DATE :20 NOVEMBER 2022

TEAM ID:PN72022MID42506

PROJECT NAME:CUSTOMER CARE REGISTRY

TABLE OF CONTENT

1.INTRODUCTION

1.1.PROJECT OVERVIEW:

The “Help Desk Ticketing System” has been developed to override the problems prevalling in the practicing manual system. This software is supported to elimination and in some cases reduce the hardships faced by this existing system. Moreover this system is designed for the particular need of the company to carry out operations in a smooth and effective manner.

The application is reduced as much as possible to avoid errors while entering the data.it also provides error message while entering invalid data.no formal knowledge is needed for the user to use this system. Thus by this, all it proves it is user-friendly .Help Desk Ticketing System, as described above, can lead to error free, secure ,reliable and fast management system. It can as it the user to concentrate on their other activities rather to concentrate on the record keeping. Thus it will help organization in better utilization of resources.

Every organization, whether big or small, has challenges to overcome and managing the information of issues, help disk, email, customer, network. Every Help Desk Ticketing System has different Help Desk needs, therefore we design exclusive employee management system that are adapted to your managerial requirements

1.2.PURPOSE:

The purpose of Help Desk Ticketing system is to automate the exiting manual system by the help of computerized requirements and full-fledged computer software fulfilling their requirements , so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. The required software and hardware are easily available and easy to work with.

Help Desk Ticketing System, as described above, can lead to error free, secure, reliable and fast management. It can assist the user to concentrate on their other activities rather to concentrate on the record keeping. Thus it will help organization in better utilization of resources .The organization can maintain computerized records without redundant entries. That means that one need not be distracted by information that is not relevant, while being able to reach the information.

The aim is to automata its exiting manual system by the help of computerized equipments and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same .Basically the project describes how to manage for good performances and better services for the clients.

2.LITERATURE SURVEY

2.1.EXISTING PROBLEM:

EXISTING SYSTEM OF HELP DESK TICKETING SYSTEM:

In the existing system the exams are done only manually but in proposed system we have to computerize the exams using this application.

- Lack of security of data.
- More man power.
- Time consuming.
- Consumes large volume of pare work.
- Needs manual calculation.
- No direct role for the higher officials.

2.2. REFERENCES:

Agiloft. 2021. Agiloft Features | Software Features. [online] Available at: <https://www.agiloft.com/resources/features/> [Accessed 13 November 2021].

Brame, D., 2019. Freshdesk Review. [online] PCMAG. Available at: <https://www.pcmag.com/reviews/freshdesk> [Accessed 8 November 2021].

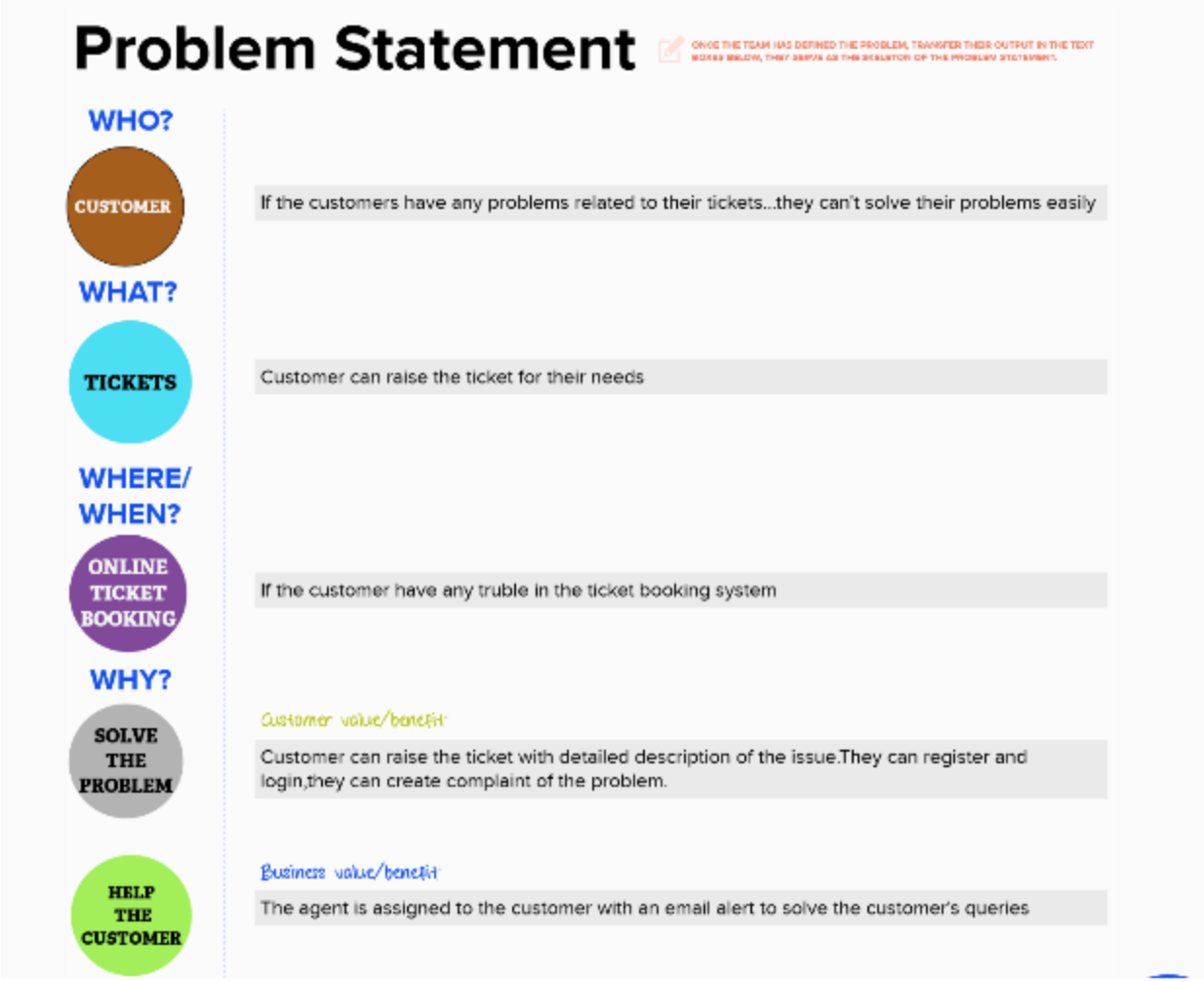
Gorgias.com. 2021. Customer Service Made Easy for Online Stores | Gorgias. [online] Available at: <https://www.gorgias.com/> [Accessed 13 November 2021].

Zendesk. 2021. Customer service software for the best customer experiences | Zendesk. [online] Available at: <https://www.zendesk.com/service/> [Accessed 13 November 2021].

