IBM PROJECT DOCUMENT

CUSTOMER CARE REGISTRY

Project Name	:	Customer Care Registry
Project Domain	:	Cloud Application Development
College	:	Sri Krishna College of Engineering and Technology
Team ID	:	**PNT2022TMID02958**
Team Size	:	4
Team Members	:	A. Naveen Kumar C B. Pradeep B C. Navin S D. Mohamed Asshar M
Team Mentor	:	Mr. Anandkumar V
Team Evaluator	:	Dr. Keerthika T

TABLE OF CONTENTS

1. INTRODUCTION:

1.1. PROJECT OVERVIEW:

Customer care describes how people are treated when they interact with a brand. This includes all experiences with the company and its employees before, during, and after a purchase. Customer care is an important aspect of customer service because it fosters an emotional connection with the brand's community. Customer care isn't measured in the same way as customer loyalty or success. That's because things like loyalty and success are a by-product of caring for your customers. It's impossible to build a trustworthy, emotional connection with your customer base if you're too focused on measuring it. Customer care goes a step further by ignoring the metrics and instead fully investing in your customers' goals and needs.

1.2. PURPOSE:

- A customer experience is an interaction between an organization and a customer as perceived through a customer’s conscious and subconscious mind. Customer experience with product or service results in customer satisfaction which results in loyalty.
- Struggles with making an impulse decision and preferring trendy and adoptable price for products act upon especially in this modern era. The aim of this application is to track the current trends and suggesting low-cost price with best quality through customer complaints. Also, to identify the sorting of simple and best way suggestion for various queries.

2. LITERATURE SURVEY:

2.1. EXISTING PROBLEM:

- In Customer Experience Management in Online Retailing, they described the customer care concept with the help of CEM. Customer experience management (CEM) is the collection of processes a company uses to track, oversee and organize every interaction between a customer and the organization throughout the customer lifecycle. The goal of CEM is to optimize interactions from the customer's point of view and, as a result, promote customer loyalty. Customer experience management (CEM) is defined as “the discipline of managing and treating customer relationships as assets with the goal of transforming satisfied customers into loyal customers, and loyal customers into advocates of your brand.” A customer experience is an interaction between an organization and a customer as perceived through a customer’s conscious and subconscious mind. It is a blend of an organization’s rational performance, the senses stimulated and the emotions evoked and intuitively measured against customer expectations across all moments of contact.

- In Customer Satisfaction towards Online Shopping, having access to online shopping has truly revolutionized and influenced our society as a whole. This use of technology has opened new doors and opportunities that enable for a more convenient lifestyle today. Variety, quick service and reduced prices were three significant ways in which online shopping influenced people from all over the world. However, this concept of online shopping led to the possibilities of fraud and privacy conflicts. Unfortunately, it has shown that it is possible for criminals to manipulate the system and access personal information. Luckily, today with the latest features of technology, measures are being taken in order to stop hackers and criminals from inappropriately accessing private databases. Through privacy and security policies, website designers are doing their best to put an end to this unethical practice.

2.2. REFERENCES:

https://www.researchgate.net/publication/274510494_Customer_Experience_Management_in_Online_Retailing-_A_Literature_Review

https://www.researchgate.net/publication/329026968_A_Study_on_customer_Satisfaction_towards_Online_Shopping

2.3. PROBLEM STATEMENT DEFINITION:

I am a regular customer in famous e-commerce websites like Amazon, Flipkart. I order regularly. The problem I have is that in most times, I don’t have any reliable sources to clear my doubts in some of the products I buy. There are reviews and customer ratings in those websites, but somehow, I don’t feel they are authentic and real. It would make my world if those replies were from a real expert, and I could clarify all my doubts in a single platform. Of course, I would need instant replies from a real expert who knows about the products I am asking for

3. IDEATION AND PROPOSED SOLUTION:

3.1. EMPATHY MAP CANVAS:

EMPATHY:

An empathy map is a powerful visual tool that captures a product team's knowledge of a certain type of user's thoughts, feelings, and actions.

It's used to quickly and easily express user needs, especially to stakeholders who may not be involved in the research and design process, such as executives or clients.

USES OF EMPATHY MAP:

Product teams and marketing teams alike can use and benefit from empathy mapping exercises. Any time you want to create a shared understanding of a certain type of user, this exercise can be helpful.

Empathy mapping can be done at the beginning of the UX design process, or it can be used further along in the design process as new products or features are being tested. It can also be used in tandem with other UX-focused exercises like customer journey mapping and story mapping.

SECTIONS OF EMPATHY MAP:

- Says
- Thinks
- Does
- Feels

SAYS:

The first quadrant to fill out is what the user says while using your product. This information should be taken directly from your research if possible. List what past and current users have expressed about your product and their experience with it.

THINKS:

The second quadrant is about what the user is thinking throughout their experience with your product. There may be some overlap here with the “says” quadrant, but the purpose of “thinks” as a separate quadrant is for you to consider what users might be thinking that they aren’t saying, and to consider why they might not be saying it.

This step may require some brainstorming from you and your team, but will be incredibly helpful in shaping the direction of your project.

DOES:

The third quadrant describes what actions users take while using the product. This information can come from user testing or interviews. Pay special attention to where they seem to get confused or behave in a way that you did not intend or predict.

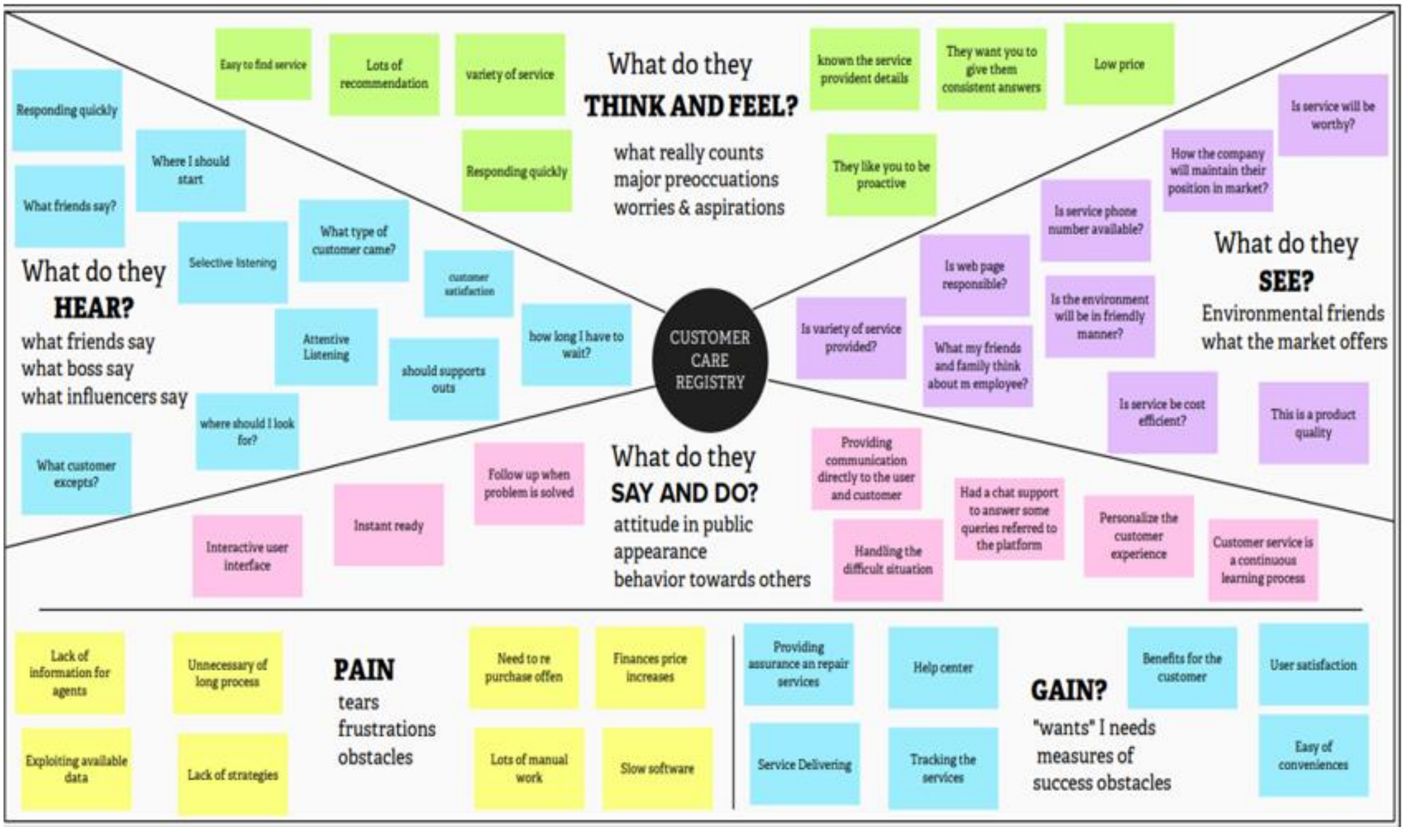
It can also include actions they take that are related to their customer journey, such as comparing other products or making purchasing decisions.

FEELS:

The final step in the empathy mapping process is to consider how the user feels throughout their experience and how they might feel coming away from the experience. This should be based on data, but also requires a little brainstorming.

When completing this quadrant, focus on the user’s emotional state, both the what and the why. For example, if something was loading slowly or they weren’t able to find something they were looking for, they might be feeling frustrated or confused. If they found the product helpful, they might be curious or excited.

CUSTOMER CARE REGISTRY EMPATHY MAP:


Easy to find service
Lots of recommendation
variety of service
What do they THINK AND FEEL?
what really counts
major preoccuations
worries & aspirations
known the service provident details
They want you to give them consistent answers
Low price
Responding quickly
They like you to be proactive
Responding quickly
Where I should start
What friends say?
Selective listening
What type of customer came?
customer satisfaction
What do they HEAR?
what friends say
what boss say
what influencers say
Attentive Listening
should supports outs
how long I have to wait?
where should I look for?
What customer excepts?
CUSTOMER CARE REGISTRY
Is service will be worthy?
How the company will maintain their position in market?
Is service phone number available?
Is web page responsible?
Is the environment will be in friendly manner?
What do they SEE?
Environmental friends
what the market offers
Is variety of service provided?
What my friends and family think about m employee?
Is service be cost efficient?
This is a product quality
Follow up when problem is solved
Instant ready
Interactive user interface
What do they SAY AND DO?
attitude in public
appearance
behavior towards others
Providing communication directly to the user and customer
Handling the difficult situation
Had a chat support to answer some queries referred to the platform
Personalize the customer experience
Customer service is a continuous learning process
Lack of information for agents
Unnecessary of long process
PAIN
tears
frustrations
obstacles
Need to re purchase offen
Finances price increases
Exploiting available data
Lack of strategies
Lots of manual work
Slow software
Providing assurance an repair services
Help center
GAIN?
"wants" I needs
measures of
success obstacles
Benefits for the customer
User satisfaction
Service Delivering
Tracking the services
Easy of conveniences

3.2. IDEATION AND BRAINSTORMING:

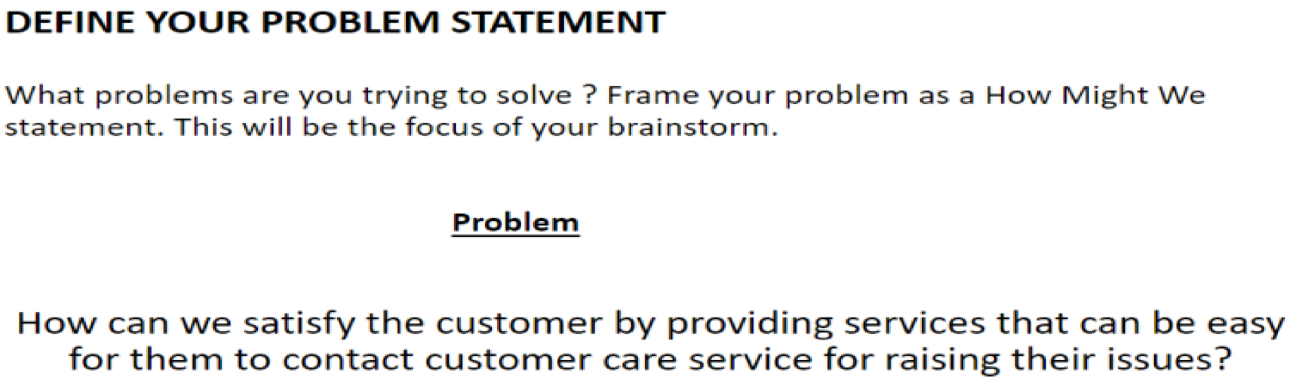
BRAINSTORMMING: Brainstorming is usually conducted by getting a group of people together to come up with either general new ideas or ideas for solving a specific problem or dealing with a specific situation.

IDEATION: Ideation is innovative thinking, typically aimed at solving a problem or providing a more efficient means of doing or accomplishing something. It encompasses thinking up new ideas, developing existing ideas and figuring out means or methods for putting new ideas into practice. Ideation is often closely related to the practice of brainstorming.

Ideation is commonly more thought of as being an individual pursuit whereas Brainstorming is almost always a group activity.

BRAINSTORM AND IDEATION:

1. TEAM GATHERING, COLLABORATION AND SELECT THE PROBLEM STATEMENT:



DEFINE YOUR PROBLEM STATEMENT

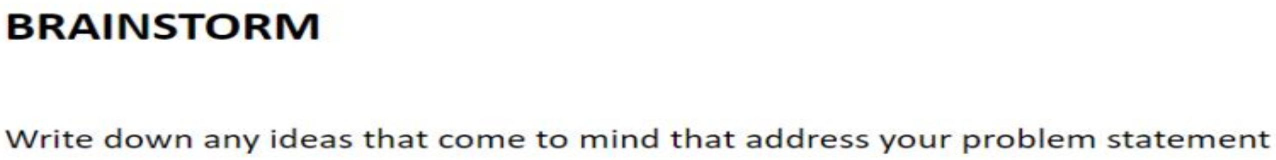
What problems are you trying to solve ? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

Problem

How can we satisfy the customer by providing services that can be easy for them to contact customer care service for raising their issues?

2. BRAINSTORM, IDEA LISTING AND GROUPING:

BRAINSTORM:



BRAINSTORM

Write down any ideas that come to mind that address your problem statement

MOHAMED ASHHAR A

LOGIN	CHATBOX
SIGNUP	VIEW PROFILE

PRADEEP B

CHANGE PASSWORD	CREATE TICKET
VIEW TICKETS	TICKET STATUS

NAVIN S

LOGOUT	CLOSE TICKET
FEEDBACK	ASSIGN AGENT

NAVEEN KUMAR C

ENCODE PASSWORD	DECODE PASSWORD
ASSIGN TICKET	ABOUT

GROUPING IDEAS:

LOGIN	CHATBOX
SIGNUP	VIEW PROFILE
CHANGE PASSWORD	CREATE TICKET
VIEW TICKETS	TICKET STATUS
LOGOUT	CLOSE TICKET
FEEDBACK	ASSIGN AGENT
ENCODE PASSWORD	DECODE PASSWORD
ASSIGN TICKET	ABOUT

3.3. PROPOSED SOLUTION:

S.NO	PARAMETER	DESCRIPTION
1	Problem Statement	I am a regular customer in famous e- commerce websites like Amazon, Flipkart. I order regularly. The problem I have is that in most times, I don't have any reliable sources to

		clear my doubts in some of the products I buy. There are reviews and customer ratings in those websites, but somehow, I don’t feel they are authentic and real. It would make my world if those replies are from a real expert and I could clarify all my doubts in a single platform. Of course, I would need instant replies from a real expert who knows about the products I am asking for.
2	Idea / Solution description	Creating a Customer Care Registry, where the customers can raise their queries in form of tickets. An agent will be assigned to them for replying/clarifying their issues.
3	Novelty / Uniqueness	The agents are experts in the product domain and they will communicate well with the customers
4	Social Impact / Customer Satisfaction	Customers will be satisfied with the instant and valid replies. Also, it creates a doubtless society, that boosts sales.
5	Business Model (Revenue Model)	Customers can be charged a minimal amount based on the number of queries (tickets) they can rise in a said period of time.
6	Scalability of the Solution	This idea is so much use to the customers that the latter may refer this registry to their friends and colleagues at work. Naturally, the user base grows so does the number of queries answered. May be in the future, may be a cross-platform mobile application may be developed, making this customer care registry much more accessible to the users.

3.4. PROBLEM FIT SOLUTION:

The Problem-Solution Fit simply means that you have found a problem with your customer and that the solution you have realized for it actually solves the

customer’s problem.

1. CUSTOMER SEGMENTS:

Customer must have to give the detailed information about the problems

they faced and how the problems have been arrived. And additionally, they can ask any new feature which they wanted with clear information.

2. PROBLEMS:

For each customer, there is an agent to be assigned so that customer can tell them queries to them so that the agent is assigned to customer will provide solution for the problem.

3. TRIGGERS:

The agent will give notifications to the customer about the level of completion of their problem in customer care.

4. EMOTIONS:

Before we don’t know how to get the solution for the problem, after we can solve the problem from online so that an agent is assigned to solve the problem.

5. AVAILABLE SOLUTIONS:

During this application, the customer can sort listed related to the problem. The agent assigned to customer can solve the problems in different ways of approach.

6. CUSTOMER CONSTRAINTS:

The application can be user friendly so that the customer can communicate with the agent through private chat, emails and calls.

7. BEHAVIOUR:

When customer tell the problem through ticket, an agent is assigned to them. With the help of the agent, the customer can get solution to their queries and notifications.

8. CHANNELS OF BEHAVIOUR:

The agent can give the good and better solution and also solve problems in different ways.

9. PROBLEM ROOT CAUSE:

Customer wants to solve the problem in quick so that agent can give more ways and take minimum time to solve the problem.

10. OUR SOLUTION:

The solution is to give the ticket to the customer to say their problem, if an agent is assigned to the customer they should approach the problem in different ways and notify the level of completion of the problem in each step by email. They can contact their agents by private chat and email.

4. REQUIREMENT ANALYSIS:

4.1. FUNCTIONAL REQUIREMENTS:

FR.NO	FUNCTIONAL REQUIREMENT (EPIC)	SUB REQUIREMENT (STORY / SUB- TASK)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through LinkedIn Register with valid mobile number
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP Two step verification for new device login.
FR-3	Agent Registration	Registration through Form Registration through Gmail Registration through LinkedIn Register with valid mobile number

FR-4	Agent Confirmation	Confirmation via Email Confirmation via OTP Two step verification for new device login.
FR-5	Admin	Admin have both user details and agent detail. Admin maintain agent allotment to the user based on problem's category.

4.2. NON- FUNCTIONAL REQUIREMENTS:

NFR NO.	NON-FUNCTIONAL REQUIREMENT	DESCRIPTION
NFR-1	Usability	To provide optimal usability for our proposed solution we have mainly concentrated on easier navigation throughout our website. For user, they can easily login with their credentials and also, they can register by themselves either with unique valid email id or with their mobile number if they don't have any prior account. After good navigation we have concentrated on visual clarity and developed web application which looks pleasant and simple thus making easier accessible to any aged person. For the first time users, Guide tour will also be available in order to provide better user satisfaction. Also, made our web application flexible to all type of

5. PROBLEM DESIGNING:

5.1. DATA FLOW DIAGRAMS:

FLOW DIAGRAM:

A data flow diagram (DFD) is a graphical or visual representation using a standardized set of symbols and notations to describe a business's operations through data movement. They are often elements of a formal methodology such asStructured Systems Analysis and Design Method (SSADM).

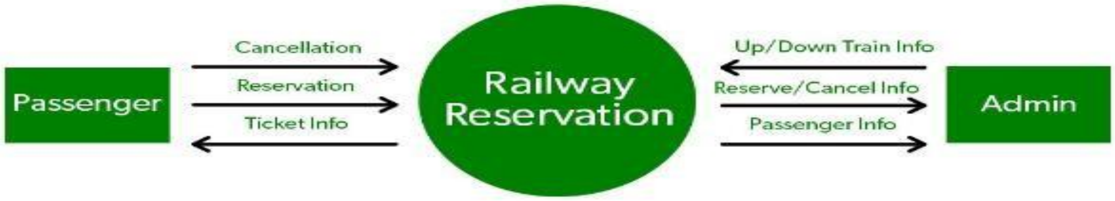
LEVELS IN DATA FLOW DIAGRAMS [DFD]:

In Software engineering DFD(data flow diagram) can be drawn to represent the system of different levels of abstraction. Higher-level DFDs are partitioned into low levels-hacking more information and functional elements. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see mainly 3 levels in the data flow diagram, which are:

- 0-level DFD
- 1-level DFD
- 2-level DFD

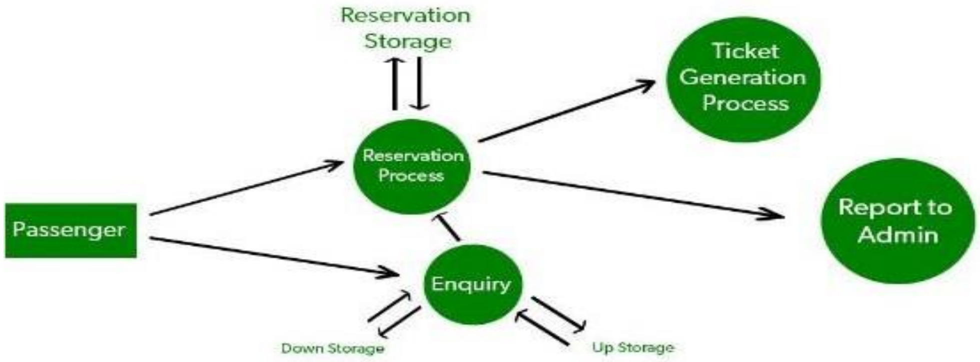
0- LEVEL DFD:

It is also known as a context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicatedby incoming/outgoing arrows.



