'email'=>$request->input('email'),
            'degree'=>$request->input('degreeType'),
            'institute'=>$request->input('institute'),
            'title'=>$request->input('jobTitle'),
            'org'=>$request->input('org'),
            'address'=>$request->input('address'),
            'softSkills'=>$request->input('softSkills'),
            'skill_name'=>$request->input('softSkills')
            );
        $updateCandidate=Candidate::where('id',$id);
        $updateCandidate->update($update);
        return redirect('/showForUpdate')->with('success','Updated Successfully');

    }

    public function deleteCandidate($id)
    {
        $delete=Candidate::where('id',$id);
        $delete->delete();
        return redirect('/showForUpdate')->with('DeleteSuccess','Deleted Successfully');
    }

    public function showApplicationForm()
    {
```



```
        return view('candidate/jobApplicationForm');
    }

    public function jobApplication($id)
    {
        $jobPosts['jobPosts']=JobPost::with('employer')->find($id);
        if(session('error_message'))
        {

            Alert::error('error', session('error_message'))->autoClose(3000);

        }
        return view('candidate.jobApplicationForm',$jobPosts);
    }



    public function jobConfirmation()
    {

        $user_id = Auth::user()->id;

        $jobPosts['appliedJobs']=JobApplication::where('candidateId',$user_id)-
>with(['jobPost'=>function($query){
            return $query->with(['employer','jobCategory']);}])->orderBy('id','DESC')->get();

        if(session('success_message'))
        {
            Alert::success('Success', session('success_message'))->autoClose(3000);

        }
```



```
        return view('candidate.seeAppliedJobs',$jobPosts);
    }


}
```


REFERENCES

Athuraliya, A. (2019). The Quick Guide to Creating a Proper Product Roadmap. Retrieved from creately: https://creately.com/blog/business/how-to-create-a-product-roadmap/

Bartlett, J. (2020, 8 14). How to Write Test Cases for Software (with a sample). Retrieved from TestLodge: https://blog.testlodge.com/how-to-write-test-cases-for-software-with-sample/

Khan, B. (2017, 7 4). Why and how to use project blogs for your projects? Retrieved from Planisware Orchestra: https://www.orchestra-ppm.com/en/2017/07/04/why-and-how-to-use-project-blogs-for-your-projects/

Kinsta. (2018). Job Board. Retrieved from Colorib: https://colorlib.com/wp/cat/job-board/

Krishna. (2020). TEST PLAN: What is, How to Create (with Example). Retrieved from Guru99: https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html

Maheshwary, S. (2019, 4 17). Laravel Multiple Guards Authentication: Setup and Login. Retrieved from Medium: https://medium.com/@sagarmaheshwary31/laravel-multiple-guards-authentication-setup-and-login-2761564da986

Mark. (2010, 6 3). Systems Analysis and Design. Retrieved from Mark's IT Blog.

Rivera, M. (2020). 7 Steps to Writing the Perfect Project Proposal. Retrieved from the Blueprint: https://www.fool.com/the-blueprint/project-proposal/

Roozemond, B. (2020, 10 31). How draw.io diagrams help you tackle the challenges of Covid-19. Retrieved from draw.io: https://drawio-app.com/how-drawio-diagrams-help-you-tackle-the-challenges-of-covid-19/

Tryfanava, D. (2020, 7 28). Functional vs Non-functional Requirements: Examples and Types. Retrieved from vironIT: https://vironit.com/functional-vs-non-functional-requirements/