Ferrill, P., 2021a. Zoho Desk Review. [online] PCMAG. Available at: <https://www.pcmag.com/reviews/zoho-desk> [Accessed 8 November 2021].

software. [online] Available at: <https://freshservice.com/pricing> [Accessed 13 November 2021].

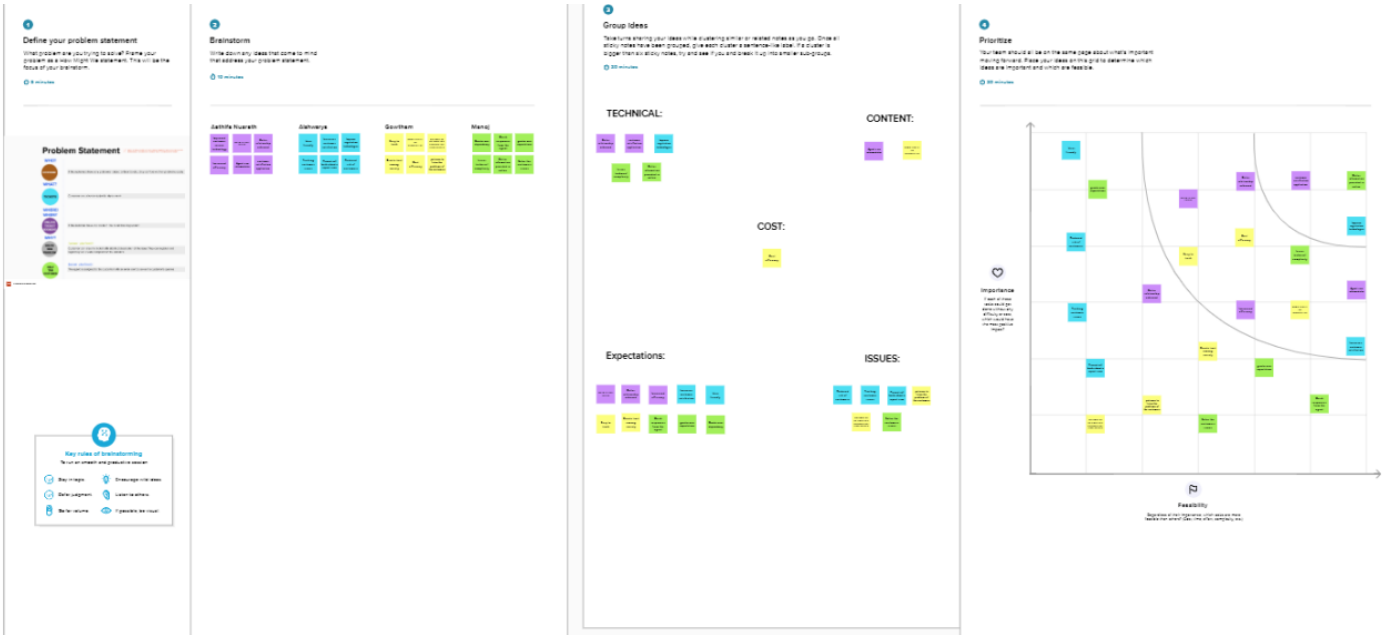
2.3.PROBLEM STATEMENT DEFINITION:



3.IDEATION AND PROPOSED SOLUTION

3.1.EMPATHY MAP CANVAS:

3.2.IDEATION & BRAINSTORMING:



3.3.PROPOSED SOLUTION:

S.NO.	PARAMETERS	DESCRIPTION
1.	Problem statement (Problem to be solved)	1. There is no accountability on the part of the agent if response times have been really prolonged. 2. If the customer finds it difficult to explain the issues due to a lack of knowledge of relent technical terms. 3. When a customer keeps getting transferred from one agent or department to another.It ensures that will never return to you or your business in the future.Neither will they recommend you to people they know this brings us to the

		second most common customer service problem.
2.	Idea / Solution description	1. The help desk generally has to cover a wide range of information technology products and services. 2. The customers can raise the ticket with a detailed description of the issues. 3. User can register for an account. After the login, they can create the complaint with a detailed description of the problem they are facing. Each user will be assigned with an agent. They can view the status of their complaint.

3.	Uniqueness	1. Whenever the agent is assigned to a customer they will be notified with an email alert. 2. Customers can view the status of the ticket till the service is provided. 3. Admin will be able to track the work assigned to the agent and a notification will be sent to the customer.
4.	Customer satisfication	1. This application help the customer in processing their complaints. 2. This application help the customer with the support of agent. 3. This application act as an user friendly.

5.	Business model	1. Our proposed system will be a helpdesk application which leads to customer satisfication. 2. An agent is assigned to customer to solve the problem with an email alert. 3. User can register for an account. After the login, they can create the complaint with a detailed description of the problem they are facing. Each user will be assigned with an agent. They can view the status of their complaint.
6.	Scalability of solution	1. The proposed application is more convenient to use in both android and IOS based systems. 2. The user can easily get solved the problems by our applications. 3. Easily can access from our devices.

3.4.PROBLEM SOLUTION FIT:

Define CS, fit into CC

1.CUSTOMER SEGMENT(S) CS

Who is your customer?

- The person who is booking the ticket.
- Person who can get solved their problem by their agent.

6. CUSTOMER CONSTRAINTS CC

What constraints prevent your customers from taking action or limit their choices of solutions?

- Working as a customer service respresentative requires you to maintain a friendly

5. AVAILABLE SOLUTIONS AS

solutions are availablet the customer when they face the problem or need to get the job done?what have they in the past?what pros &cons do these solutions have?

Pros:

- Customer issues can be easily solved by their assigned agent.

Cons:

- Delivering false information.

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Which problem do you solve for your customer?
There could be more than one,explore different sides.

- Issues related to the ticket.
- Network and server issues.

9. PROBLEM ROOT CAUSE RC

What is the real reason that this problem exists? What is the back story behind the need to do this job?

- Server down problem.
- Network issues.

7.BEHAVIOR BE

What does your customer do about/around/directly or indirectly related to the problem?

- Use qualitative and quantitative methods like focus group,surveys and customer behaviour data.

Focus on J&P, tap into BE, understand RC

3. TRIGGERS TR

What triggers customers to act?

- It is user friendly.
- Network and server issues.

10. YOUR SOLUTION SL

Every customer is assigned with their agent to solve their problem by email notification.

8. CHANNELS of BEHAVIOUR CH

8.1ONLINE:

- Online ticket booking system.
- Customer get the email notification from the agent.

4.REQUIREMENT ANALYSIS

4.1.FUNCTIONAL REQUIREMENT:

FR NO.	FUNCTION REQUIREMENTS (EPIC)	SUB REQUIREMENTS (STORY/SUB-TASK)
FR-1	User Registration	Registration done by the customer through website or our application form.
FR-2	User confirmation	Email alert to the customer by the admin to know their assigned agent.
FR-3	User description	After the login,they can create the complaint with a description of the problem they are facing.
FR-4	User satisfaction	Their assigned agent solved their customer problem.
FR-5	User website	The customer can view their status of the complaint.