1- LEVEL DFD:

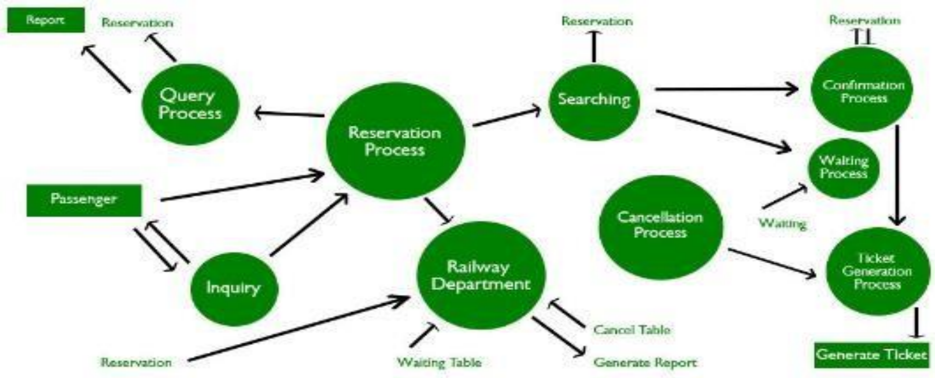
In 1-level DFD, the context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main functions of the system and breakdown the high-level process of 0-level DFD into subprocesses.



1-LEVEL DFD

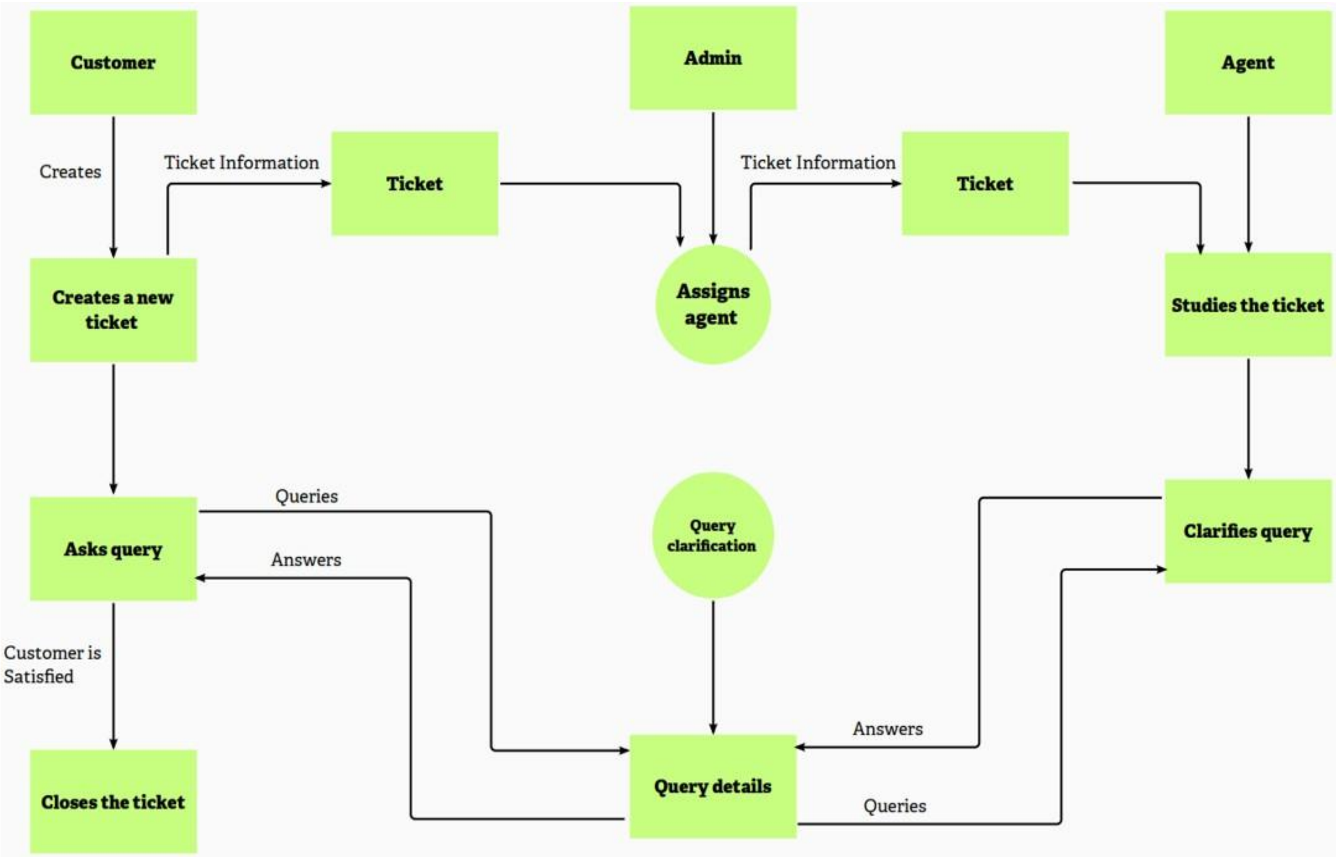
2- LEVEL DFD:

2-level DFD goes one step deeper into parts of 1-level DFD. It can be used to plan or record the specific/necessary detail about the system’s functioning.



2-LEVEL DFD

DATA FLOW DIAGRAM [CUSTOMER CARE REGISTRY]:



5.2. SOLUTION AND TECHNICAL ARCHITECTURE:

SOLUTION ARCHITECTURE:

A solution architecture (SA) is an architectural description of a specific solution. SAs combine guidance from different enterprise architecture viewpoints (business, information and technical), as well as from the enterprise solution architecture (ESA).

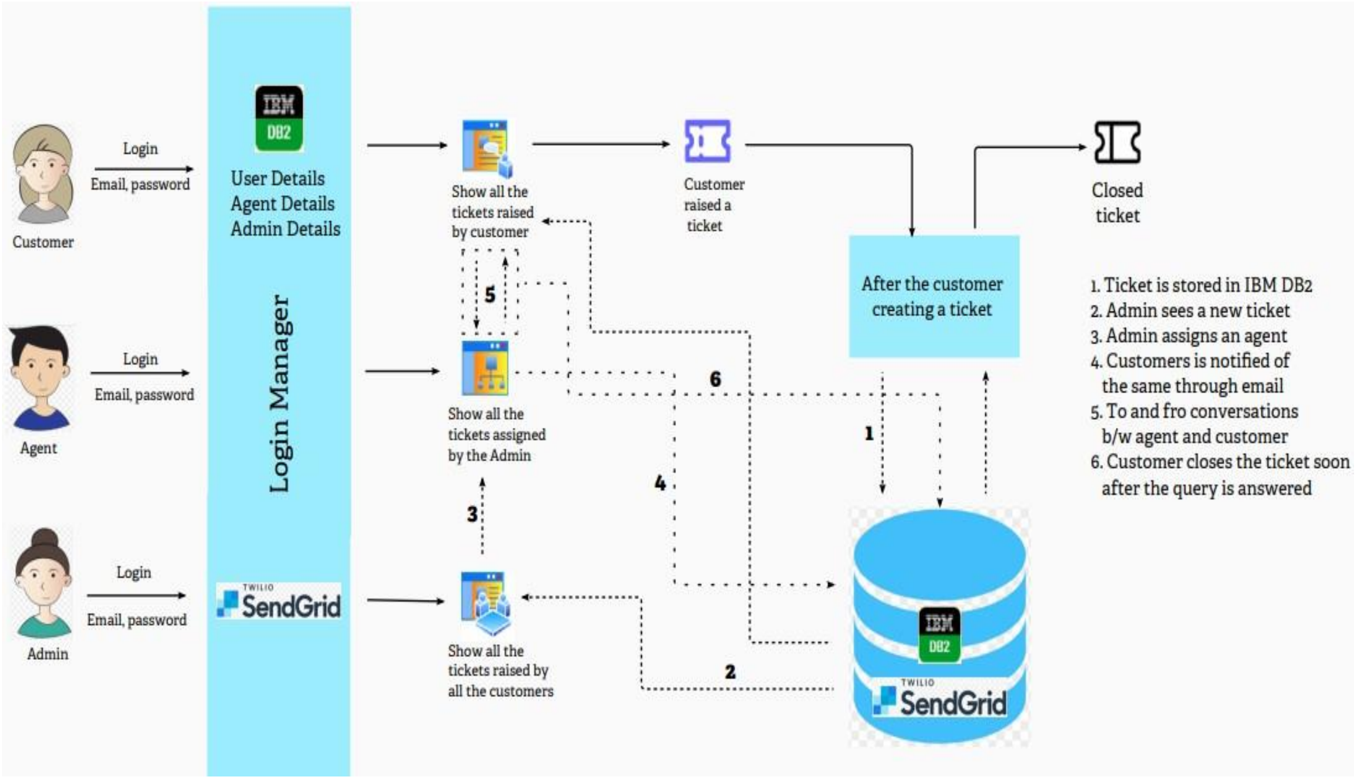
KEY FEATURES OF SOLUTION ARCHITECTURE:

Solution Architect: Processes, Role Description, Responsibilities, and Certifications

- Matching solutions with the corporate environment.
- Meeting the requirements of all stakeholders.
- Accounting for project constraints.
- Selecting the project technology stack.
- Compliance with non-functional requirements.

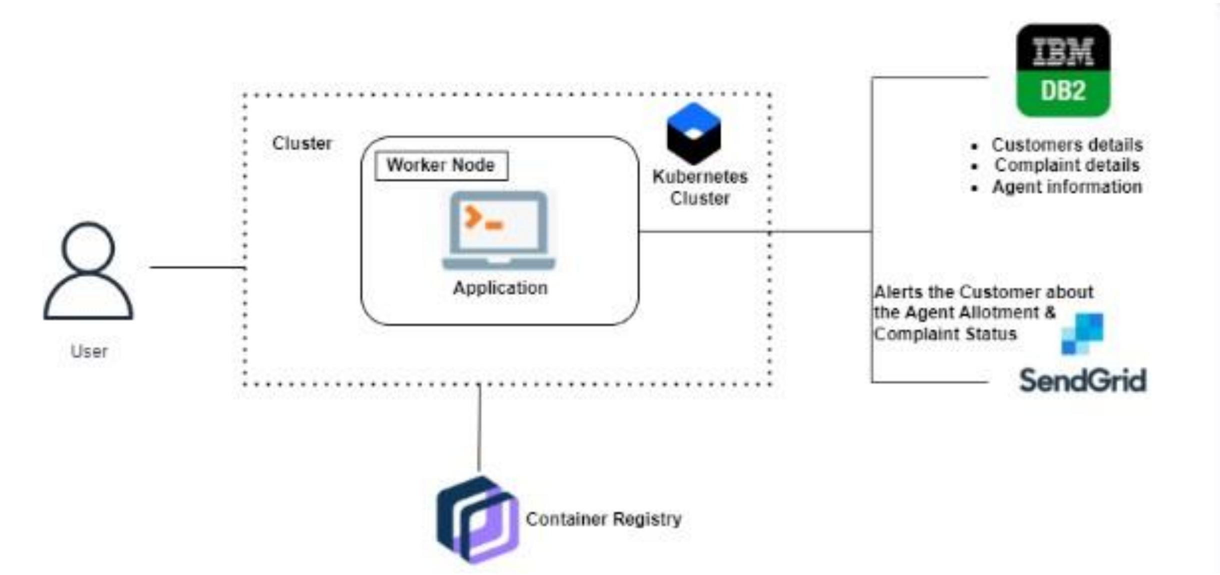
SOLUTION ARCHITECTURE DIAGRAM: [Customer Care Registry]

Based on the complexity of the deployment, a solution architecture diagram may actually be a set of diagrams documenting various levels of the architecture. The diagram relates the information that you gather on the environment to both physical and logical choices for your architecture in an easily understood manner.



1. Ticket is stored in IBM DB2
2. Admin sees a new ticket
3. Admin assigns an agent
4. Customers is notified of the same through email
5. To and fro conversations b/w agent and customer
6. Customer closes the ticket soon after the query is answered

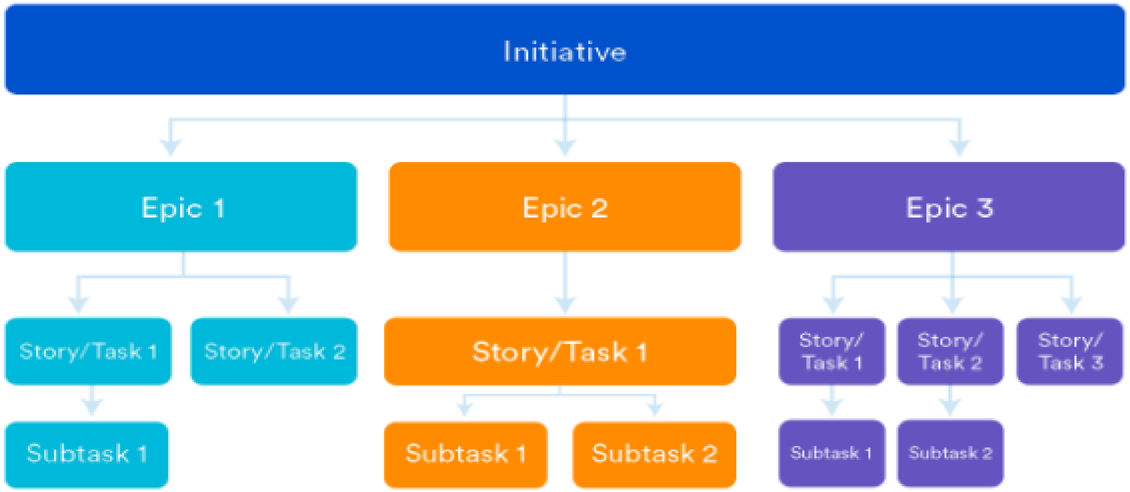
TECHNICAL ARCHITECTURE:



5.3. USER STORIES:

A user story is an informal, general explanation of a software feature written from the perspective of the end user or customer. The purpose of a user story is to articulate how a piece of work will deliver a particular value back to the customer.

EXAMPLE:



USER STOREIS [CUSTOMER CARE REGISTRY]:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Registration	USN-1	As a customer, I can register for the application by entering my email, password, and confirming my password.	I can access my account /dashboard	High	Sprint-1

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
	Login	USN-2	As a customer, I can login to the application by entering correct email and password	I can access my account /dashboard	High	Sprint-1
	Dashboard	USN-3	As a customer, I can see all the tickets raised by me and lot more	I get all the info needed in my dashboar d	High	Sprint-1
	Ticket creatio n	USN-4	As a customer, I can create a newticket with the detailed description of my query	I can ask my query	High	Sprint-2
	Address Column	USN-5	As a customer, I can have conversations with the assignedagent and get my queries clarified	My queries are clarified	High	Sprint-3
	Forgot passwor d	USN-6	As a customer, I can reset my password by thisoption in case I forgot my old password	I get access to my account again	Medium	Sprint-4

	Ticket details	USN-7	As a customer, I can see the current status ofmy tickets	I get better understanding	Medium	Sprint-4
Agent (Web user)	Login	USN-1	As an agent, I can login to the application by entering correct email and password	I can access my account /dashboard	High	Sprint-3

6. PROJECT PLANNING AND SCHEDULING:

6.1. SPRINT PLANNING AND ESTIMATION:

TITLE	DESCRIPTION	DATE
Literature Survey & Information Gathering	Literature survey on the selected project & gatheringinformation by referring the, technical papers, research publications etc.	5 OCTOBER 2022
Prepare Empathy Map	Prepare Empathy Map Canvasto capture the user Pains & Gains, Prepare list of problem statements	7 OCTOBER 2022
Ideation	List the by organizing the brainstorming session andprioritize the top 3 ideas based on the feasibility & importance.	11 OCTOBER 2022

Proposed Solution	Prepare the proposed solution document, which includes the novelty, feasibility of idea, business model, social impact, scalability of solution, etc.	14 OCTOBER 2022
Problem Solution Fit	Prepare problem - solution fit document.	16 OCTOBER 2022
Solution Architecture	Prepare solution architecture document.	17 OCTOBER 2022

Customer Journey	Prepare the customer journeymaps to understand the user interactions & experiences with the application (entry to exit).	23 OCTOBER 2022
Functional Requirement	Prepare the functional requirement document.	12 OCTOBER 2022
Data Flow Diagrams	Draw the data flow diagrams and submit forreview.	23 OCTOBER 2022
Technology Architecture	Prepare the technology architecture diagram.	20 OCTOBER 2022
Prepare Milestone & ActivityList	Prepare the milestones & activity list of the project.	22 OCTOBER 2022
Project Development - Delivery of Sprint-1, 2, 3 & 4	Develop & submit the developed code by testing it.	In Progress…

6.2 SPRINT DELIVERY SCHEDULE:

PRODUCT BACKLOG, SPRINT SCHEDULE AND ESTIMATION:PRODUCT BACKLOG:

A product backlog lists and prioritizes the task-level details required to execute thestrategic plan set forth in the roadmap. The backlog should communicate what’s next on the development team’s to-do list as they execute on the roadmap’s big- picture vision. Typical items in a product backlog include user stories, bug fixes, and other tasks.

SPRINT SCHEDULE:

Sprint Schedule means the applicable schedule posted by Sprint on Sprint’s web site.

There are 5 events in Sprint:

- Sprint Planning
- Daily Scrum

- Sprint Retrospection
- Sprint Demo and review
- Sprint Refinement

SPRINT	EPIC	USER STORY NUMBER	TASK (USER STORY)	STORY POINTS	PRIORITY	TEAM MEMBERS
SPRINT 1	REGISTRATION	USN-1	As a user I can register for the application by entering email and password.	2	HIGH	4
SPRINT 1	LOGIN	USN-2	As a user, I can login by entering registered email and password	3	HIGH	4
SPRINT 1	DASHBOARD	USN-3	As a user, I can tell my problems in given ticket	3	HIGH	4
SPRINT 1		USN-4	As a user, I canget email after registration in application	1	HIGH	4
SPRINT 1		USN-5	As a user, I canwait for my solution	2	HIGH	4

SPRINT 1	SOFTWARE	USN-6	IBM Watson Assistant Platform. For chat with each assistant.	2	HIGH	4
SPRINT 2		USN-7	As a user, I can check whether if Ireceived mail	1	HIGH	4
SPRINT 2		USN-8	As a user, I can register for the application through email.	2	MEDIUM	4
SPRINT 2	SOFTWARE	USN-9	Desktop dockerfor upload a container image	2	HIGH	4

SPRINT 3		USN-10	As a user, I candoubt for the given solution	2	HIGH	4
SPRINT 3	APPLICATION	USN-11	To develop asoftware application	2	HIGH	4
SPRINT 4	DATABASE	USN-12	As a user, I can view and access database information	2	HIGH	4

PROJECT TRACKER, VELOCITY, TASK BURNDOWN:

PROJECT TRACKER:

A project tracker is a tool that lets managers measure the progress of their team asthey execute tasks and use resources. It's an essential tool to keeping projects on schedule and within their budgets.

VELOCITY:

velocity is the measure of how much you can realistically accomplish within a sprint cycle. To work this out, you need to look at what your team accomplishedon previous sprints. Consider how long the sprint lasted and the volume of workcompleted.

TASK BURNDOWN:

Burndown chart is a major parameter used in agile software development and scrum to detect how much work remains to be completed. It is the graphical representation of showing the left-out portion of the task versus time. Generally,time is taken on the abscissa and left out work on ordinates.