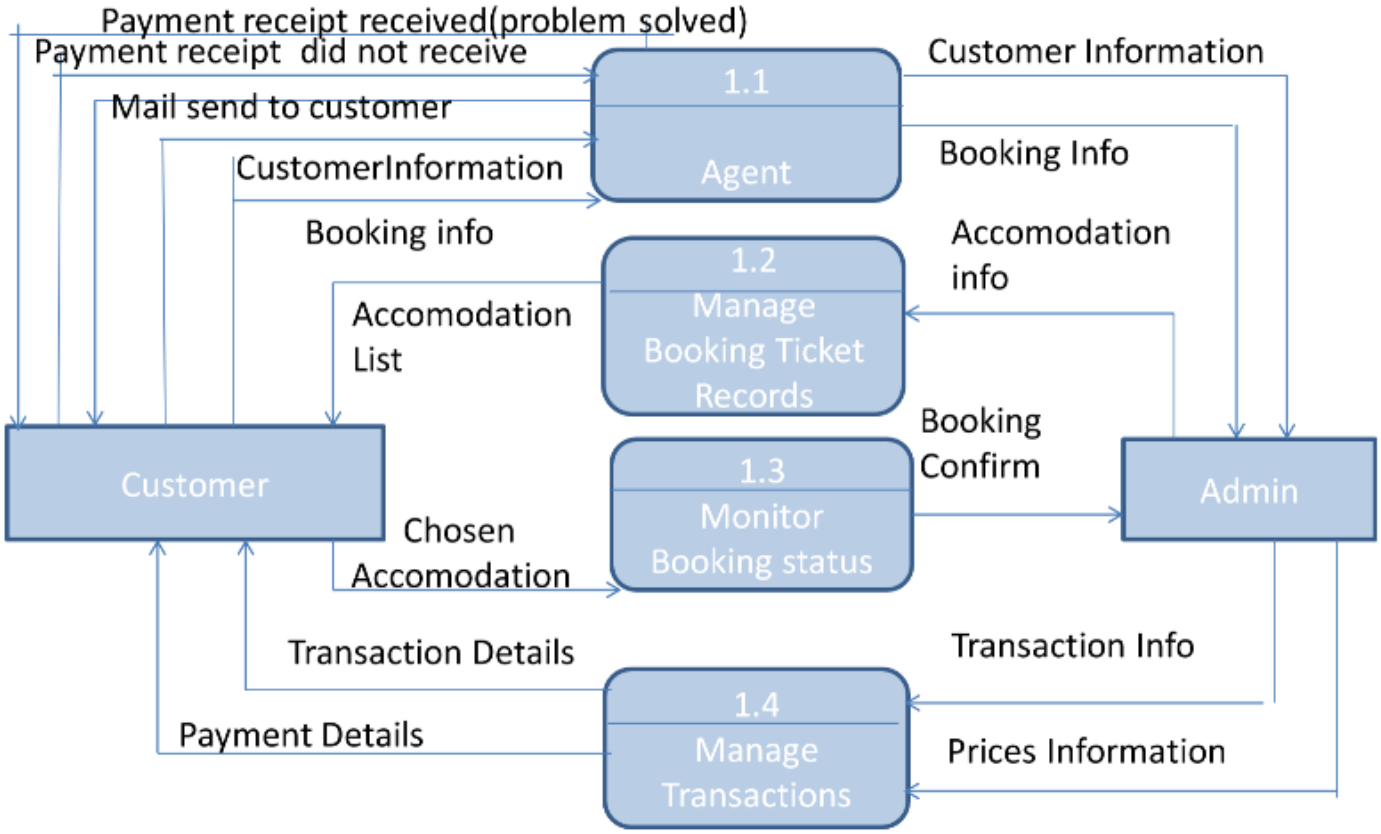
4.2.NON-FUNCTIONAL REQUIREMENTS:

FR NO.	NON-FUNCTION REQUIREMENT	DESCRIPTION
NFR-1	Usability	➢ An user friendly and simple customer support application. ➢ User can easily booking their tickets and their complaints can be also easily solved.

NFR-2	Security	➢ Secure customer support application. ➢ User can create an account with their unique username and password.
NFR-3	Reliability	➢ Defect free. ➢ The website load time is not more than second for users.
NFR-4	Performance	➢ Fast and quick response from their customer agent. ➢ Easy to access with any local devices.
NFR-5	Availability	➢ Anytime anywhere available web application almost can found in all popular search engines like google, etc… ➢ User are requested to have good internet connection.
NFR-6	Scalability	➢ More than one of many users can access use this customer support application. ➢ Reduced traffic in case of multiple user interaction .

5.PROJECT DESIGN

5.1.DATA FLOW DIAGRAMS:



5.2.SOLUTION & TECHNICAL ARCHITECTURE:

TECHNICAL ARCHITECTURE:

Table-1 : Components & Technologies:

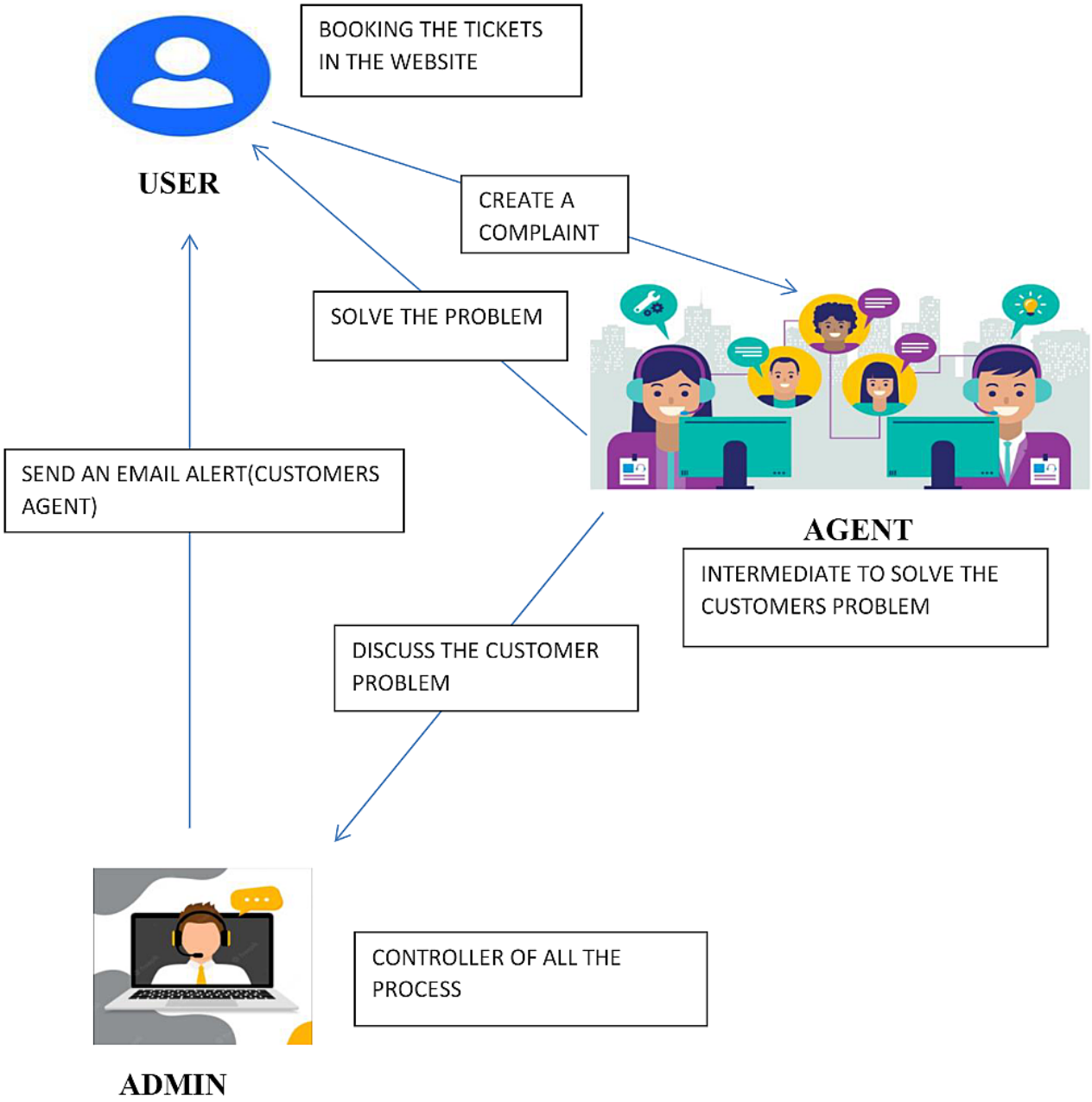
S.No	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI, Mobile App, Chatbot etc.	HTML, CSS, JavaScript / Angular Js / React Js etc.

2.	Application Logic-1	Logic for a process in the application	Java / Python
3.	Application Logic-2	Logic for a process in the application	IBM Watson STT service
4.	Application Logic-3	Logic for a process in the application	IBM Watson Assistant
5.	Database	Data Type, Configurations etc.	MySQL, NoSQL, etc.
6.	Cloud Database	Database Service on Cloud	IBM DB2, IBM Cloudant etc.
7.	File Storage	File storage requirements	IBM Block Storage or Other Storage Service or Local Filesystem
8.	External API-1	Purpose of External API used in the application	IBM Weather API, etc.
9.	External API-2	Purpose of External API used in the application	Aadhar API, etc.
10.	Machine Learning Model	Purpose of Machine Learning Model	Object Recognition Model, etc.
11.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Server Configuration: Cloud Server Configuration :	Local, Cloud Foundry, Kubernetes, etc.

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	Technology of Opensource framework
2.	Security Implementations	List all the security / access controls implemented, use of firewalls etc.	e.g. SHA-256, Encryptions, IAM Controls, OWASP etc.
3.	Scalable Architecture	Justify the scalability of architecture (3 – tier, Micro-services)	Technology used
4.	Availability	Justify the availability of application (e.g. use of load balancers, distributed servers etc.)	Technology used
5.	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	Technology used

SOLUTION ARCHITECTURE:



5.3.USER STORIES:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I c;an access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard	USN-6	As a user ,I can access our profile,booking details,assigned agent details	I can access our profile,booking details,assigned agent details	High	Sprint-1
Customer (Web user)	Registration	USN-7	As a user, I can register for the application by entering my email, password, and confirming my password	I can register for the application by entering my email, password, and confirming my password	High	Sprint-1
Customer Care Executive	Satisfaction	USN-8	As a agent,I can solve my customers problems	I can solve my customers problems	High	Sprint-3
Administrator	Customer support	USN-9	AS a admin,I can access all the process in our system	I can access all the process in our system	High	Sprint-4

6.PROJECT PLANNING & SCHEDULING

6.1.SPRINT PLANNING & ESTIMATION:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Aishwarya,Gowtham.
Sprint-1		USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Manoj,Aathifa Nusrath..
Sprint-2		USN-3	As a user, I can register for the application through Facebook	2	Low	Aathifa Nusrath,Aishwarya.
Sprint-1		USN-4	As a user, I can register for the application through Gmail	2	Medium	Gowtham,Manoj.
Sprint-1	Login	USN-5	As a user, I can log into the application by entering email & password	1	High	Aathifa Nusrath,Gowtham.
Sprint-1	Dashboard	USN-6	As a user,I can access our profile,booking,details,assigned agent details	1	High	Aishwarya,Manoj.
Sprint-3	Satisfaction	USN-7	As a user,I can register for the application by entering my email,password,and confirming my password	1	High	Aathifa Nusrath,Aishwarya.
Sprint-4	Customer support	USN-8	As a agent,I can solve my customers problems	1	High	Gowtham,Manoj.

6.2.SPRINT DELIVERY SCHEDULE:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	5 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