References:

Project Tracker: The Ultimate Guide to Project Tracking (projectmanager.com)Overview of Burndown Chart in Agile - GeeksforGeeks

How to calculate sprint velocity | EasyRetro

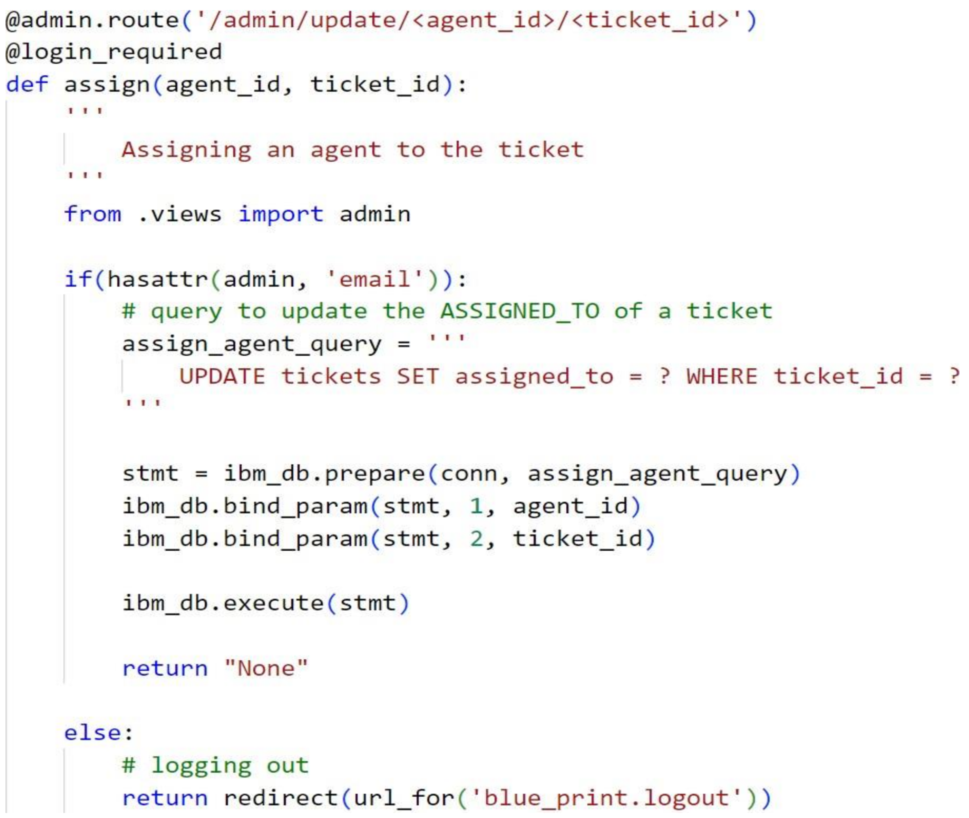
How many events are there in agile scrum? (c-sharpcorner.com)Sprint Schedule Definition | Law Insider

What is a Product Backlog? | Definition and Overview (productplan.com)

7.CODING AND SOLUTIONING

7.1 Admin assigning an agent to a ticket

Code:



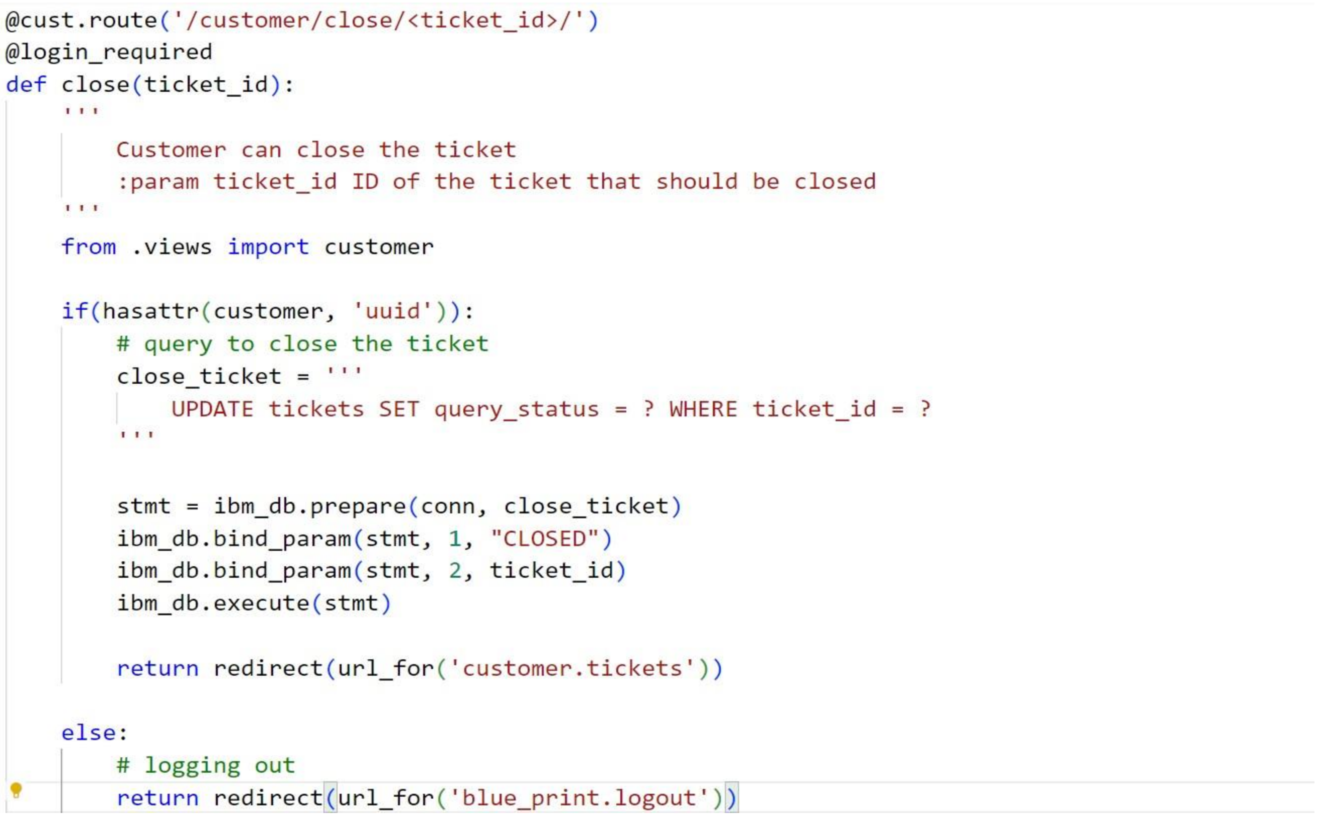
```
@admin.route('/admin/update/<agent_id>/<ticket_id>')
@login_required
def assign(agent_id, ticket_id):
    '''
        Assigning an agent to the ticket
    '''
    from .views import admin

    if(hasattr(admin, 'email')):
        # query to update the ASSIGNED_TO of a ticket
        assign_agent_query = '''
            UPDATE tickets SET assigned_to = ? WHERE ticket_id = ?
        '''

        stmt = ibm_db.prepare(conn, assign_agent_query)
        ibm_db.bind_param(stmt, 1, agent_id)
        ibm_db.bind_param(stmt, 2, ticket_id)

        ibm_db.execute(stmt)

        return "None"

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))
```

Explanation:

- User creates a ticket by describing the query
- Admin views the newly created ticket in the dashboard
- In the dropdown given, admin selects an agent
- Once selected, using fetch() the request is sent to the server
- The request URL contains both the Ticket ID and the selected Agent ID
- Using the shown SQL query, the assigned_to column of the tickets table is set to agent_idwhere the ticket_id column = ticket_id
- Then, the dashboard of the admin gets refreshed

7.2Customer closing a ticket

Code:



```
@cust.route('/customer/close/<ticket_id>/')
@login_required
def close(ticket_id):
    '''
        Customer can close the ticket
        :param ticket_id ID of the ticket that should be closed
    '''
    from .views import customer

    if(hasattr(customer, 'uuid')):
        # query to close the ticket
        close_ticket = '''
            UPDATE tickets SET query_status = ? WHERE ticket_id = ?
        '''

        stmt = ibm_db.prepare(conn, close_ticket)
        ibm_db.bind_param(stmt, 1, "CLOSED")
        ibm_db.bind_param(stmt, 2, ticket_id)
        ibm_db.execute(stmt)

        return redirect(url_for('customer.tickets'))

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))
```

Explanation:

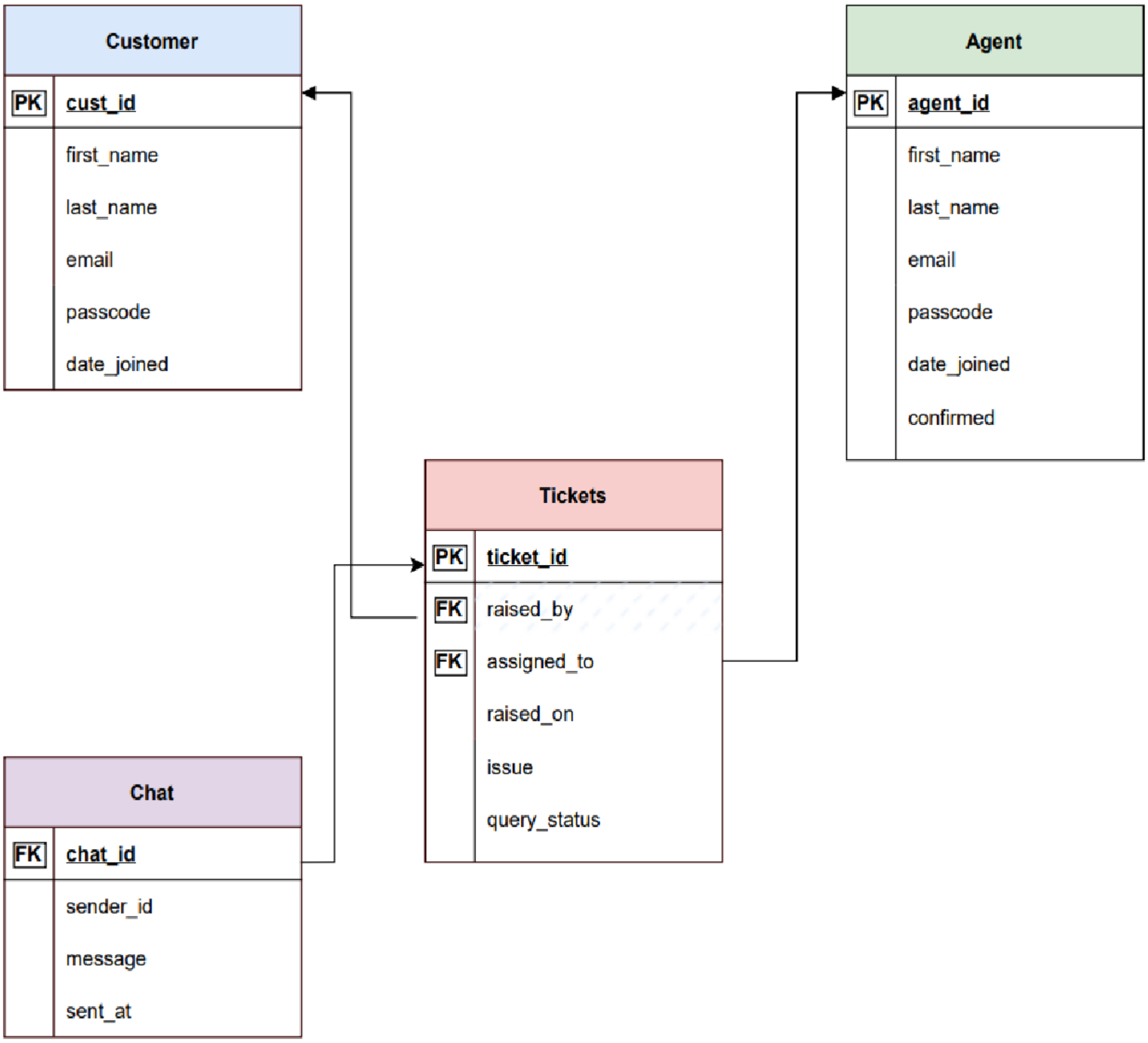
- User creates a ticket by describing the query
- Admin assigns an agent to this ticket
- The customer and the agent, chat with each other, in the view of clearing the customer’sdoubts
- Once the customer is satisfied, the customer decides to close the ticket
- Using fetch() the request is sent to the server. The requested URL contains the Ticket ID

- Using the shown SQL query, the status of the ticket is set to “CLOSED”
- Thus the ticket is closed
- Then the customer gets redirected to the all-tickets page

7.2 Database Schema

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains adescriptive detail of the database, which can be depicted by means of schema diagrams. It’s the database designers who design the schema to help programmers understand the database and makeit useful.


Customer
PK
cust_id
first_name
last_name
email
passcode
date_joined
Agent
PK
agent_id
first_name
last_name
email
passcode
date_joined
confirmed
Tickets
PK
ticket_id
FK
raised_by
FK
assigned_to
raised_on
issue
query_status
Chat
FK
chat_id
sender_id
message
sent_at

1. TESTING

8.1 Test Cases

The test case is defined as a group of conditions under which a tester determines whethera software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.

Test case gives detailed information about testing strategy, testing process, preconditions,and expected output. These are executed during the testing process to check whether the software application is performing the task for that it was developed or not.

Test case helps the tester in defect reporting by linking defect with test case ID. Detailed test case documentation works as a full proof guard for the testing team because if developer missed something, then it can be caught during execution of these full-proof test cases.

To write the test case, we must have the requirements to derive the inputs, and the test scenarios must be written so that we do not miss out on any features for testing. Then we should have the test case template to maintain the uniformity, or every test engineer follows the same approach to prepare the test document.

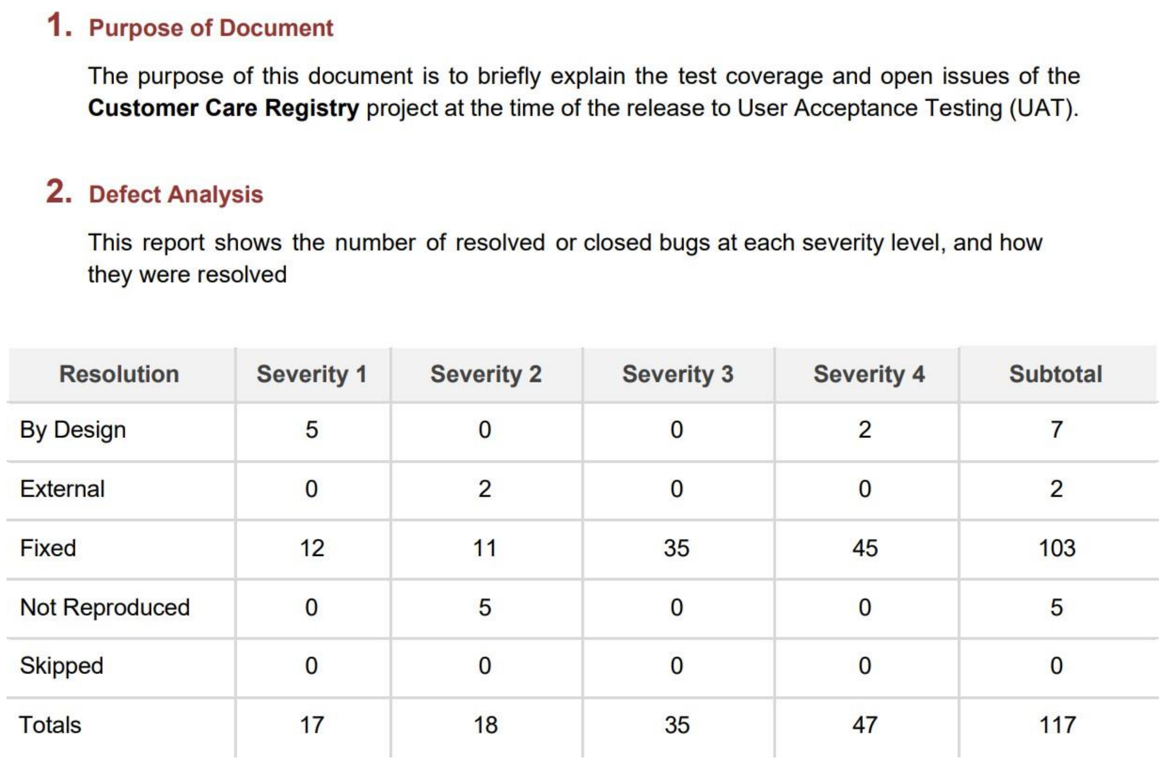
Test Cases Performed:

1. Test Cases

   Report : Click

   here

8.2 User Acceptance Testing



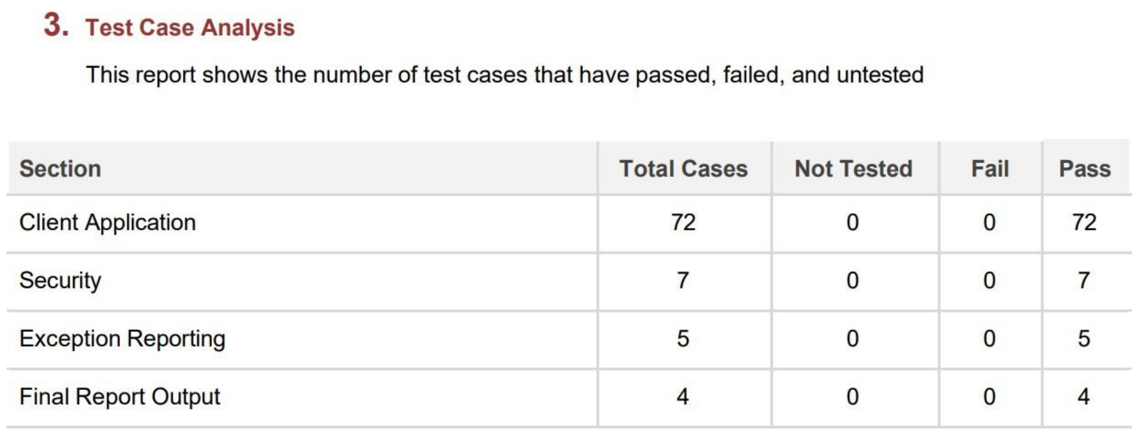
1. Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of the **Customer Care Registry** project at the time of the release to User Acceptance Testing (UAT).

2. Defect Analysis

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	5	0	0	2	7
External	0	2	0	0	2
Fixed	12	11	35	45	103
Not Reproduced	0	5	0	0	5
Skipped	0	0	0	0	0
Totals	17	18	35	47	117



3. Test Case Analysis

This report shows the number of test cases that have passed, failed, and untested

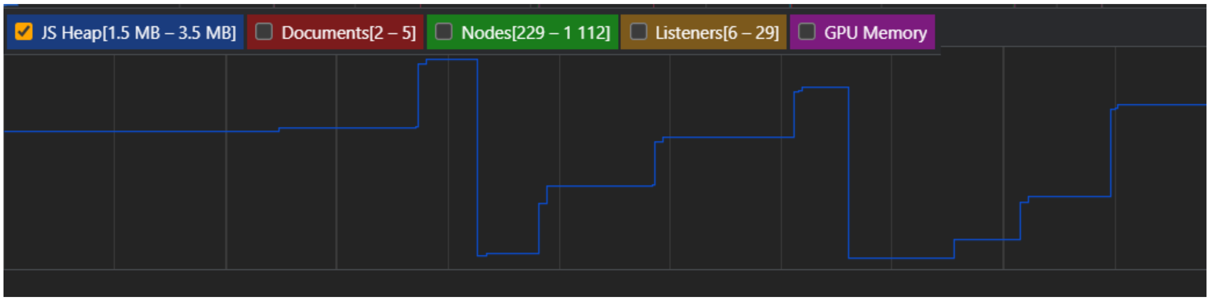
Section	Total Cases	Not Tested	Fail	Pass
Client Application	72	0	0	72
Security	7	0	0	7
Exception Reporting	5	0	0	5
Final Report Output	4	0	0	4

9.RESULTS

9.1 Performance Metrics:

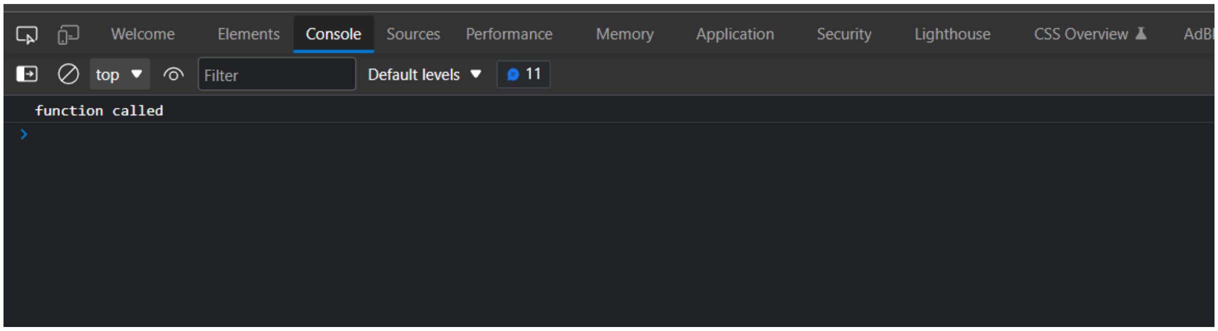
CPU usage:

- ✓ Since all the operations run using Flask is in server-side, the client (browser) need not worryabout the CPU usage. Just rendering the page, static contents take place in the client-side.
- ✓ Memory for client-side functions (Javascript) is allocated using heap. It can be eitherincreased based upon the requirement or removed from the heap.