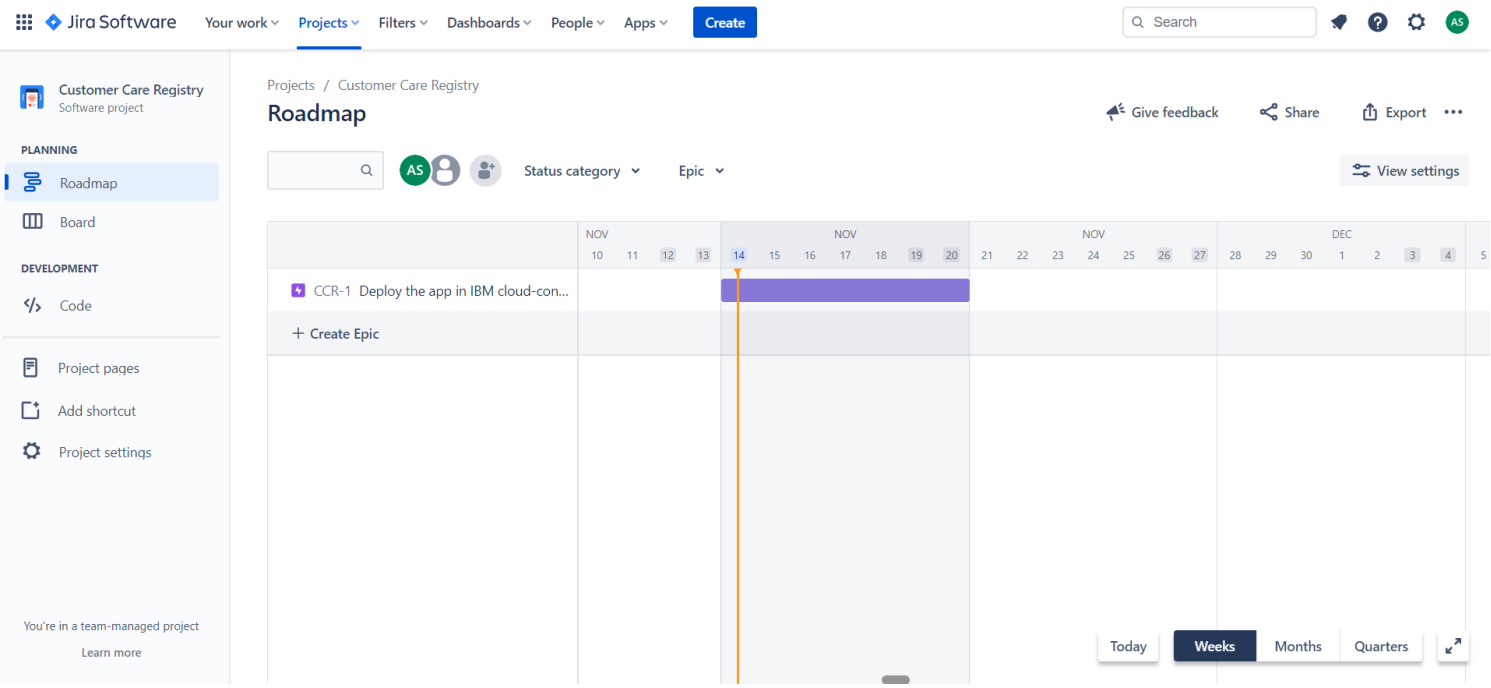
6.3.REPORTS FROM JIRA:

SPRINT 1:

SPRINT 2:

SPRINT 3:

SPRINT 4:


Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
Customer Care Registry
Software project
PLANNING
Roadmap
Board
DEVELOPMENT
Code
Project pages
Add shortcut
Project settings
You're in a team-managed project
Learn more
Projects / Customer Care Registry
Roadmap
Give feedback
Share
Export
Status category
Epic
View settings
NOV
DEC
CCR-1 Deploy the app in IBM cloud-con...
Create Epic
Today
Weeks
Months
Quarters

7.CODING AND SOLUTIONING

7.1.FEATURE 1:

CODING:

ADMIN REGISTER.HTML:

```
{% extends ‘base.html’ %}
{% block body %}
<form method=”POST” class=”register-form”>
<div class=”container”>
<h2>Administrator Sign-Up</h2>
<div class=”mb-3”>
<label for=”email-address” class=”form-label”>Email address</label>
<input type=”email” name=”email” class=”form-control” id=”email-address”
placeholder=”name@example.com”>
</div>
<div class=”mb-3”>
<label for=”username” class=”form-label”>Username</label>
<input type=”text” name=”username” class=”form-control” id=”username”
placeholder=”name” />
</div>
<div class=”row g-2”>
<div class=”col-auto”>
<label for=”password” class=”visually-hidden”>Password</label>
<input type=”password” name=”password” class=”form-control” id=”password”
placeholder=”Password”>
</div>
<div class=”col-auto”>
<label for=”secret” class=”visually-hidden”>Secret Key</label>
<input type=”password” name=”secret” class=”form-control” id=”secret”
placeholder=”Secret-Key”>
</div>
<div class=”col-auto”>
<button type=”submit” class=”btn btn-primary mb-3”>Create Account</button>
```

```
  </div>
   </div>
     <p>Already have an Account ? <a href="{{ url_for('login') }}">Login</a></p>
   </div>
   </form>
    {% endblock %}
```

BASE.HTML:

```
<!DOCTYPE html>
  <html lang="en">
   <head>
   <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+Rp48ckxlpbzKgwra6"
 crossorigin="anonymous">
     <link rel="stylesheet" href="{{ url_for('static',filename='css/main.css') }}" />
     <title>Customer-Care Registry</title>
   </head>
 <body>
    {% block body %}
{% endblock %}
   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
   integrity="sha384-
JEW9xMcG8R+Ph31jmWH6WWP0WintQrMb4s7ZodauHnUtxwoG2Vi5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
</body>
</html>
```

DETAILS.HTML:

```
  {% extends ‘base.html’ %}
  {% block body %}
   <div class=”container ticket-detail”>
 <div class=”jumbotron”>
       <div class=”row”>
         <div class=”col”>
     <h2>{{ ticket[3] }}</h2>
    <p>{{ ticket[4] }}</p>
   </div>
<div class=”col”>
  <div class=”row detail-card bl”>
  <h4>Complaint Filed by {{ customer[1] }}</h4>
   </div>
   <div class=”row detail-card gr”>
 <h4>Progress: {{ ticket[5] }}</h4>
   </div>
 <div class=”row detail-card yl”>
<h4>Assigned to: {{ agent[1] }}</h4>
</div>
  </div>
 </div>
 {% if user[3] == 2 %}
    <div class=”row” >
  <form method=”POST”>
   <select name=”agent”>
    {% for user in all_users %}
  <option value=”{{user[4]}}”>{{user[0]}}</option>
 {% endfor %}
    </select>
 <input class=”btn btn-danger btn-sm” type=”submit” value=”Assign”/>
    </form>
```

```
 </div>
  {% elif user[3] == 1%}
      {% if ticket[5] == “assigned” %}
 <a href=”/accept/{{ticket[0]}}/{{user[4]}}”><button class=”btn btnsecondary”>
       Accept</button></a>
     {% elif ticket[5] == “accepted” %}
     <a href=”/delete/{{ticket[0]}}/{{user[4]}}”><button class=”btn btn-danger”
    >Close</button></a>
      {% endif %}
       {% endif %}
    </div>
     </div>
      {% endblock %}
```