Errors:

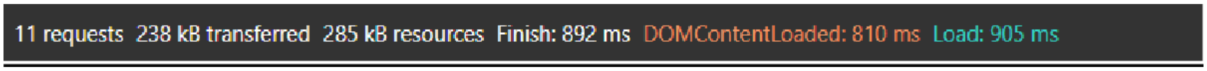
- ✓ Since all the backend functions are done using flask, any exceptions / errors rising are well-handled. Though they appear, user's interaction with the site is not affected in any way



```
function called
```

Latency and Response time:

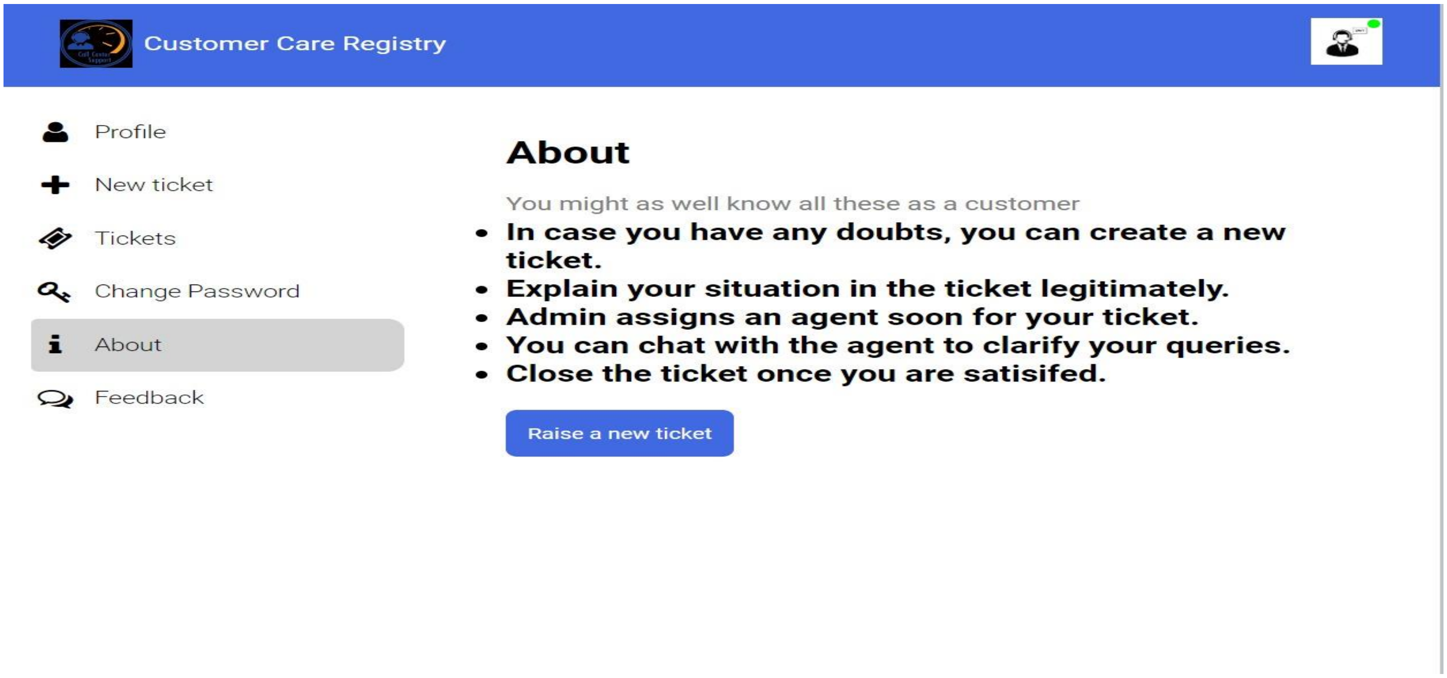
It takes less than a second to load a page in the client. From this it is evident that there is low latency



11 requests 238 kB transferred 285 kB resources Finish: 892 ms DOMContentLoaded: 810 ms Load: 905 ms

9.2. OUTPUT:

About Page:

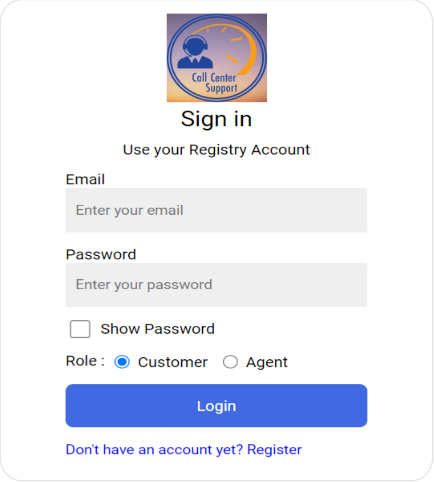


About

You might as well know all these as a customer

- In case you have any doubts, you can create a new ticket.
- Explain your situation in the ticket legitimately.
- Admin assigns an agent soon for your ticket.
- You can chat with the agent to clarify your queries.
- Close the ticket once you are satisifed.

Login Page:



Sign in

Use your Registry Account

Email

Enter your email

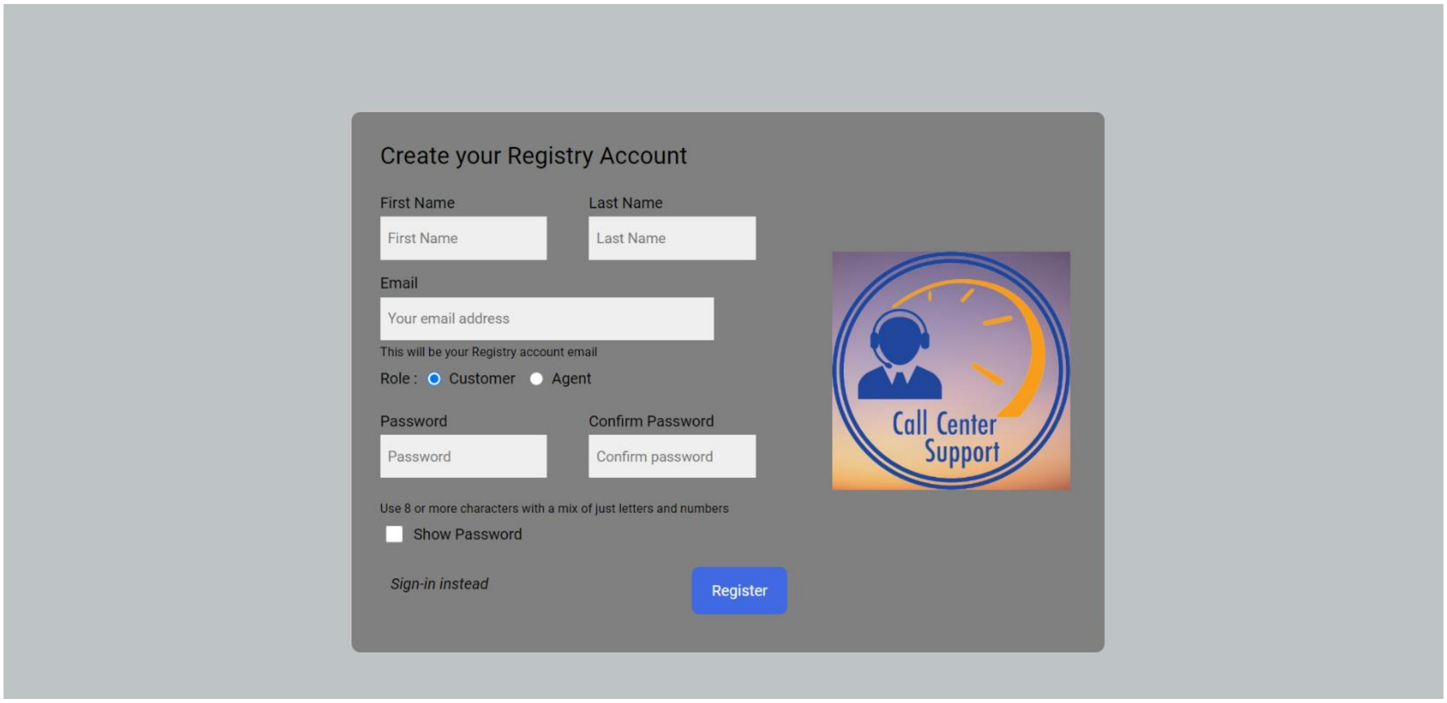
Password

Enter your password

□ Show Password

Role : ● Customer ○ Agent

Registration Page:



Create your Registry Account

First Name: First Name

Last Name: Last Name

Email: Your email address

This will be your Registry account email

Role : ● Customer ○ Agent

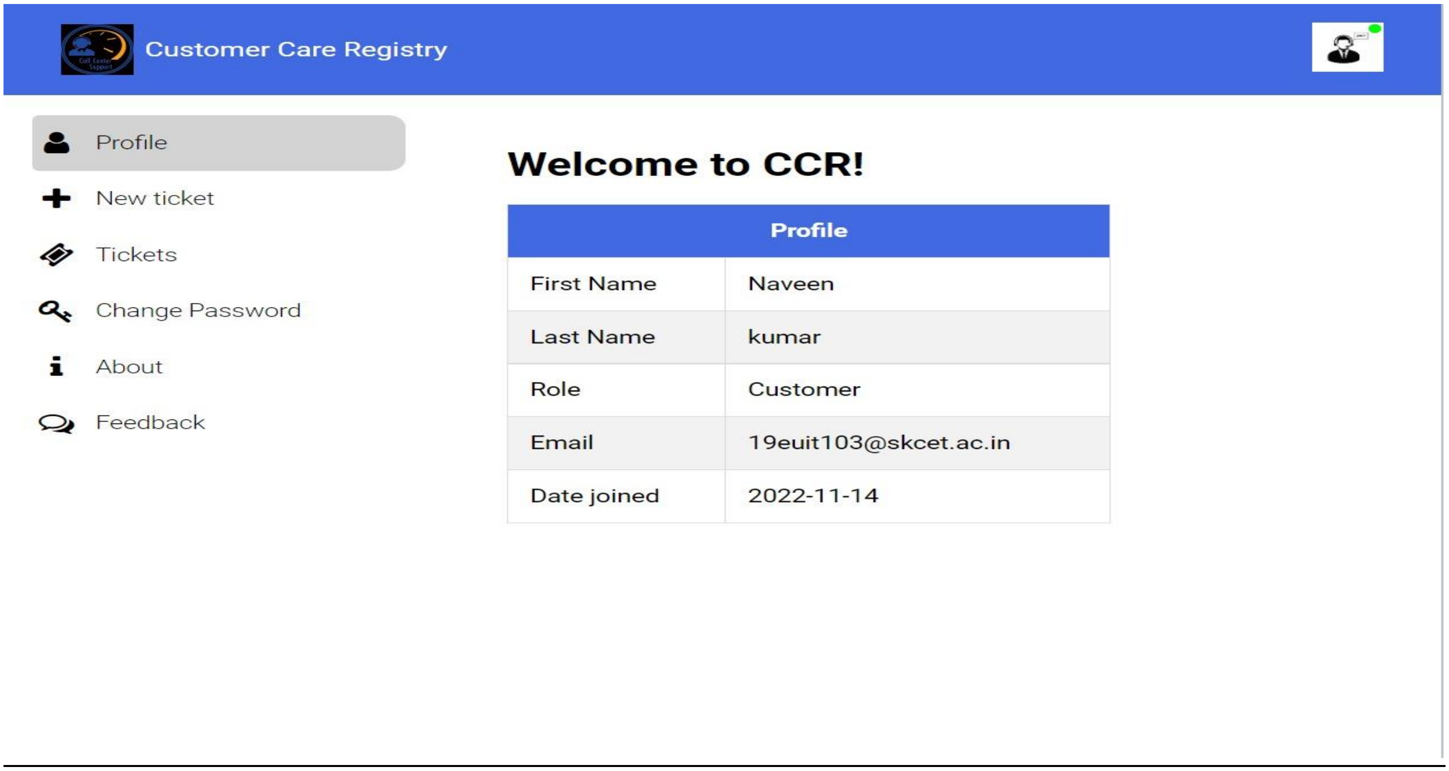
Password: Password

Confirm Password: Confirm password

Use 8 or more characters with a mix of just letters and numbers

□ Show Password

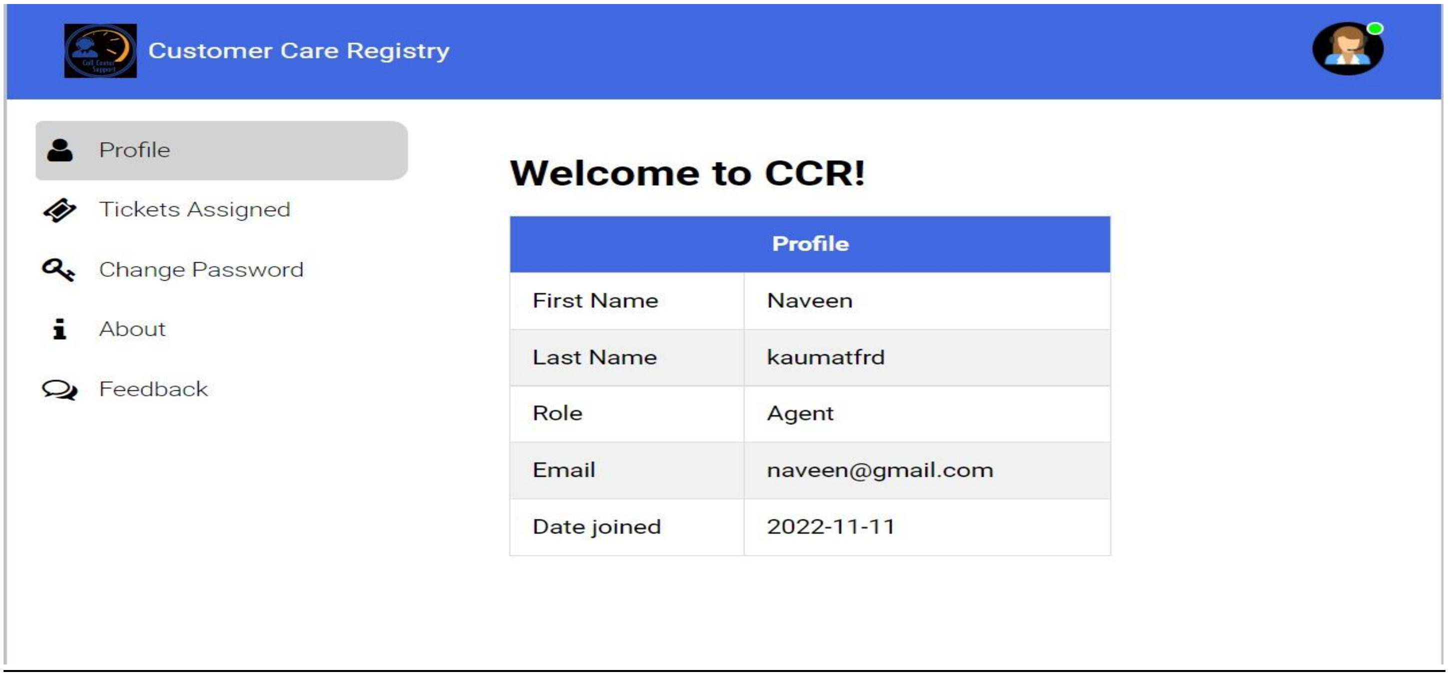
Customer Profile Page:



Welcome to CCR!

Profile	
First Name	Naveen
Last Name	kumar
Role	Customer
Email	19euit103@skcet.ac.in
Date joined	2022-11-14

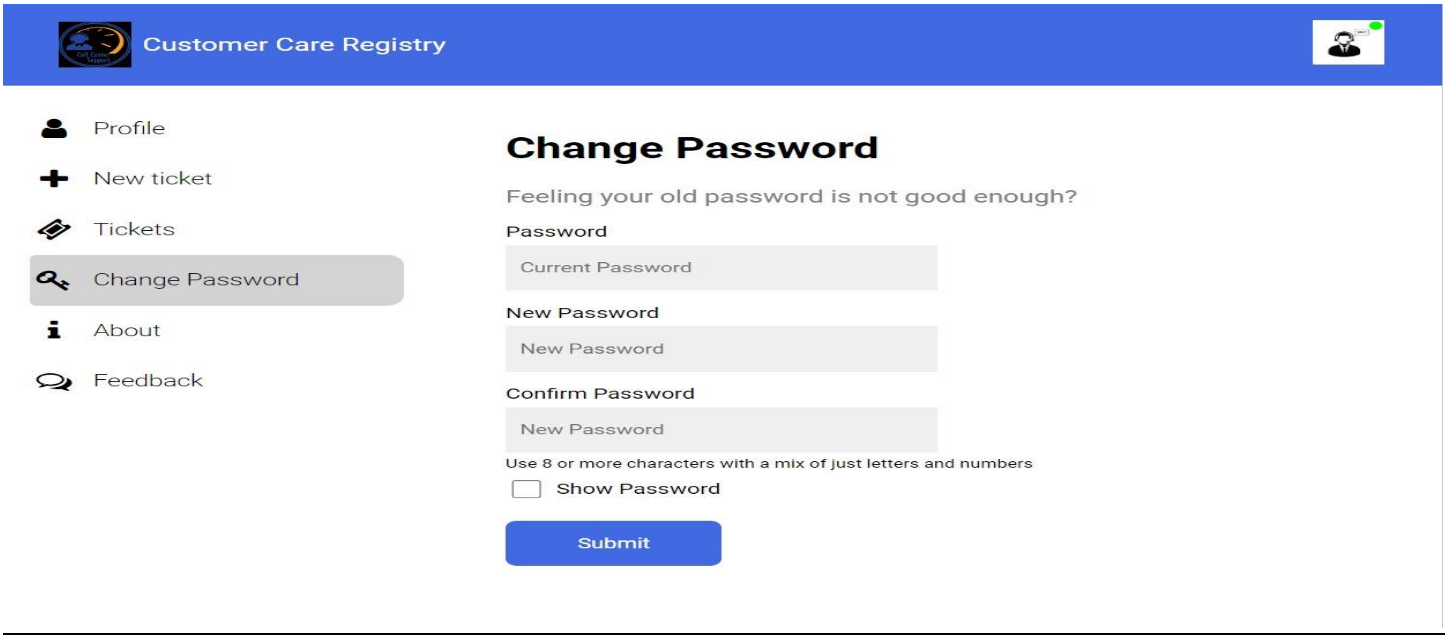
Agent Profile Page:



Welcome to CCR!

Profile	
First Name	Naveen
Last Name	kaumatfrd
Role	Agent
Email	naveen@gmail.com
Date joined	2022-11-11

Change Password:



Change Password

Feeling your old password is not good enough?

Password

Current Password

New Password

New Password

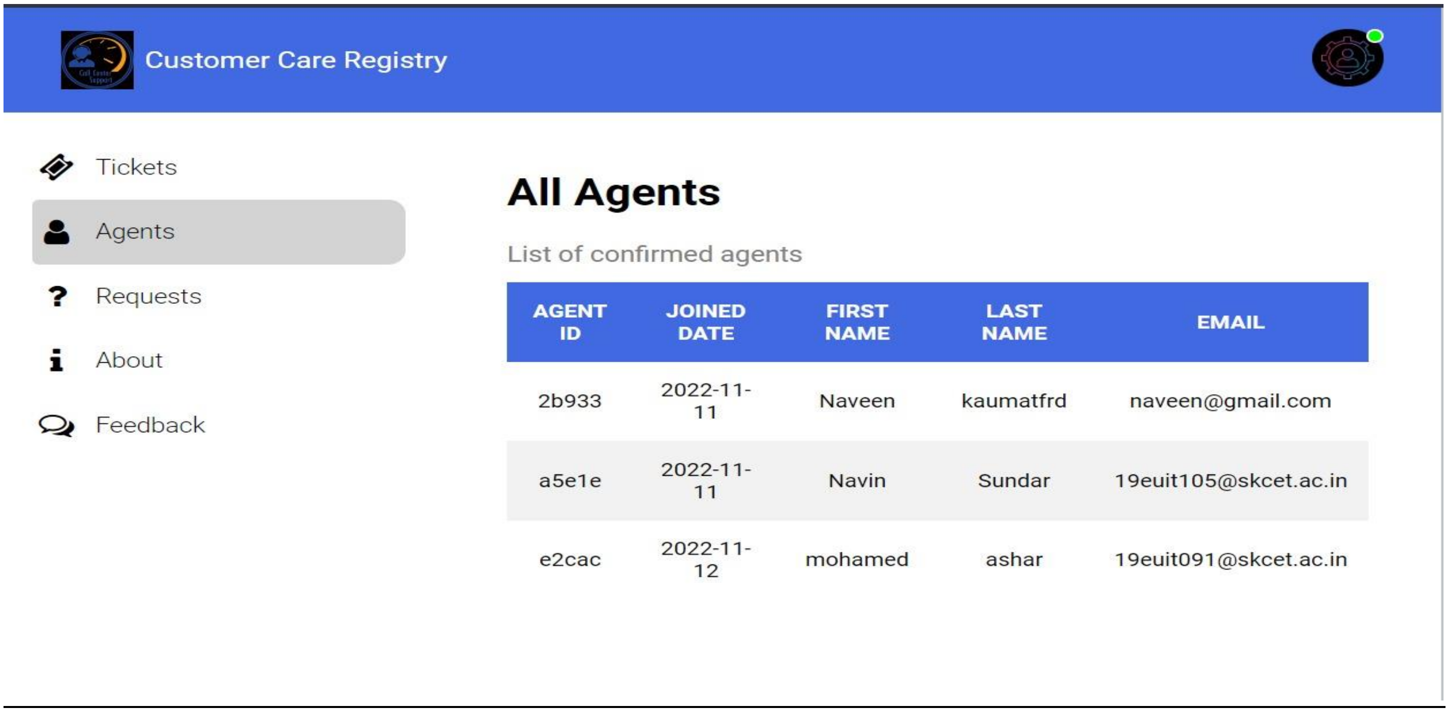
Confirm Password

New Password

Use 8 or more characters with a mix of just letters and numbers

□ Show Password

All Agents Page:

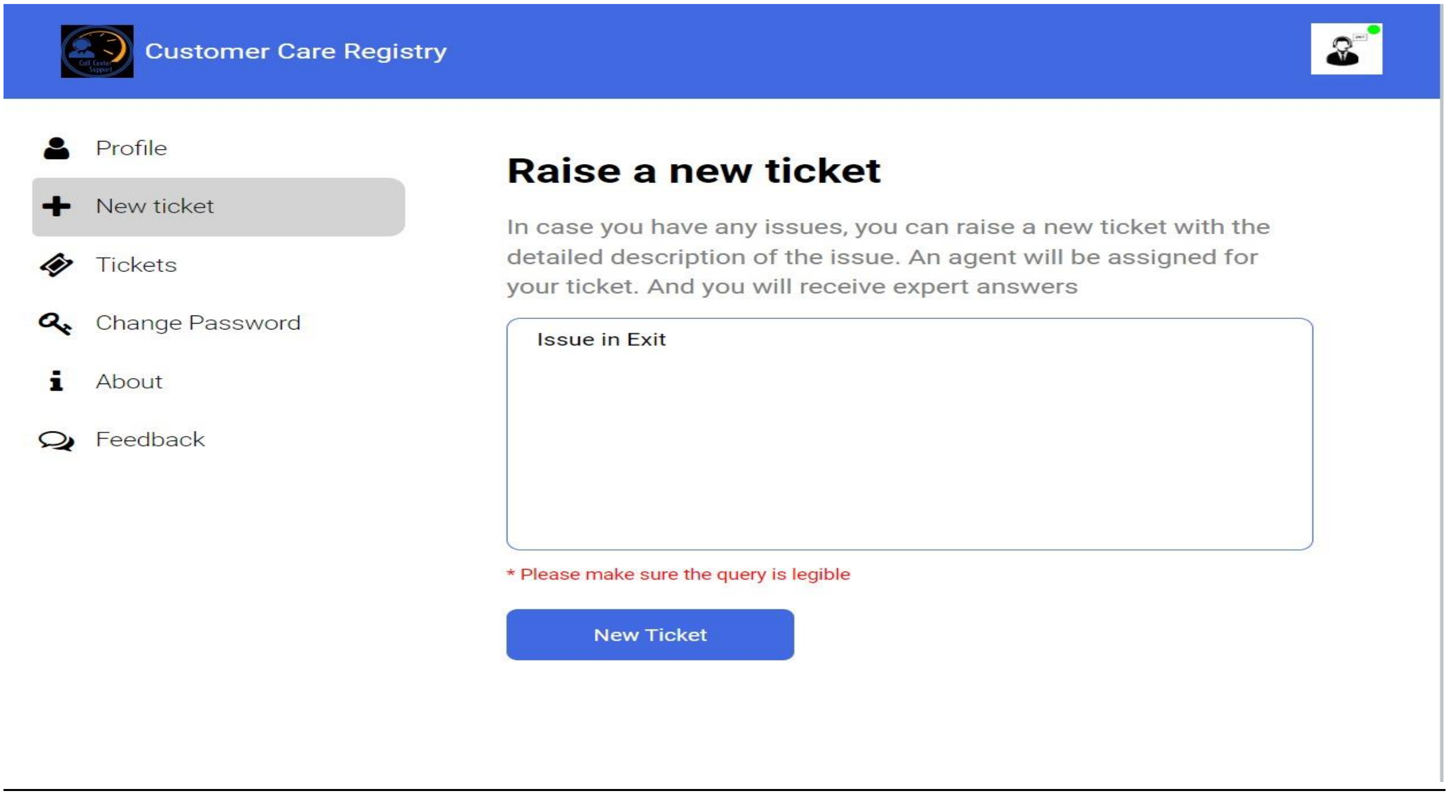


All Agents

List of confirmed agents

AGENT ID	JOINED DATE	FIRST NAME	LAST NAME	EMAIL
2b933	2022-11-11	Naveen	kaumatfrd	naveen@gmail.com
a5e1e	2022-11-11	Navin	Sundar	19euit105@skcet.ac.in
e2cac	2022-11-12	mohamed	ashar	19euit091@skcet.ac.in

Raise a New Ticket:



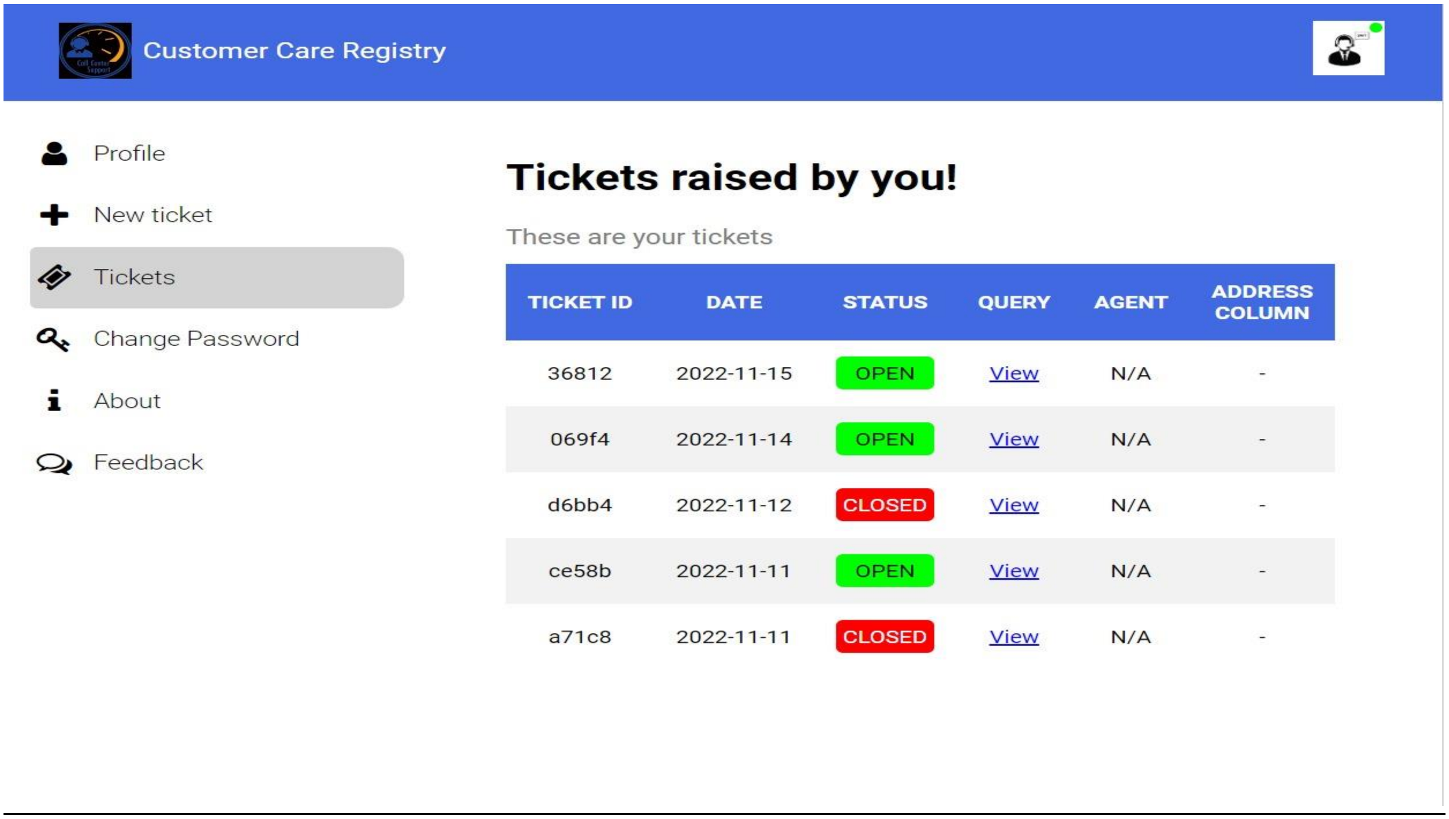
Raise a new ticket

In case you have any issues, you can raise a new ticket with the detailed description of the issue. An agent will be assigned for your ticket. And you will receive expert answers

Issue in Exit

* Please make sure the query is legible

Tickets Page:

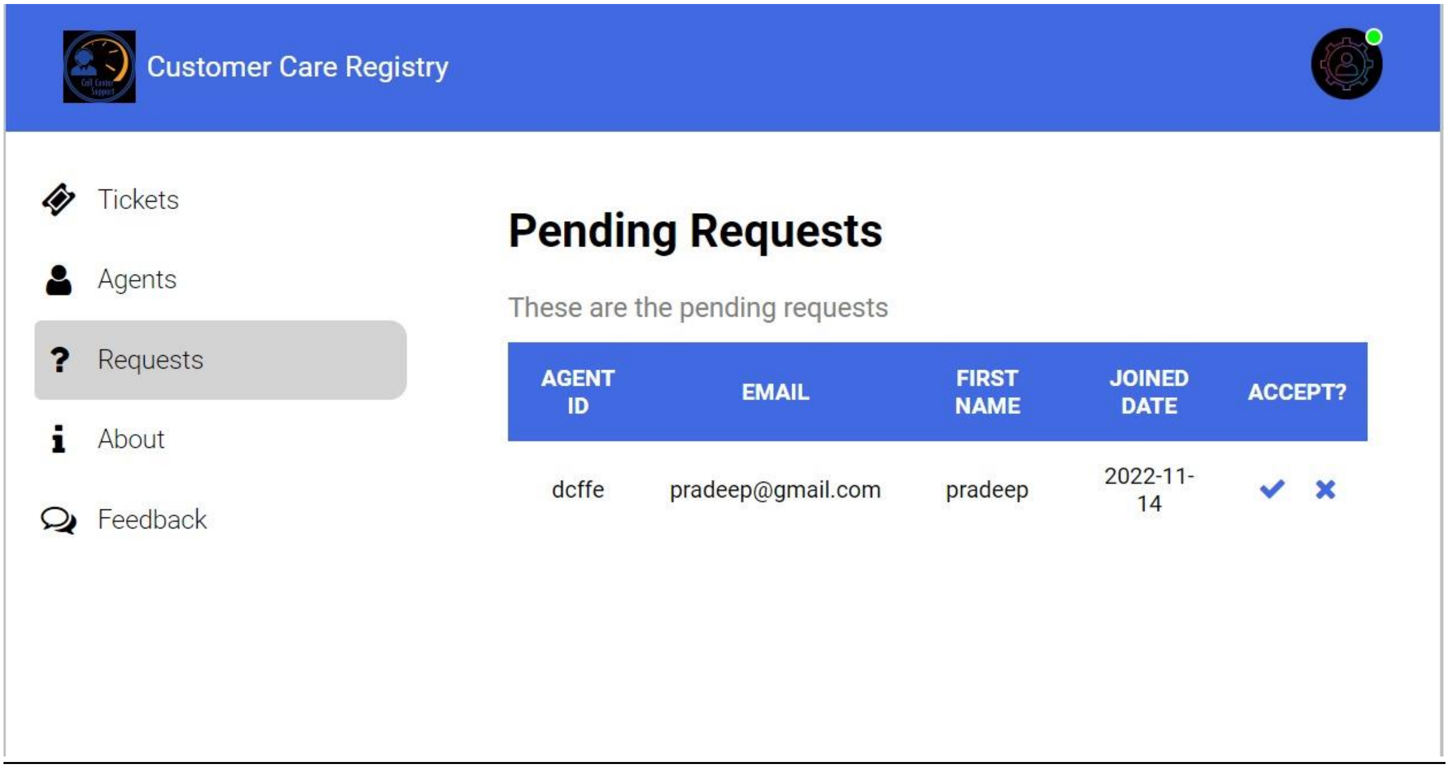


Tickets raised by you!

These are your tickets

TICKET ID	DATE	STATUS	QUERY	AGENT	ADDRESS COLUMN
36812	2022-11-15	OPEN	View	N/A	-
069f4	2022-11-14	OPEN	View	N/A	-
d6bb4	2022-11-12	CLOSED	View	N/A	-
ce58b	2022-11-11	OPEN	View	N/A	-
a71c8	2022-11-11	CLOSED	View	N/A	-

Pending Requests:

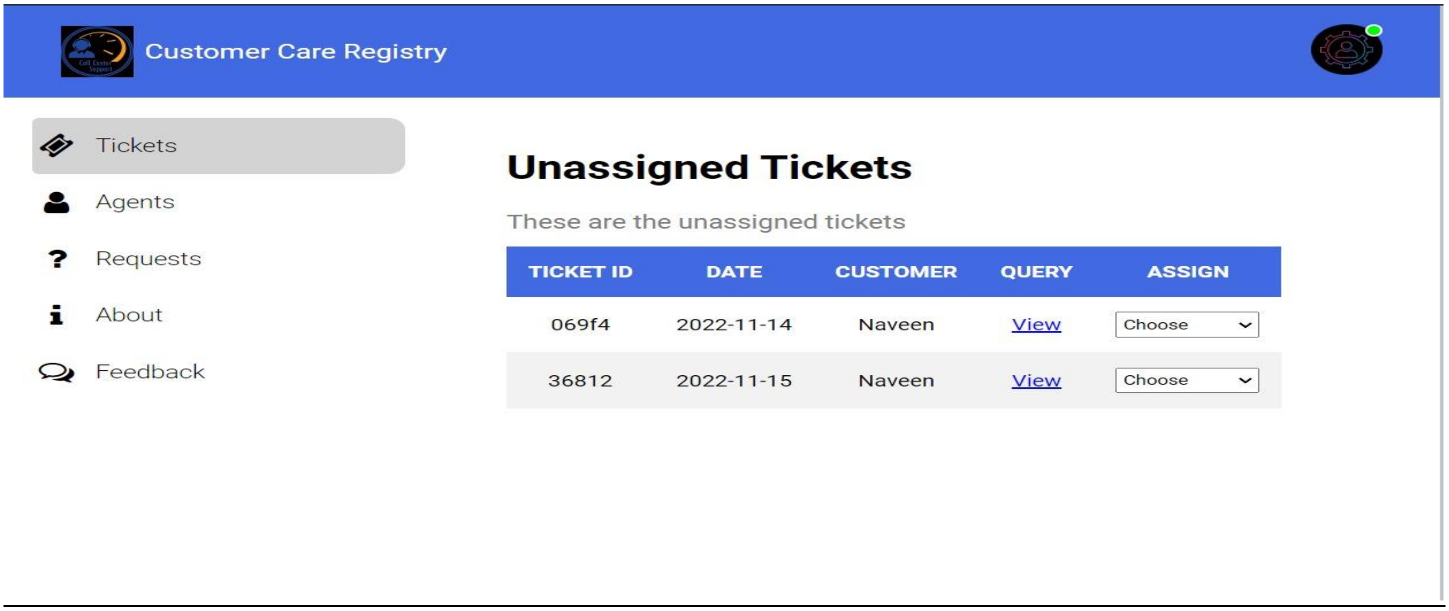


Pending Requests

These are the pending requests

AGENT ID	EMAIL	FIRST NAME	JOINED DATE	ACCEPT?
dcffe	pradeep@gmail.com	pradeep	2022-11-14	✓ ✗

Unassigned Ticket:

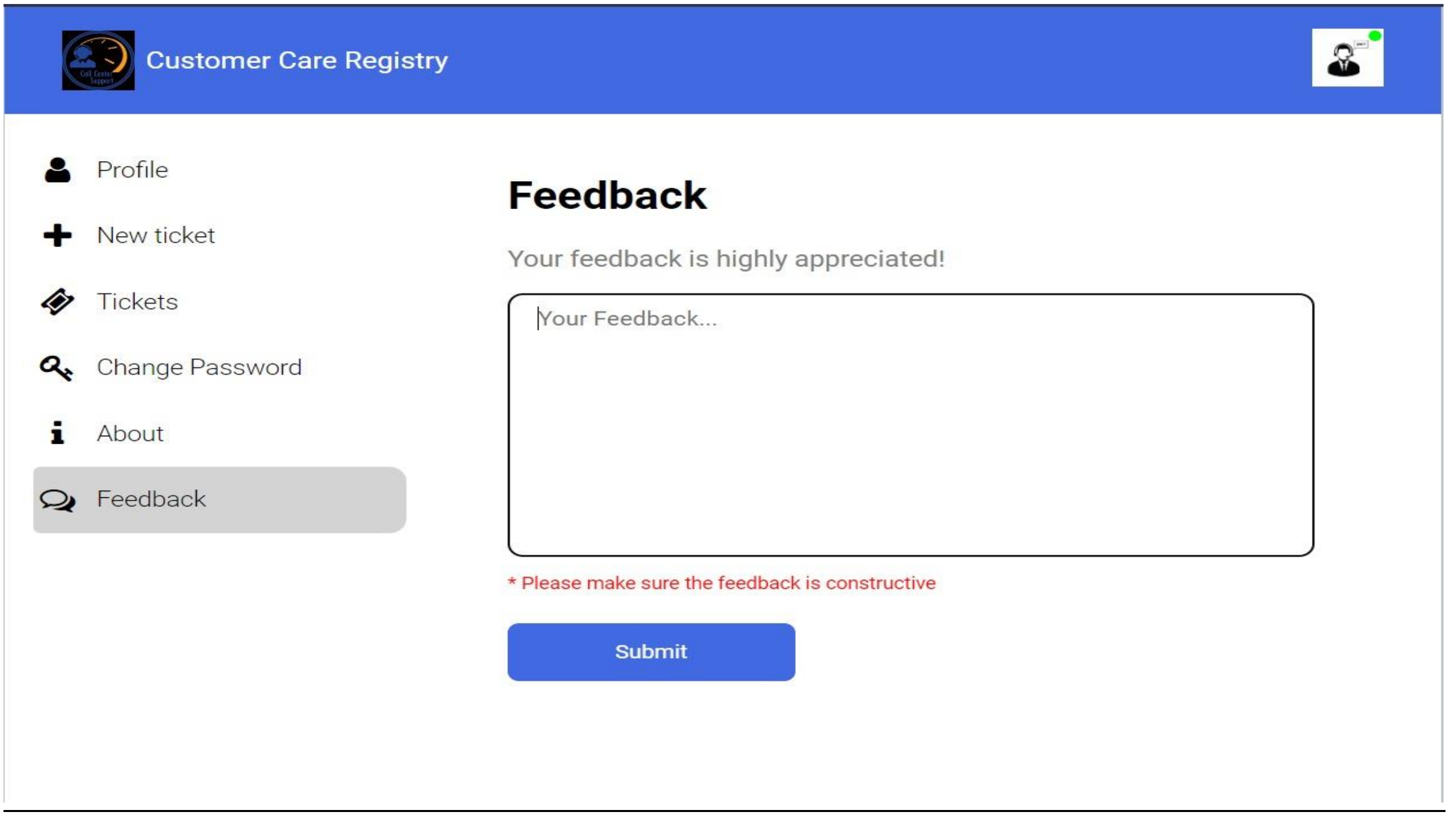


Unassigned Tickets

These are the unassigned tickets

TICKET ID	DATE	CUSTOMER	QUERY	ASSIGN
069f4	2022-11-14	Naveen	View	Choose
36812	2022-11-15	Naveen	View	Choose

Feedback:



Feedback

Your feedback is highly appreciated!

Your Feedback...

* Please make sure the feedback is constructive

10. ADVANTAGES AND DISADVANTAGES:

ADVANTAGES:

- Customer can raise their problem by creating the ticket.
- Agent will be assigned to customer's ticket by admin, so that they can get solution from him.
- After creating ticket, customer can chat privately with agent to get his doubts to be clarified.

DISADVANTAGES:

- Customer can get his doubts clarified by agent only through chat.
- Agent cannot be active at any time.
- There is no time limit fixed for a ticket to be get completed.

11. CONCLUSION:

Thus, there are many customer service applications available on the internet. Noting down the structural components of those applications and building a customer care registry. It will be web application build with Flask (Python micro-web framework), HTML, JavaScript. It will be a ticket-based customer service registry. Customers can register into the application using their email, password, and a username. Then, they can login to the system, and raise as queries as they want in the form of their tickets. These tickets will be sent to the admin, for which an agent is assigned. Then, the assigned agent will have a one-to-one chat with the customer and the latter's queries will be clarified. It is also the responsibility of the admin, to create an agent.

12. FUTURE SCOPE:

In future, we are trying to add email feature with progress bar of the task inside the mail to show the customer to satisfy them and the mail is sent to customer from agent. And also, trying to add some of the features to the application. So, that the customer can ask their queries at any time and they will get a reply from the agent. So that, we can satisfy the customer and customer will give positive feedback to us.

13. APPENDIX:

Flask:

- ✓ Flask is a micro web framework written in Python. It is classified as a microframework because itdoes not require particular tools or libraries
- ✓ It has no database abstraction layer, form validation, or any other components where pre-existingthird-party libraries provide common functions

JavaScript:

- ✓ JavaScript, often abbreviated as JS, is a programming language that is one of the coretechnologies of the World Wide Web, alongside HTML and CSS
- ✓ As of 2022, 98% of websites use JavaScript on the client side for webpage behavior, oftenincorporating third-party libraries

IBM Cloud:

- ✓ IBM cloud computing is a set of cloud computing services for business offered by the informationtechnology company IBM

Kubernetes:

- ✓ Kubernetes is an open-source container orchestration system for automating softwaredeployment, scaling, and management

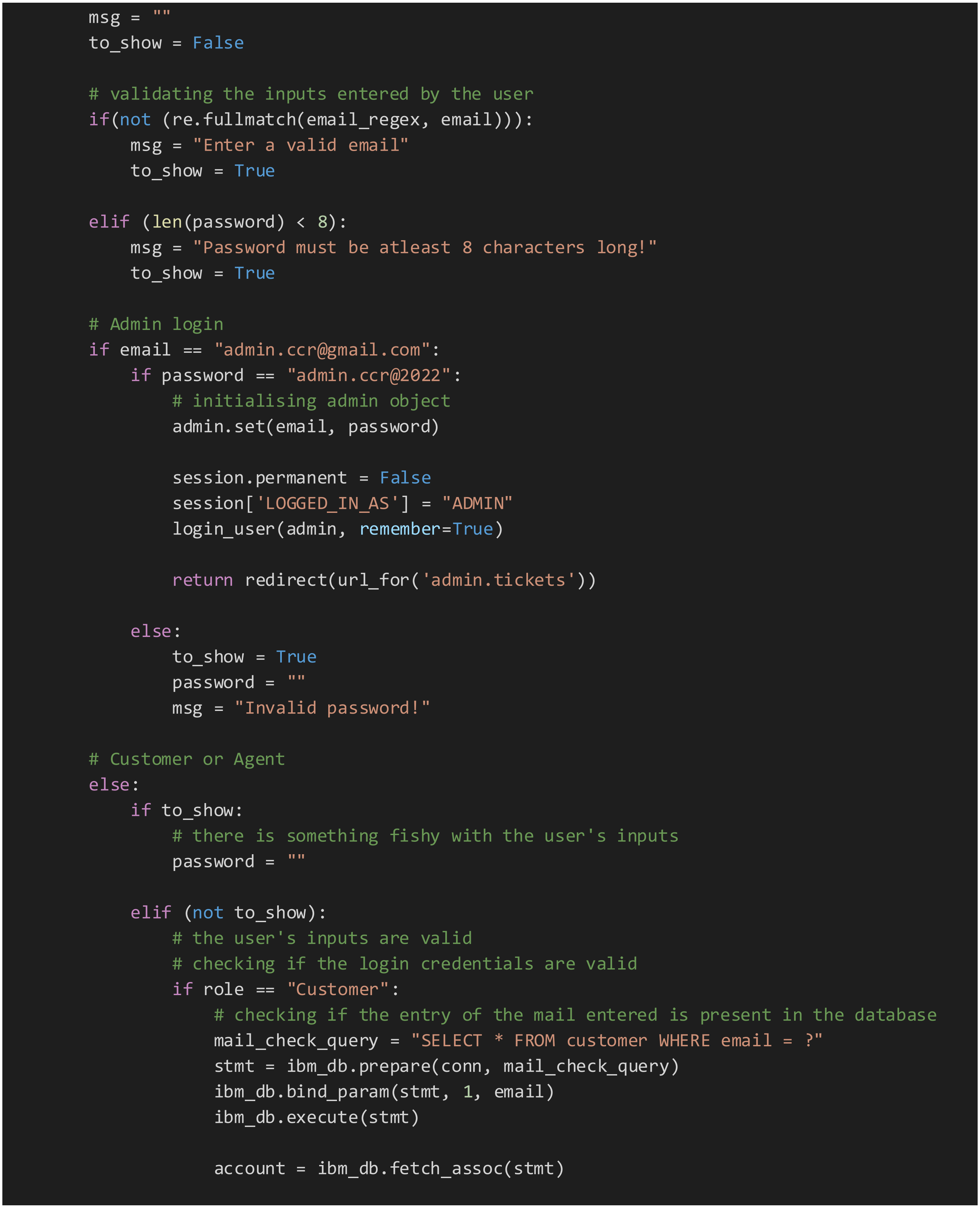
Docker:

- ✓ Docker is a set of platforms as a service product that use OS-level virtualization to deliversoftware in packages called containers

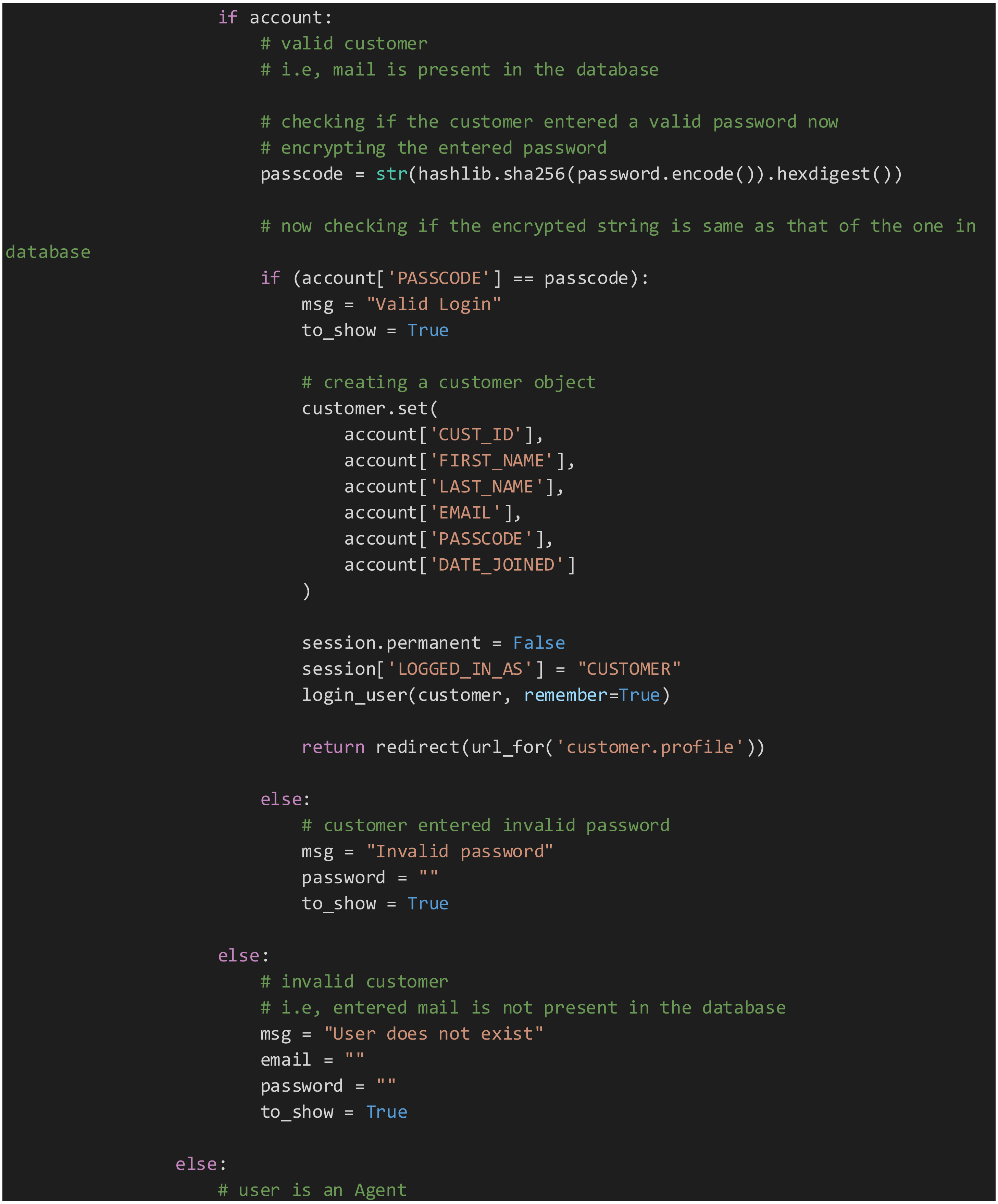
13.1. SOURCE CODE:

SAMPLE CODE:

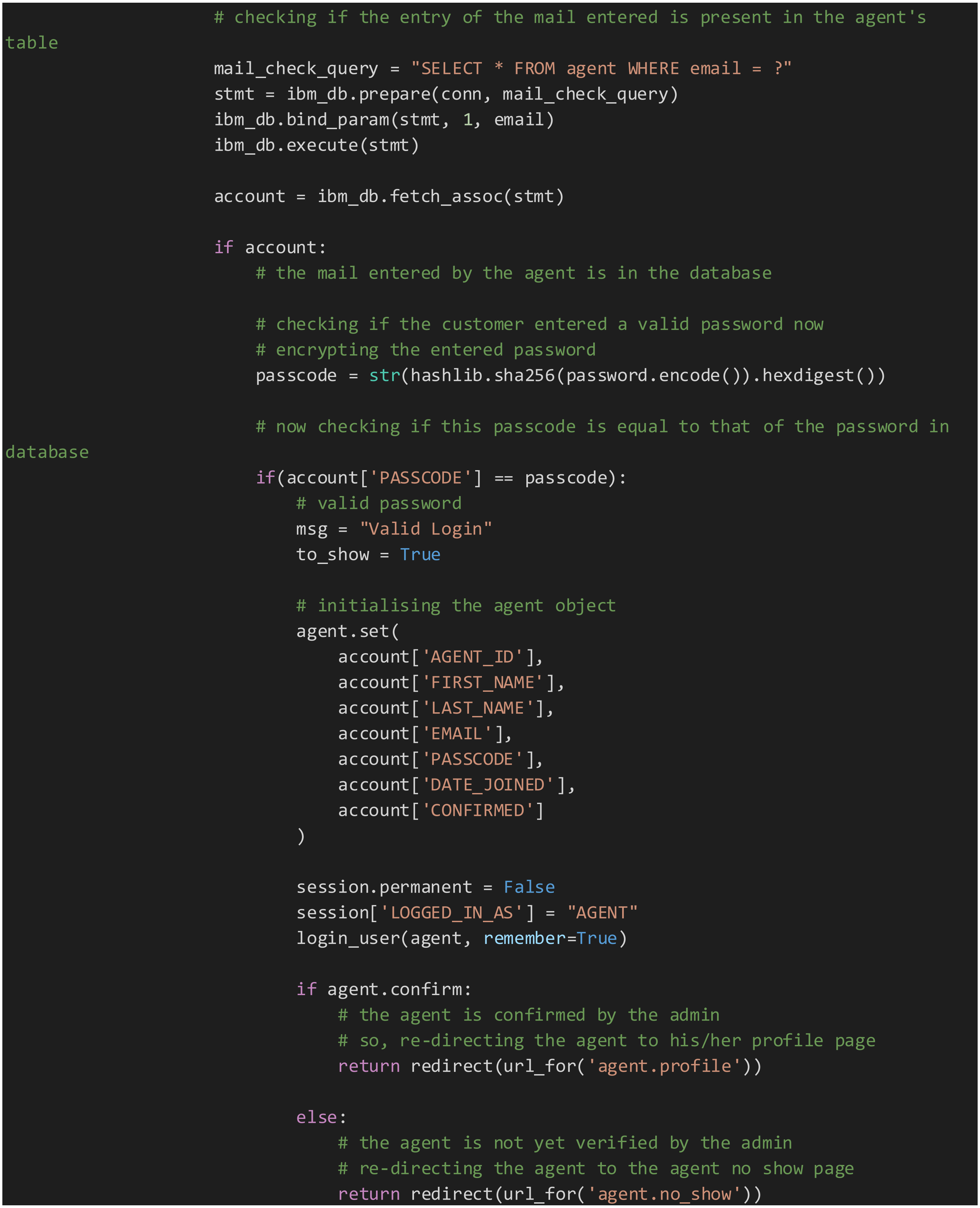
```
from flask import Blueprint, render_template, request, redirect, session, url_for
import hashlib
import re
from flask_login import login_required, login_user, logout_user
import ibm_db
import uuid
from datetime import date
import random
from registry.model import Customer, Agent, Admin, Mail

views = Blueprint("blue_print", __name__)
email_regex = r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b"
pass_regex = r"^[A-Za-z0-9_-]*$"

customer = Customer()
agent = Agent()
admin = Admin()
mail = Mail()

conn = ibm_db.connect('DATABASE=bludb;HOSTNAME=815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30367;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=mxx40464;PWD=9xoG1eDiCcNIrouX', '', '')


@views.route('/logout')
@login_required
def logout():
    session.pop('LOGGED_IN_AS')
    logout_user()

    return redirect(url_for('blue_print.login'))

@views.route('/', methods = ['GET', 'POST'])
@views.route('/login', methods = ['GET', 'POST'])
def login():
    # if method is POST
    if request.method == 'POST':
        # getting the data entered by the user
        email = request.form.get('email')
        password = request.form.get('password')
        role = request.form.get('role-check')
```



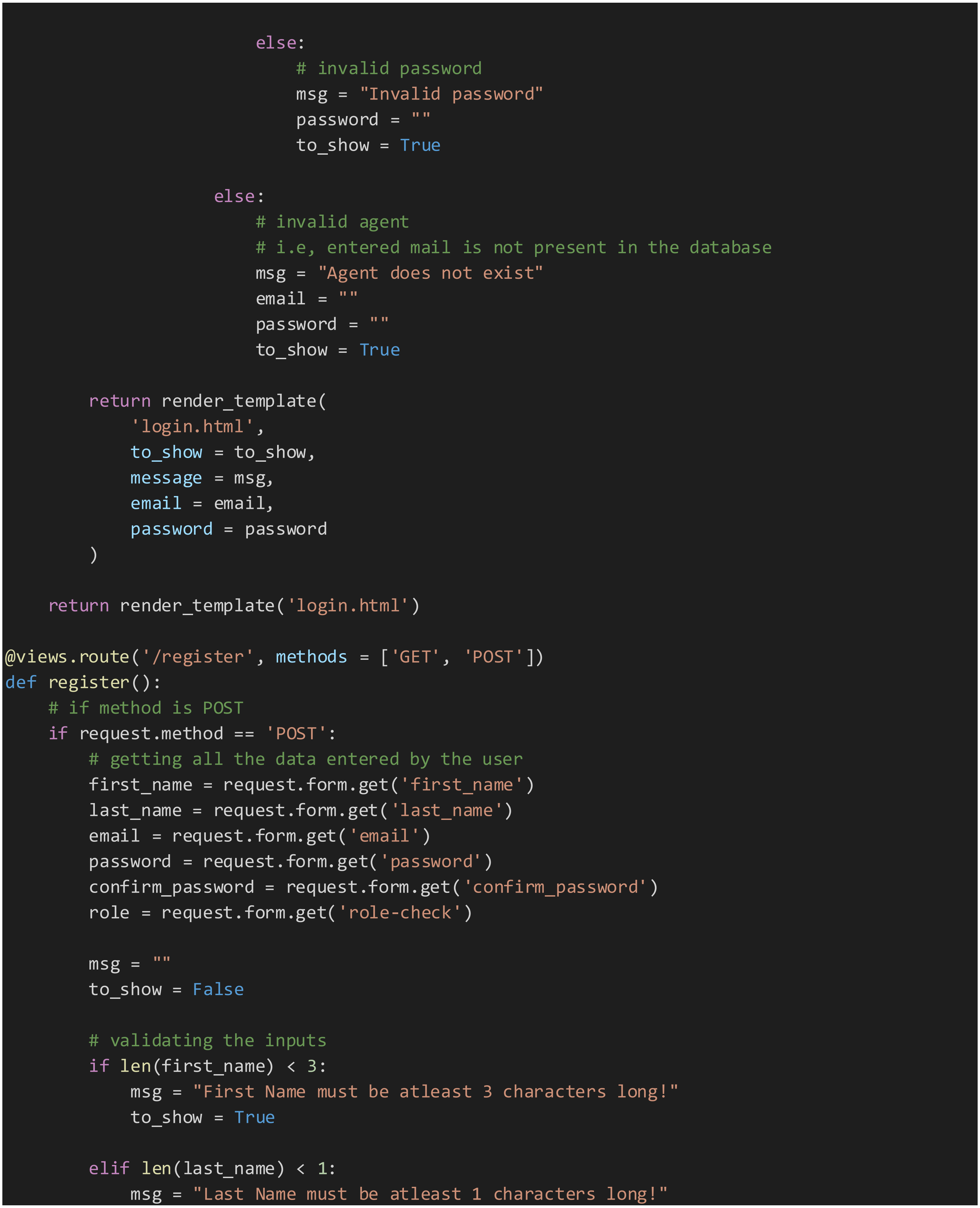
```
        msg = ""
        to_show = False

        # validating the inputs entered by the user
        if(not (re.fullmatch(email_regex, email))):
            msg = "Enter a valid email"
            to_show = True

        elif (len(password) < 8):
            msg = "Password must be atleast 8 characters long!"
            to_show = True

        # Admin login
        if email == "admin.ccr@gmail.com":
            if password == "admin.ccr@2022":
                # initialising admin object
                admin.set(email, password)

                session.permanent = False
                session['LOGGED_IN_AS'] = "ADMIN"
                login_user(admin, remember=True)

                return redirect(url_for('admin.tickets'))

            else:
                to_show = True
                password = ""
                msg = "Invalid password!"

        # Customer or Agent
        else:
            if to_show:
                # there is something fishy with the user's inputs
                password = ""

            elif (not to_show):
                # the user's inputs are valid
                # checking if the login credentials are valid
                if role == "Customer":
                    # checking if the entry of the mail entered is present in the database
                    mail_check_query = "SELECT * FROM customer WHERE email = ?"
                    stmt = ibm_db.prepare(conn, mail_check_query)
                    ibm_db.bind_param(stmt, 1, email)
                    ibm_db.execute(stmt)

                    account = ibm_db.fetch_assoc(stmt)
```



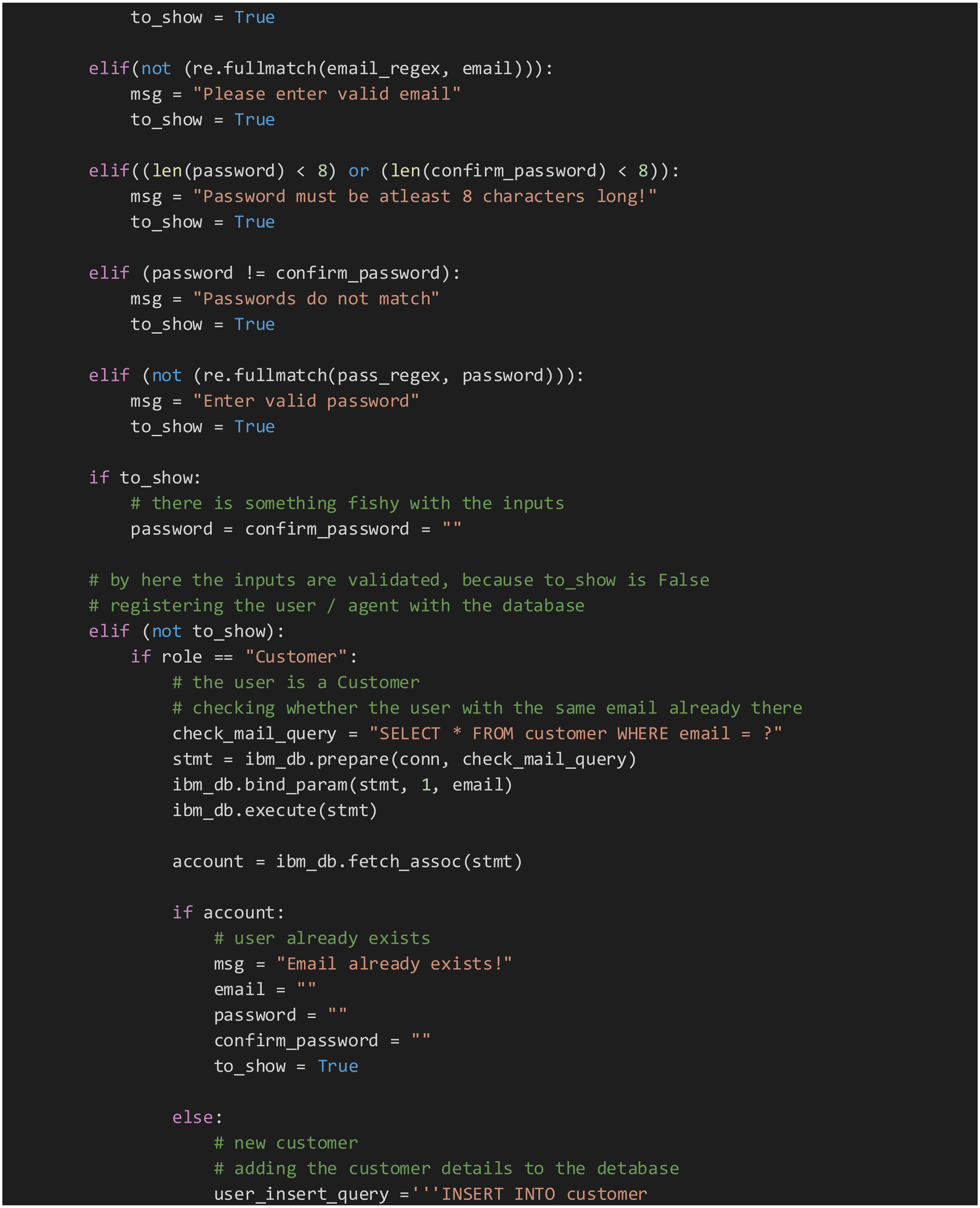
```
                    if account:
                        # valid customer
                        # i.e, mail is present in the database

                        # checking if the customer entered a valid password now
                        # encrypting the entered password
                        passcode = str(hashlib.sha256(password.encode()).hexdigest())

                        # now checking if the encrypted string is same as that of the one in
database

                        if (account['PASSCODE'] == passcode):
                            msg = "Valid Login"
                            to_show = True

                            # creating a customer object
                            customer.set(
                                account['CUST_ID'],
                                account['FIRST_NAME'],
                                account['LAST_NAME'],
                                account['EMAIL'],
                                account['PASSCODE'],
                                account['DATE_JOINED']
                            )

                            session.permanent = False
                            session['LOGGED_IN_AS'] = "CUSTOMER"
                            login_user(customer, remember=True)

                            return redirect(url_for('customer.profile'))

                        else:
                            # customer entered invalid password
                            msg = "Invalid password"
                            password = ""
                            to_show = True

                    else:
                        # invalid customer
                        # i.e, entered mail is not present in the database
                        msg = "User does not exist"
                        email = ""
                        password = ""
                        to_show = True

                else:
                    # user is an Agent
```



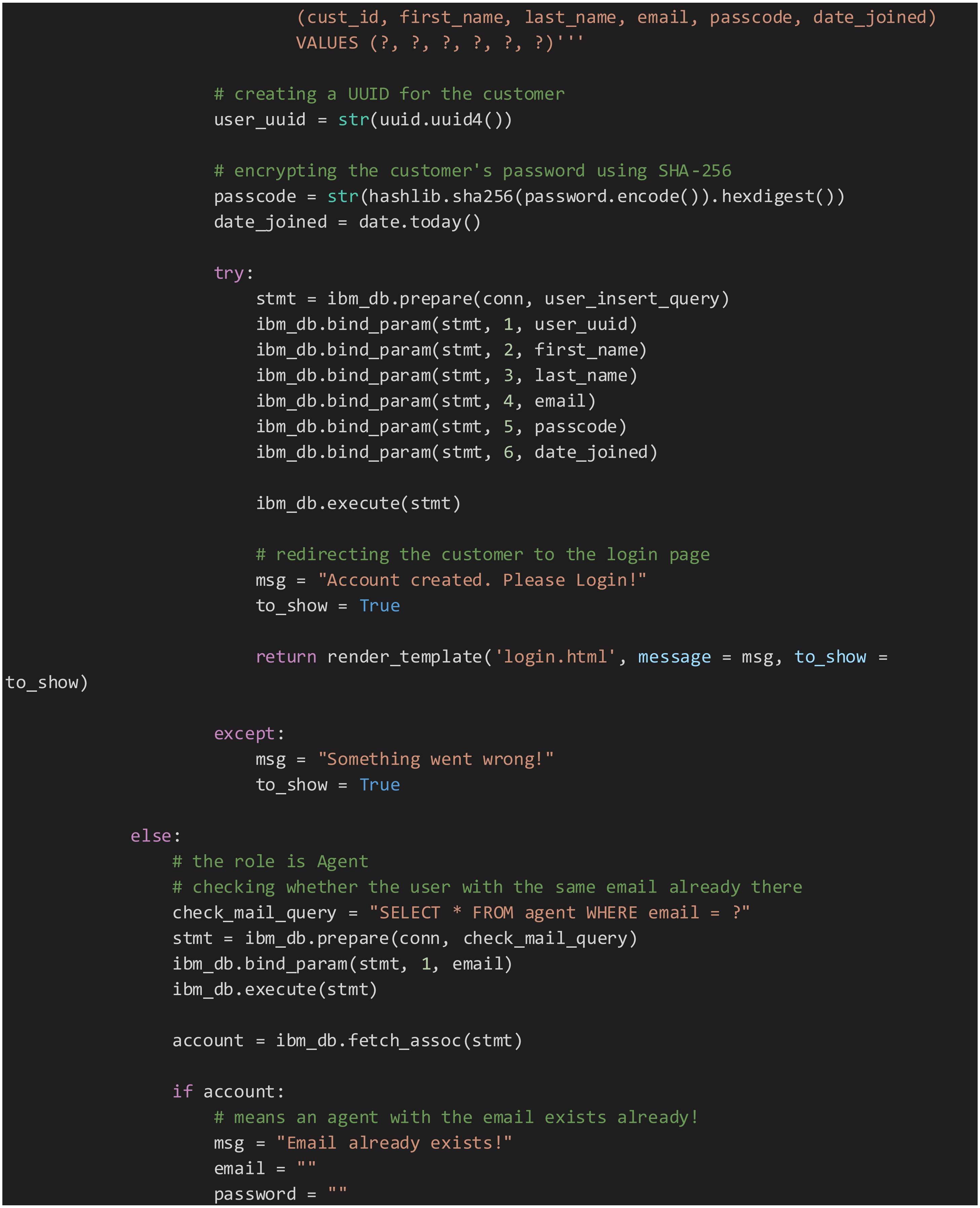
```python
                    # checking if the entry of the mail entered is present in the agent's table
                    mail_check_query = "SELECT * FROM agent WHERE email = ?"
                    stmt = ibm_db.prepare(conn, mail_check_query)
                    ibm_db.bind_param(stmt, 1, email)
                    ibm_db.execute(stmt)

                    account = ibm_db.fetch_assoc(stmt)

                    if account:
                        # the mail entered by the agent is in the database

                        # checking if the customer entered a valid password now
                        # encrypting the entered password
                        passcode = str(hashlib.sha256(password.encode()).hexdigest())

                        # now checking if this passcode is equal to that of the password in database
                        if(account['PASSCODE'] == passcode):
                            # valid password
                            msg = "Valid Login"
                            to_show = True

                            # initialising the agent object
                            agent.set(
                                account['AGENT_ID'],
                                account['FIRST_NAME'],
                                account['LAST_NAME'],
                                account['EMAIL'],
                                account['PASSCODE'],
                                account['DATE_JOINED'],
                                account['CONFIRMED']
                            )

                            session.permanent = False
                            session['LOGGED_IN_AS'] = "AGENT"
                            login_user(agent, remember=True)

                            if agent.confirm:
                                # the agent is confirmed by the admin
                                # so, re-directing the agent to his/her profile page
                                return redirect(url_for('agent.profile'))

                            else:
                                # the agent is not yet verified by the admin
                                # re-directing the agent to the agent no show page
                                return redirect(url_for('agent.no_show'))
```



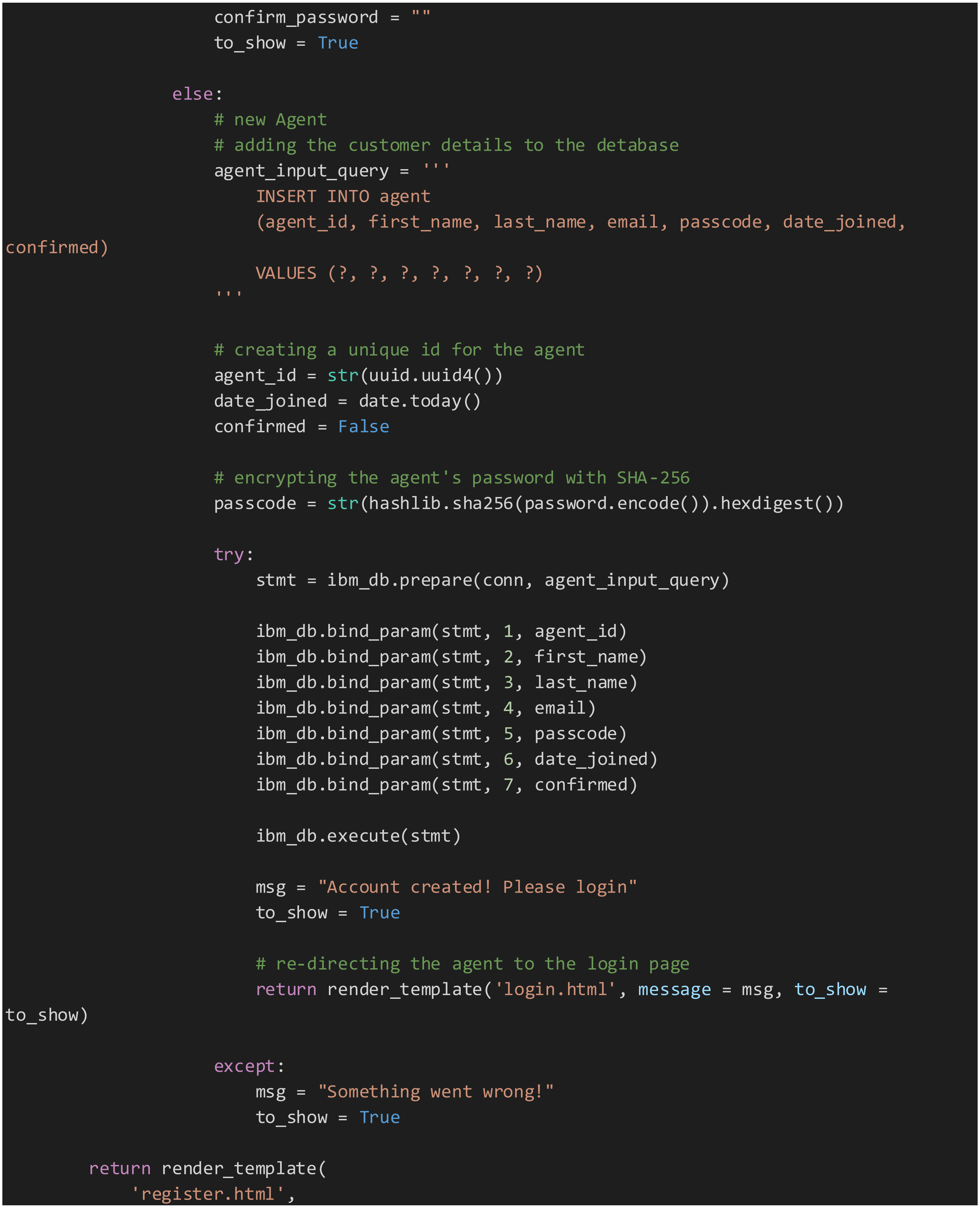
```
                        else:
                            # invalid password
                            msg = "Invalid password"
                            password = ""
                            to_show = True

                    else:
                        # invalid agent
                        # i.e, entered mail is not present in the database
                        msg = "Agent does not exist"
                        email = ""
                        password = ""
                        to_show = True

        return render_template(
            'login.html',
            to_show = to_show,
            message = msg,
            email = email,
            password = password
        )

    return render_template('login.html')

@views.route('/register', methods = ['GET', 'POST'])
def register():
    # if method is POST
    if request.method == 'POST':
        # getting all the data entered by the user
        first_name = request.form.get('first_name')
        last_name = request.form.get('last_name')
        email = request.form.get('email')
        password = request.form.get('password')
        confirm_password = request.form.get('confirm_password')
        role = request.form.get('role-check')

        msg = ""
        to_show = False

        # validating the inputs
        if len(first_name) < 3:
            msg = "First Name must be atleast 3 characters long!"
            to_show = True

        elif len(last_name) < 1:
            msg = "Last Name must be atleast 1 characters long!"
```



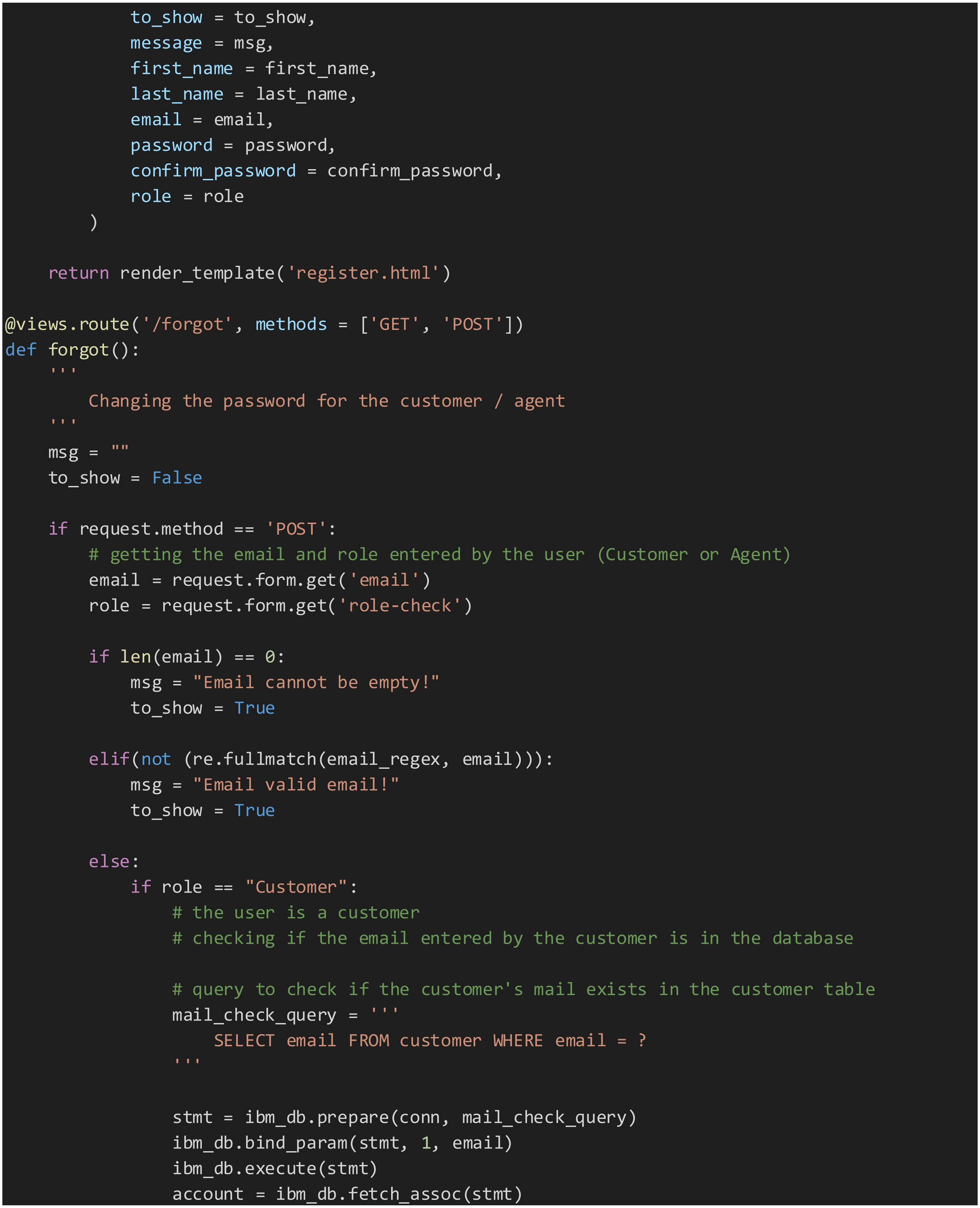
```
            to_show = True

        elif(not (re.fullmatch(email_regex, email))):
            msg = "Please enter valid email"
            to_show = True

        elif((len(password) < 8) or (len(confirm_password) < 8)):
            msg = "Password must be atleast 8 characters long!"
            to_show = True

        elif (password != confirm_password):
            msg = "Passwords do not match"
            to_show = True

        elif (not (re.fullmatch(pass_regex, password))):
            msg = "Enter valid password"
            to_show = True

        if to_show:
            # there is something fishy with the inputs
            password = confirm_password = ""

        # by here the inputs are validated, because to_show is False
        # registering the user / agent with the database
        elif (not to_show):
            if role == "Customer":
                # the user is a Customer
                # checking whether the user with the same email already there
                check_mail_query = "SELECT * FROM customer WHERE email = ?"
                stmt = ibm_db.prepare(conn, check_mail_query)
                ibm_db.bind_param(stmt, 1, email)
                ibm_db.execute(stmt)

                account = ibm_db.fetch_assoc(stmt)

                if account:
                    # user already exists
                    msg = "Email already exists!"
                    email = ""
                    password = ""
                    confirm_password = ""
                    to_show = True

                else:
                    # new customer
                    # adding the customer details to the detabase
                    user_insert_query ='''INSERT INTO customer
```



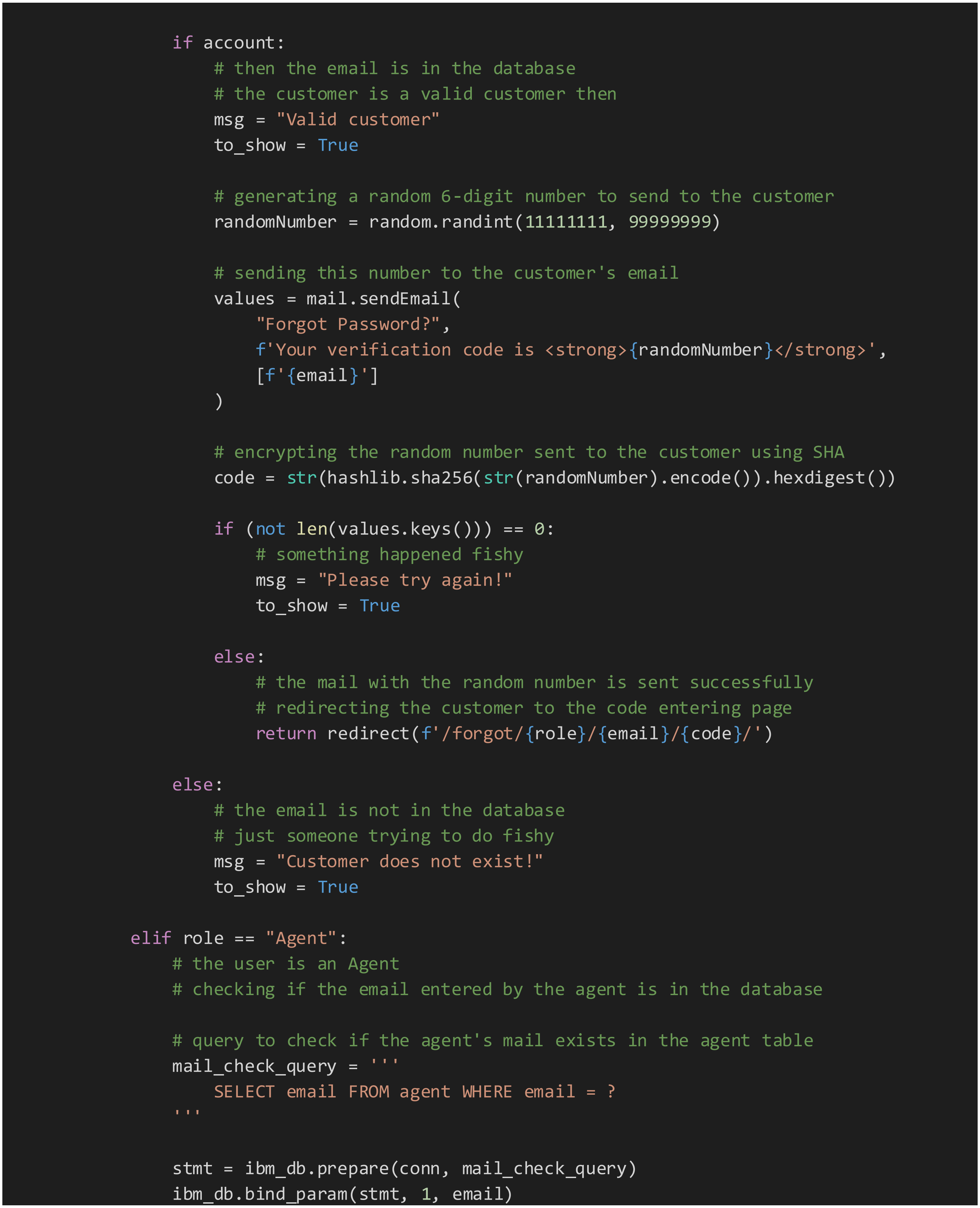
```
                            (cust_id, first_name, last_name, email, passcode, date_joined)
                            VALUES (?, ?, ?, ?, ?, ?)'''

                    # creating a UUID for the customer
                    user_uuid = str(uuid.uuid4())

                    # encrypting the customer's password using SHA-256
                    passcode = str(hashlib.sha256(password.encode()).hexdigest())
                    date_joined = date.today()

                    try:
                        stmt = ibm_db.prepare(conn, user_insert_query)
                        ibm_db.bind_param(stmt, 1, user_uuid)
                        ibm_db.bind_param(stmt, 2, first_name)
                        ibm_db.bind_param(stmt, 3, last_name)
                        ibm_db.bind_param(stmt, 4, email)
                        ibm_db.bind_param(stmt, 5, passcode)
                        ibm_db.bind_param(stmt, 6, date_joined)

                        ibm_db.execute(stmt)

                        # redirecting the customer to the login page
                        msg = "Account created. Please Login!"
                        to_show = True

                        return render_template('login.html', message = msg, to_show =
to_show)

                    except:
                        msg = "Something went wrong!"
                        to_show = True

            else:
                # the role is Agent
                # checking whether the user with the same email already there
                check_mail_query = "SELECT * FROM agent WHERE email = ?"
                stmt = ibm_db.prepare(conn, check_mail_query)
                ibm_db.bind_param(stmt, 1, email)
                ibm_db.execute(stmt)

                account = ibm_db.fetch_assoc(stmt)

                if account:
                    # means an agent with the email exists already!
                    msg = "Email already exists!"
                    email = ""
                    password = ""
```



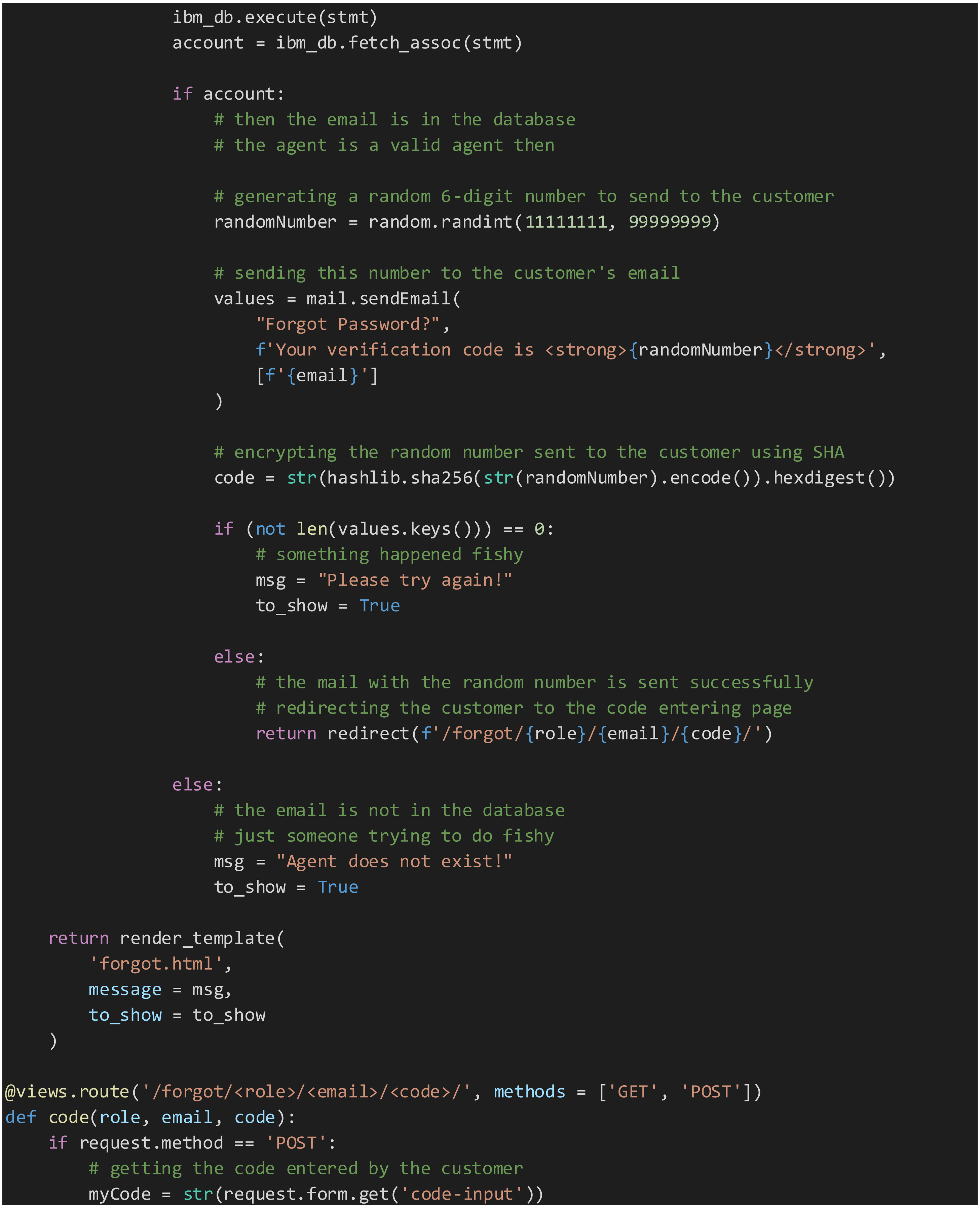
```
                    confirm_password = ""
                    to_show = True

                else:
                    # new Agent
                    # adding the customer details to the detabase
                    agent_input_query = '''
                        INSERT INTO agent
                        (agent_id, first_name, last_name, email, passcode, date_joined,
confirmed)
                        VALUES (?, ?, ?, ?, ?, ?, ?)
                    '''

                    # creating a unique id for the agent
                    agent_id = str(uuid.uuid4())
                    date_joined = date.today()
                    confirmed = False

                    # encrypting the agent's password with SHA-256
                    passcode = str(hashlib.sha256(password.encode()).hexdigest())

                    try:
                        stmt = ibm_db.prepare(conn, agent_input_query)

                        ibm_db.bind_param(stmt, 1, agent_id)
                        ibm_db.bind_param(stmt, 2, first_name)
                        ibm_db.bind_param(stmt, 3, last_name)
                        ibm_db.bind_param(stmt, 4, email)
                        ibm_db.bind_param(stmt, 5, passcode)
                        ibm_db.bind_param(stmt, 6, date_joined)
                        ibm_db.bind_param(stmt, 7, confirmed)

                        ibm_db.execute(stmt)

                        msg = "Account created! Please login"
                        to_show = True

                        # re-directing the agent to the login page
                        return render_template('login.html', message = msg, to_show =
to_show)

                    except:
                        msg = "Something went wrong!"
                        to_show = True

        return render_template(
            'register.html',
```



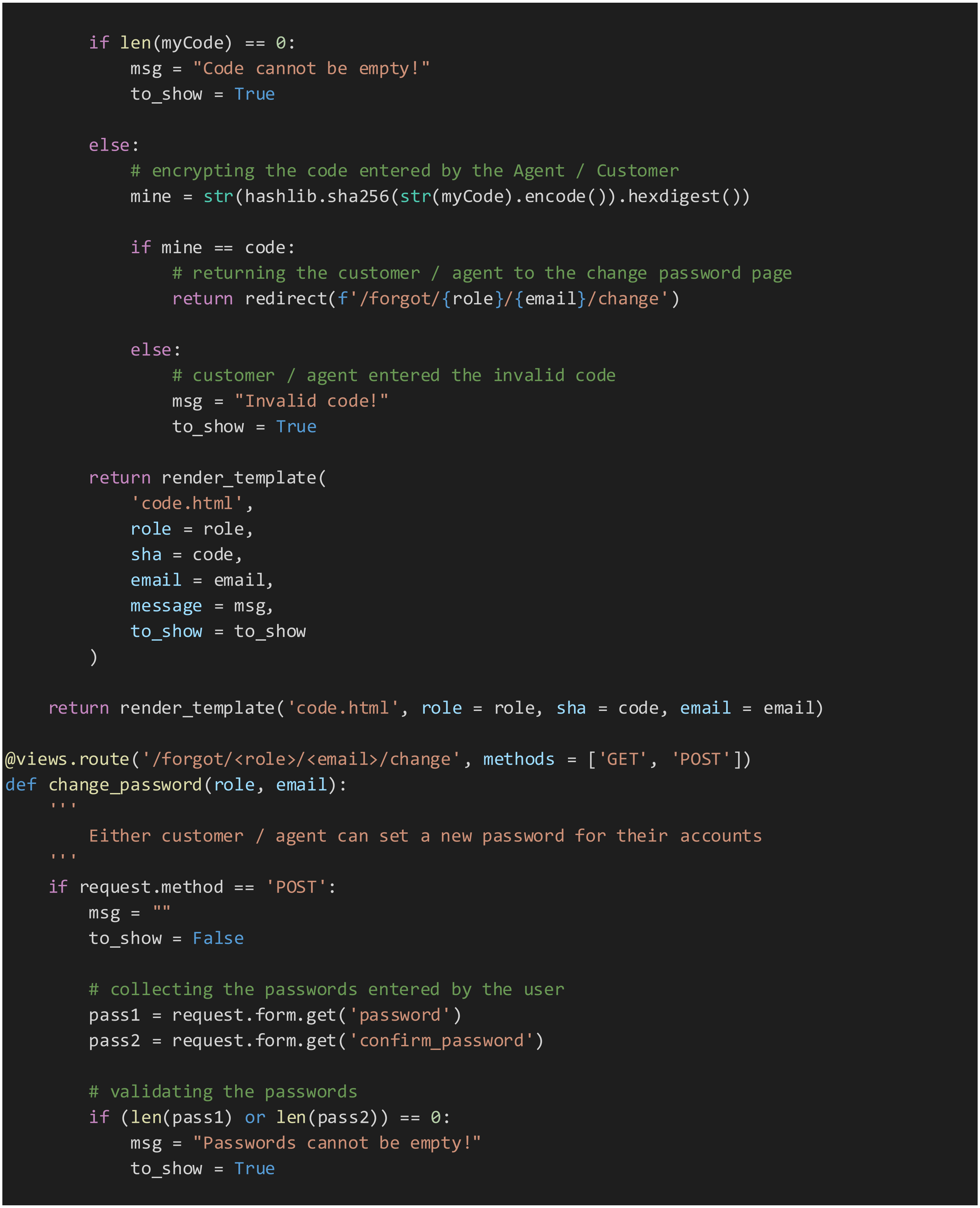
```
            to_show = to_show,
            message = msg,
            first_name = first_name,
            last_name = last_name,
            email = email,
            password = password,
            confirm_password = confirm_password,
            role = role
        )

    return render_template('register.html')

@views.route('/forgot', methods = ['GET', 'POST'])
def forgot():
    '''
        Changing the password for the customer / agent
    '''
    msg = ""
    to_show = False

    if request.method == 'POST':
        # getting the email and role entered by the user (Customer or Agent)
        email = request.form.get('email')
        role = request.form.get('role-check')

        if len(email) == 0:
            msg = "Email cannot be empty!"
            to_show = True

        elif(not (re.fullmatch(email_regex, email))):
            msg = "Email valid email!"
            to_show = True

        else:
            if role == "Customer":
                # the user is a customer
                # checking if the email entered by the customer is in the database

                # query to check if the customer's mail exists in the customer table
                mail_check_query = '''
                    SELECT email FROM customer WHERE email = ?
                '''

                stmt = ibm_db.prepare(conn, mail_check_query)
                ibm_db.bind_param(stmt, 1, email)
                ibm_db.execute(stmt)
                account = ibm_db.fetch_assoc(stmt)
```



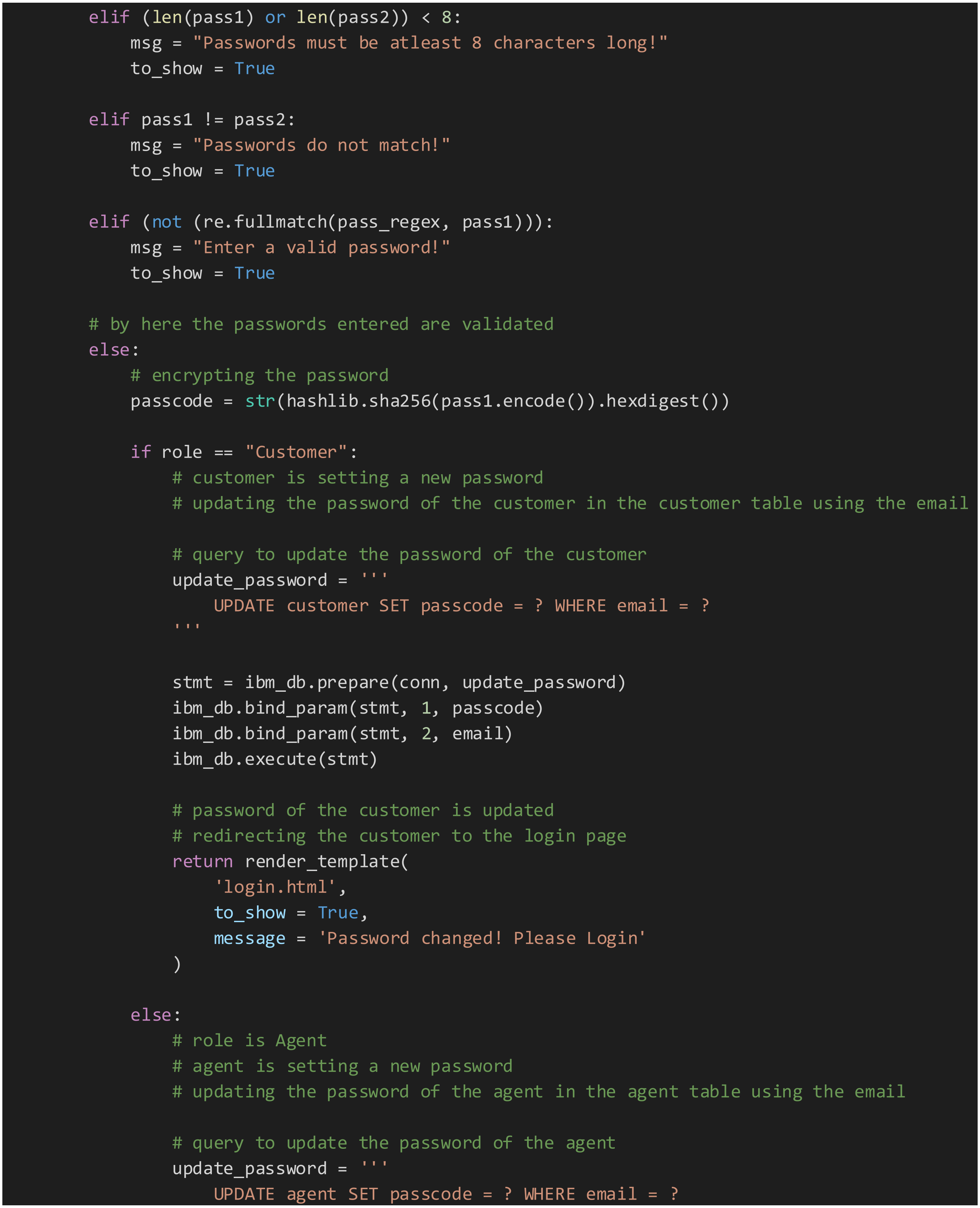
```
            if account:
                # then the email is in the database
                # the customer is a valid customer then
                msg = "Valid customer"
                to_show = True

                # generating a random 6-digit number to send to the customer
                randomNumber = random.randint(11111111, 99999999)

                # sending this number to the customer's email
                values = mail.sendEmail(
                    "Forgot Password?",
                    f'Your verification code is <strong>{randomNumber}</strong>',
                    [f'{email}']
                )

                # encrypting the random number sent to the customer using SHA
                code = str(hashlib.sha256(str(randomNumber).encode()).hexdigest())

                if (not len(values.keys())) == 0:
                    # something happened fishy
                    msg = "Please try again!"
                    to_show = True

                else:
                    # the mail with the random number is sent successfully
                    # redirecting the customer to the code entering page
                    return redirect(f'/forgot/{role}/{email}/{code}/')

            else:
                # the email is not in the database
                # just someone trying to do fishy
                msg = "Customer does not exist!"
                to_show = True

        elif role == "Agent":
            # the user is an Agent
            # checking if the email entered by the agent is in the database

            # query to check if the agent's mail exists in the agent table
            mail_check_query = '''
                SELECT email FROM agent WHERE email = ?
            '''

            stmt = ibm_db.prepare(conn, mail_check_query)
            ibm_db.bind_param(stmt, 1, email)
```



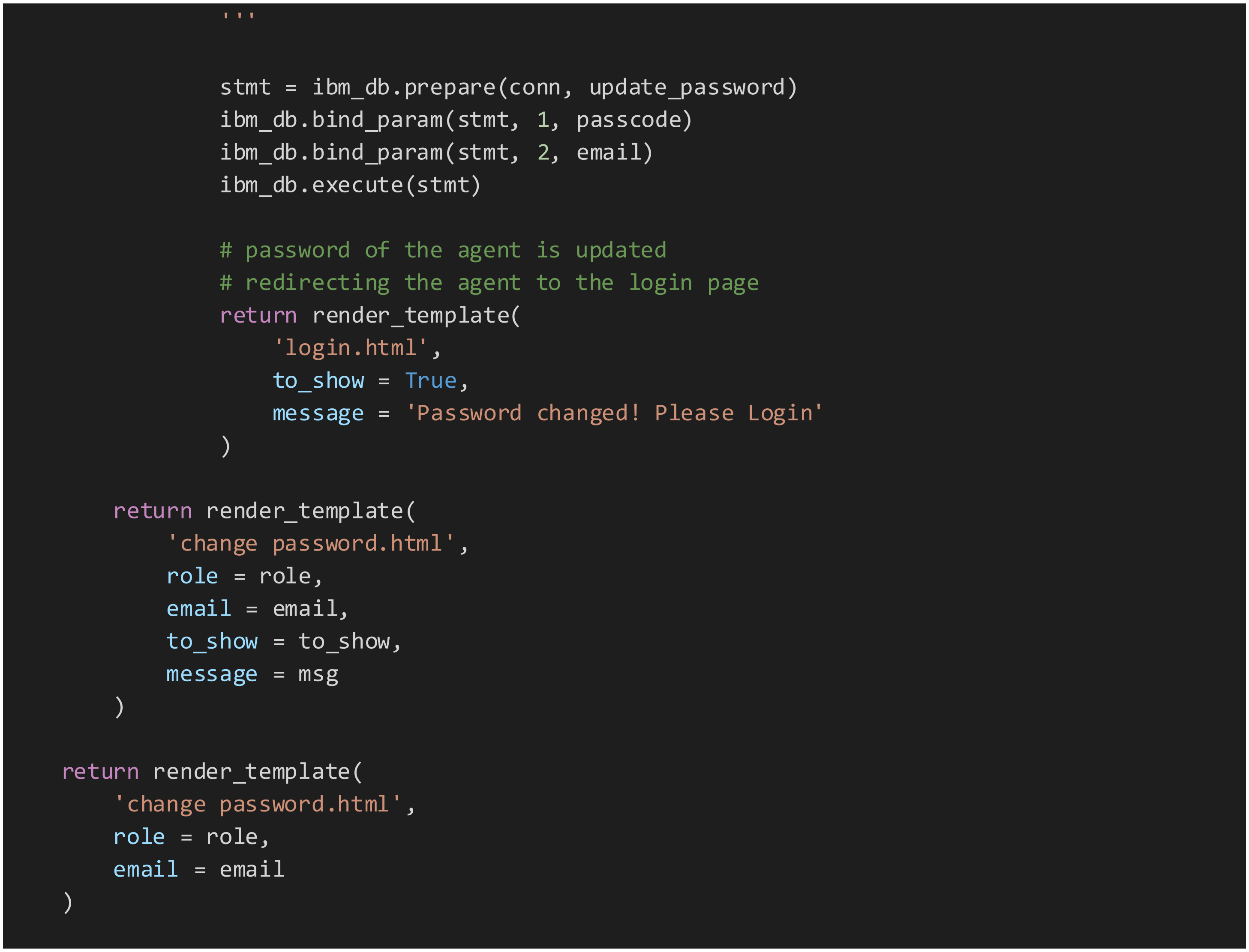
```
                ibm_db.execute(stmt)
                account = ibm_db.fetch_assoc(stmt)

                if account:
                    # then the email is in the database
                    # the agent is a valid agent then

                    # generating a random 6-digit number to send to the customer
                    randomNumber = random.randint(11111111, 99999999)

                    # sending this number to the customer's email
                    values = mail.sendEmail(
                        "Forgot Password?",
                        f'Your verification code is <strong>{randomNumber}</strong>',
                        [f'{email}']
                    )

                    # encrypting the random number sent to the customer using SHA
                    code = str(hashlib.sha256(str(randomNumber).encode()).hexdigest())

                    if (not len(values.keys())) == 0:
                        # something happened fishy
                        msg = "Please try again!"
                        to_show = True

                    else:
                        # the mail with the random number is sent successfully
                        # redirecting the customer to the code entering page
                        return redirect(f'/forgot/{role}/{email}/{code}/')

                else:
                    # the email is not in the database
                    # just someone trying to do fishy
                    msg = "Agent does not exist!"
                    to_show = True

    return render_template(
        'forgot.html',
        message = msg,
        to_show = to_show
    )

@views.route('/forgot/<role>/<email>/<code>/', methods = ['GET', 'POST'])
def code(role, email, code):
    if request.method == 'POST':
        # getting the code entered by the customer
        myCode = str(request.form.get('code-input'))
```



```python
        if len(myCode) == 0:
            msg = "Code cannot be empty!"
            to_show = True

        else:
            # encrypting the code entered by the Agent / Customer
            mine = str(hashlib.sha256(str(myCode).encode()).hexdigest())

            if mine == code:
                # returning the customer / agent to the change password page
                return redirect(f'/forgot/{role}/{email}/change')

            else:
                # customer / agent entered the invalid code
                msg = "Invalid code!"
                to_show = True

        return render_template(
            'code.html',
            role = role,
            sha = code,
            email = email,
            message = msg,
            to_show = to_show
        )

    return render_template('code.html', role = role, sha = code, email = email)

@views.route('/forgot/<role>/<email>/change', methods = ['GET', 'POST'])
def change_password(role, email):
    '''
        Either customer / agent can set a new password for their accounts
    '''
    if request.method == 'POST':
        msg = ""
        to_show = False

        # collecting the passwords entered by the user
        pass1 = request.form.get('password')
        pass2 = request.form.get('confirm_password')

        # validating the passwords
        if (len(pass1) or len(pass2)) == 0:
            msg = "Passwords cannot be empty!"
            to_show = True
```



```
        elif (len(pass1) or len(pass2)) < 8:
            msg = "Passwords must be atleast 8 characters long!"
            to_show = True

        elif pass1 != pass2:
            msg = "Passwords do not match!"
            to_show = True

        elif (not (re.fullmatch(pass_regex, pass1))):
            msg = "Enter a valid password!"
            to_show = True

        # by here the passwords entered are validated
        else:
            # encrypting the password
            passcode = str(hashlib.sha256(pass1.encode()).hexdigest())

            if role == "Customer":
                # customer is setting a new password
                # updating the password of the customer in the customer table using the email

                # query to update the password of the customer
                update_password = '''
                    UPDATE customer SET passcode = ? WHERE email = ?
                '''

                stmt = ibm_db.prepare(conn, update_password)
                ibm_db.bind_param(stmt, 1, passcode)
                ibm_db.bind_param(stmt, 2, email)
                ibm_db.execute(stmt)

                # password of the customer is updated
                # redirecting the customer to the login page
                return render_template(
                    'login.html',
                    to_show = True,
                    message = 'Password changed! Please Login'
                )

            else:
                # role is Agent
                # agent is setting a new password
                # updating the password of the agent in the agent table using the email

                # query to update the password of the agent
                update_password = '''
                    UPDATE agent SET passcode = ? WHERE email = ?
```



```
                '''

                stmt = ibm_db.prepare(conn, update_password)
                ibm_db.bind_param(stmt, 1, passcode)
                ibm_db.bind_param(stmt, 2, email)
                ibm_db.execute(stmt)

                # password of the agent is updated
                # redirecting the agent to the login page
                return render_template(
                    'login.html',
                    to_show = True,
                    message = 'Password changed! Please Login'
                )

        return render_template(
            'change password.html',
            role = role,
            email = email,
            to_show = to_show,
            message = msg
        )

    return render_template(
        'change password.html',
        role = role,
        email = email
    )
```

13.2. GITHUB AND PROJECT DEMO LINK:

Github link : click here

Project Demo link : click here