HOME.HTML:

```
{% extends ‘base.html’ %}
  {% block body %}
 <div class=”container “>
 <h2>Hi, {{ user[0] }}</h2>
{% if user[3] == 0 %}
   <p>
     As a customer of our sevice, you can raise a ticket to
  bring you issue forward with a detailed description of
       the problem.
Your issues will be assigned to an agent who will take
   care of it.
 </p>
   <div class=”row”>
<div class=”col”>
<h3>File a Complaint</h3>
<form method=”POST” >
 {% if msg %}
```

```
    <div class=”alert alert-success” role=”alert”>
   {{ msg }}
  </div>
  {% endif %}
    <input name=”title” class=”form-control form-control-sm” type=”text”
 placeholder=”Ticket Header” aria-label=”.form-control-sm example” />
<br>
  <div class=”mb-3”>
     <textarea name=”description” placeholder=”Problem Description…” class=”formcontrol”
 id=”problem-desc” rows=”3”></textarea>
 </div>
 <input type=”submit” value=”Raise” class=”btn btn-warning” />
  </form>
 </div>
 <div class=”col”>
 <h3>List of Pending Complaints</h3>
<table class=”table”>
<thead class=”table-dark”>
  <tr>
  <th>Title</th>
<th>Description</th>
 <th>View</th>
</tr>
  </thead>
 <tbody>
  {% for ticket in tickets %}
      <tr>
  <td>{{ ticket[3] }}</td>
<td>{{ ticket[4] }}</td>
<td><a href=”/ticket/{{ticket[0]}}”><button class=”btn btnprimary”>
    View</button></a></td>
 </tr>
```

```
      {% endfor %}
     </tbody>
    </table>
</div>
   </div>
 {% elif user[3] == 2 %}
    <div class=”row”>
 <a href=”{{url_for(‘panel’)}}”><button class=”btn btn-primary”>Go To Admin
Panel</button></a>
  </div>
   {% elif user[3] == 1%}
 <table class=”table”>
   <thead class=”table-dark”>
   <tr>
<th>Title</th>
<th>Description</th>
    <th>View</th>
 </tr>
 </thead>
   <tbody>
  {% for ticket in tickets %}
  <tr>
 <td>{{ ticket[3] }}</td>
   <td>{{ ticket[4] }}</td>
     <td><a href=”/ticket/{{ticket[0]}}”><button class=”btn btnprimary”>
  View</button></a></td>
  </tr>
  {% endfor %}
 </tbody>
   </table>
{% endif %}
<br>
```

```
    <a href="{{url_for('logout')}}" ><button class="btn btn-outline-success">Logout</button></a>
   </div>
   {% endblock %}
```

LOGIN.HTML:

```
 {% extends 'base.html' %}
 {% block body %}
 <form method="POST" class="login-form">
 <h2 style="text-align: center;" >Login</h2>
<div class="container">
{% if msg == "Incorrect Password"%}
<div class="alert alert-danger" role="alert">
 {{ msg }}
 </div>
{% elif msg == "User does not exist" %}
 <div class="alert alert-primary" role="alert">
 {{ msg }}
 </div>
 {% endif %}
 <div class="mb-3">
  <label for="email-address" class="form-label">Email address</label>
  <input type="email" name="email" class="form-control" id="email-address"
 placeholder="name@example.com">
  </div>
 <div class="row g-2">
 <div class="col-auto">
 <label for="password" class="visually-hidden">Password</label>
 <input type="password" name="password" class="form-control" id="password"
  placeholder="Password">
 </div>
 <div class="col-auto">
   <button type="submit" class="btn btn-primary mb-3">Login</button>
```

```
 </div>
</div>
 <p>Do not have an account ? <a href="{{ url_for('register_account') }}">Sign Up</a></p>
</div>
 </form>
 {% endblock %}
```

PANEL.HTML:

```
 {% extends 'base.html' %}
 {% block body %}
<div class="container">
<div class="row">
<div class="col">
<h3>Promote Agents</h3>
<div class="container">
 <form method="POST">
  <select name="admin-candidate">
{% for user in all_users %}
<option value="{{user[4]}}">{{user[0]}}</option>
{% endfor %}
</select>
 <input class="btn btn-danger btn-sm" type="submit" value="Make Agent"/>
</form>
</div>
</div>
<div class="col">
<h3>Assign Tasks</h3>
<div class="container">
 <table class="table">
 <thead class="table-dark">
 <tr>
<th>Title</th>
```

```
 <th>Description</th>
 <th>View</th>
 </tr>
 </thead>
 <tbody>
{% for ticket in tickets %}
 <tr>
<td>{{ ticket[3] }}</td>
 <td>{{ ticket[4] }}</td>
 <td><a href=”/ticket/{{ticket[0]}}”><button class=”btn btnprimary”>
View</button></a></td>
</tr>
{% endfor %}
</tbody>
</table>
 </div>
</div>
</div>
 </div>
  {% endblock %}
```

REGISTER.HTML:

```
 {% extends ‘base.html’ %}
  {% block body %}
 <form method=”POST” class=”register-form”>
 <h2 style=”text-align: center;” >Register Account</h2>
     <div class=”container”>
 <div class=”mb-3”>
 <label for=”email-address” class=”form-label”>Email address</label>
<input type=”email” name=”email” class=”form-control” id=”email-address”
placeholder=”name@example.com”>
 </div>
```

```
  <div class=”mb-3”>
  <label for=”username” class=”form-label”>Username</label>
 <input type=”text” name=”username” class=”form-control” id=”username”
    placeholder=”name” />
 </div>
 <div class=”row g-2”>
   <div class=”col-auto”>
     <label for=”password” class=”visually-hidden”>Password</label>
   <input type=”password” name=”password” class=”form-control” id=”password”
  placeholder=”Password”>
  </div>
   <div class=”col-auto”>
    <button type=”submit” class=”btn btn-primary mb-3”>Create Account</button>
 </div>
</div>
  <p>Already have an Account ? <a href=”{{ url_for(‘login’) }}”>Login</a></p>
 </div>
</form>
  {% endblock %}
```

MAIN.CSS:

```
  .detail-card{
     text-align: center;
  margin-top:0.5em !important;
     border-bottom: 1px black solid;
   border-radius: 5px;
  border: none;
  font-family: ‘Lucida Sans’, ‘Lucida Sans Regular’, ‘Lucida Grande’, ‘Lucida Sans Unicode’, Geneva,
  Verdana, sans-serif;
  }
.bl{
  background-color: rgb(151, 151, 245) !important;
```

```
  }
 .yl{
   background-color: rgb(243, 240, 88);
  }
  .gr{
  background-color: rgb(95, 204, 91);
 }
 /* login form */
  .login-form{
  margin-top:10em;
 }
    /* ticket detail */
   .ticket-detail{
 margin-top:15em;
  border: 1px black solid;
 padding:1em;
border-radius:0.2em;
  }
.register-form{
margin-top:10em;
}
  body{
  background-image: url(‘https://externalcontent.
   Duckduckgo.com/iu/?u=http%3A%2F%2Fwww.pixelstalk.net%2Fwpcontent%
  2Fuploads%2F2016%2F04%2Fphotos-download-abstract-minimalist-wallpaper-
  HD.jpg&f=1&nofb=1’);
    background-repeat: no-repeat;
 background-size: cover;
  }
```

7.2.FEATURE 2:

REQUIREMENTS.TXT:

```
blinker==1.4
click==7.1.2
Flask==1.1.2
Flask-Mail==0.9.1
Flask-MySQLdb==0.2.0
itsdangerous==1.1.0
Jinja2==2.11.3
MarkupSafe==1.1.1
mysqlclient==2.0.3
passlib==1.7.4
Werkzeug==1.0.1
```

.GITIGNORE:

```
customer-registry
database-design.txt
config.py
__pycache__
```

DEPLOYMENT.YAML:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
name: flask-node-deployment
spec:
replicas: 1
selector:
matchLabels:
app: flasknode
template:
```

```
  metadata:
  labels:
 app: flasknode
 spec:
 containers:
- name: flasknode
 image: registry.ng.bluemix.net/flask-node/app
 imagePullPolicy: Always
 ports:
- containerPort: 5000
```

SERVICE.YAML:

```
apiVersion: v1
kind: Service
metadata:
name: flask-node-deployment
 spec:
ports:
- port: 5000
targetPort: 5000
selector:
 app: flasknode
```

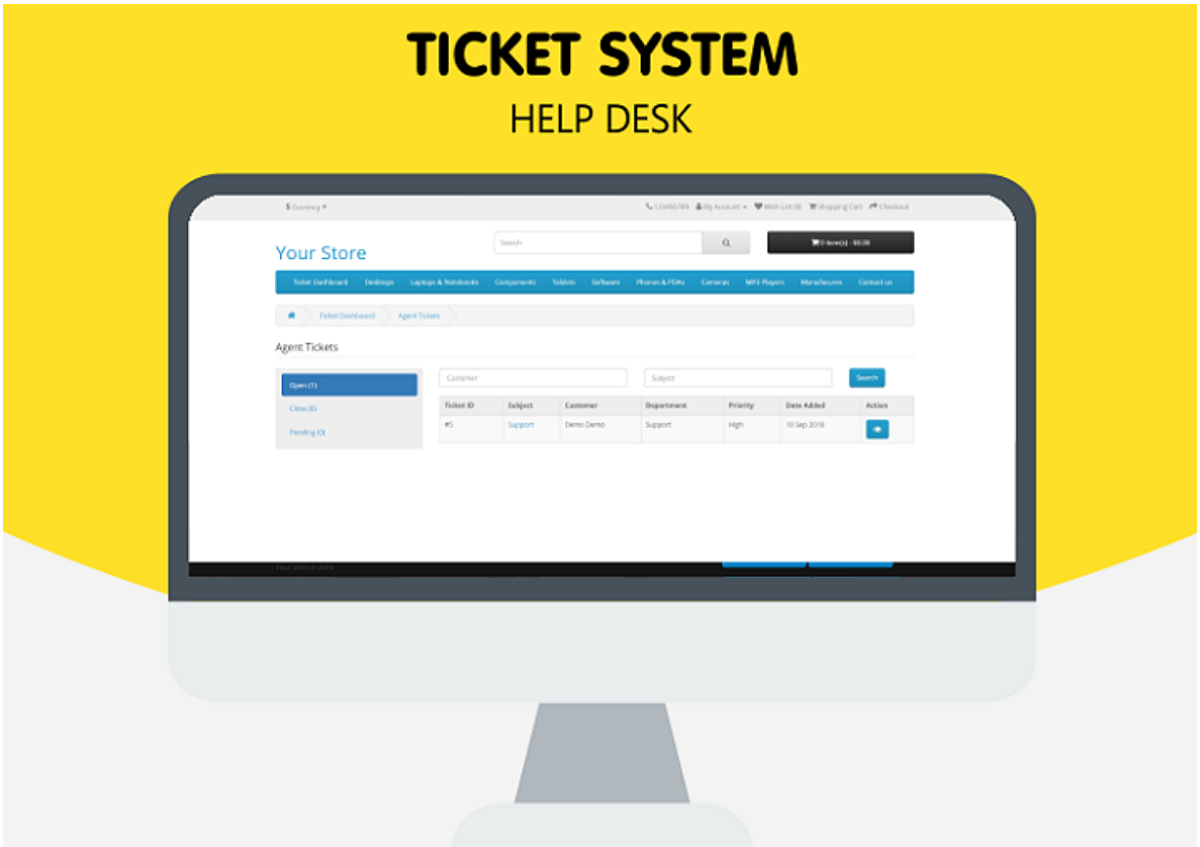
DOCKERFILE:

```
FROM python:3.8.5-alpine
WORKDIR /app
ADD . /app
RUN set -e; \
apk add –no-cache –virtual .build-deps \
 gcc \
libc-dev \
 linux-headers \
```

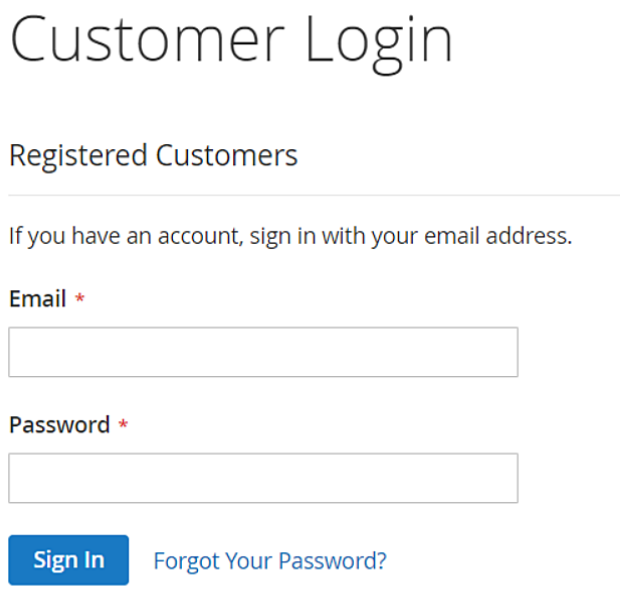
```
mariadb-dev \
 python3-dev \
 postgresql-dev \
;
 COPY requirements.txt /app
 RUN pip install -r requirements.txt
 CMD [“python”,”app.py”]
```

SOLUTIONING:

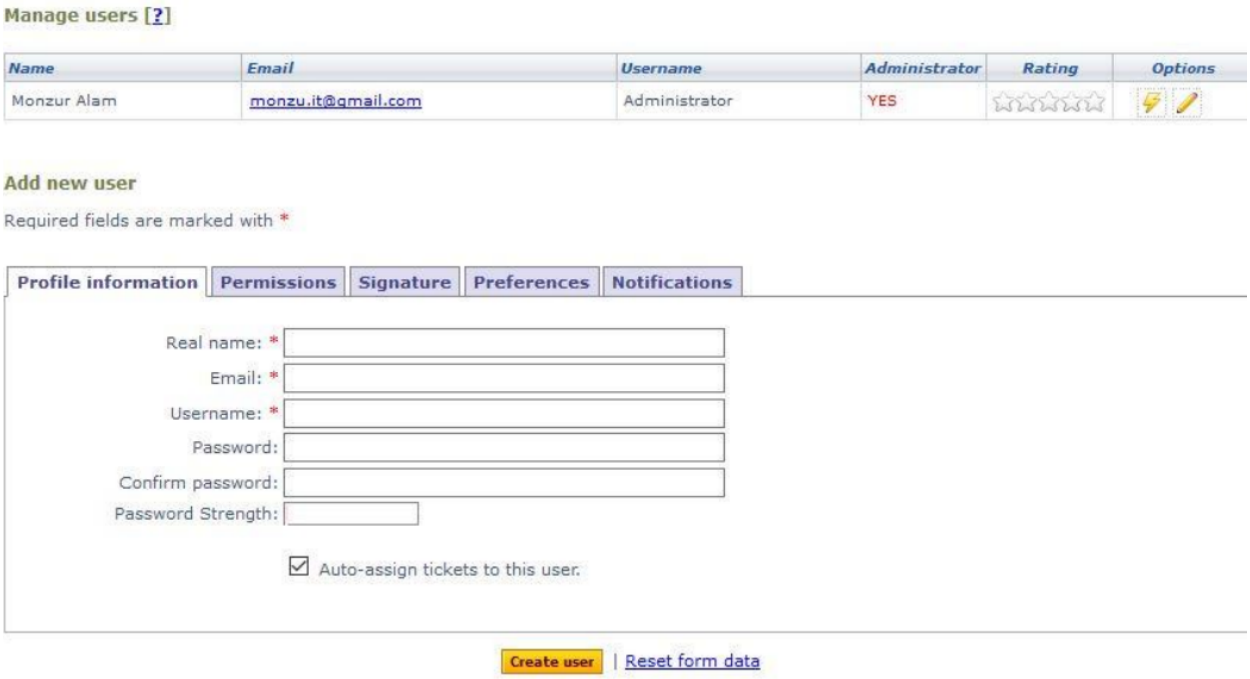
HOME PAGE:



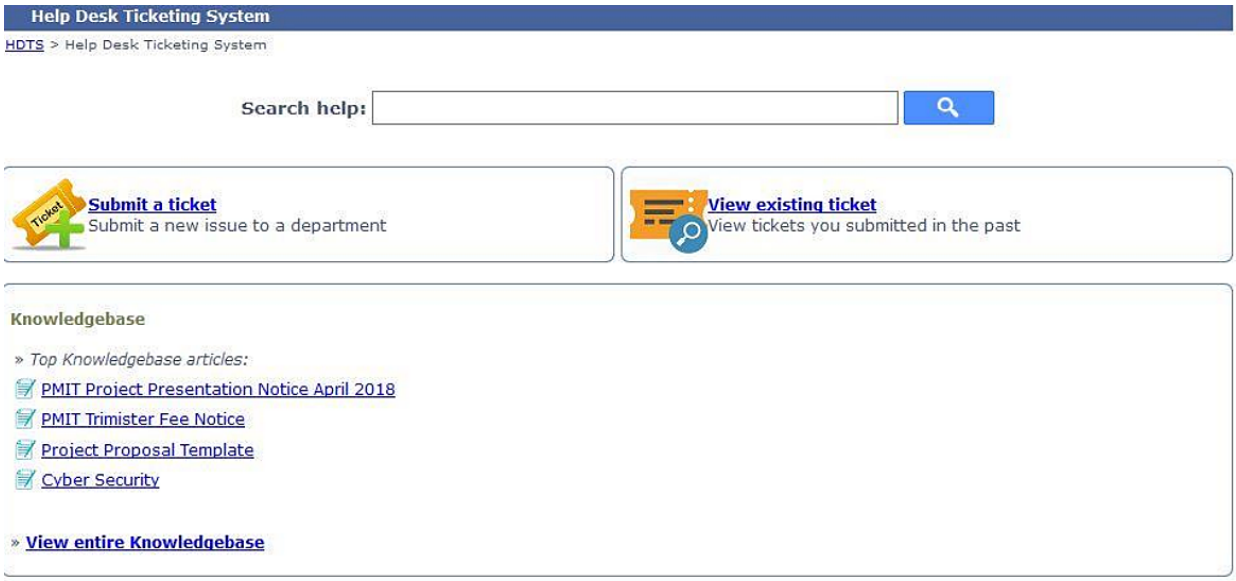
LOGIN PAGE:



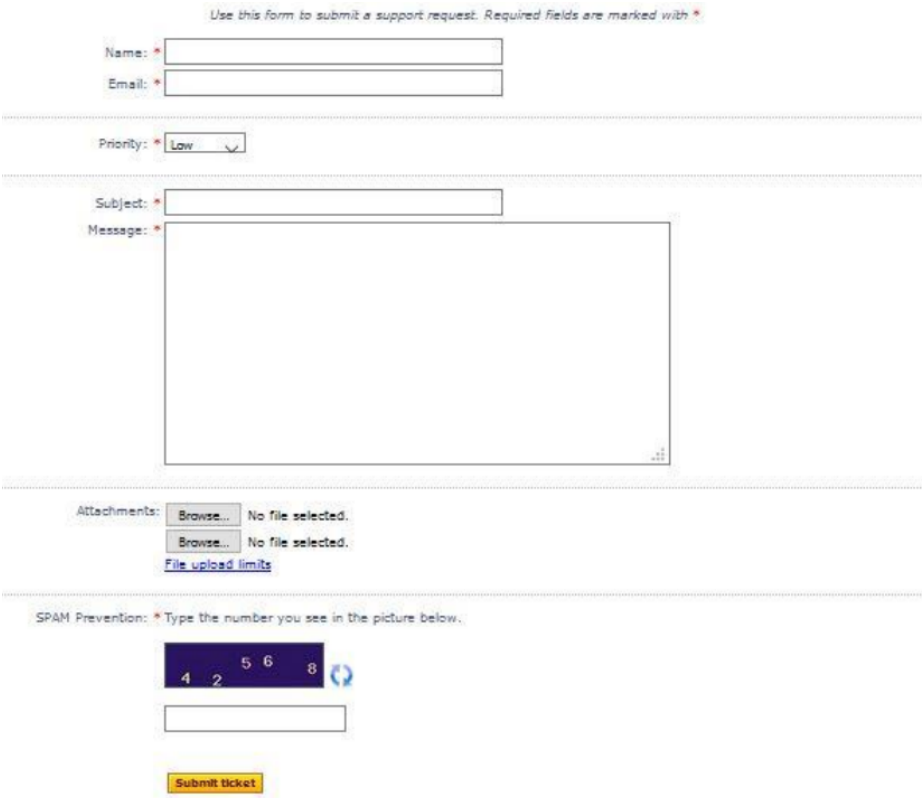
REGISTRATION PAGE:



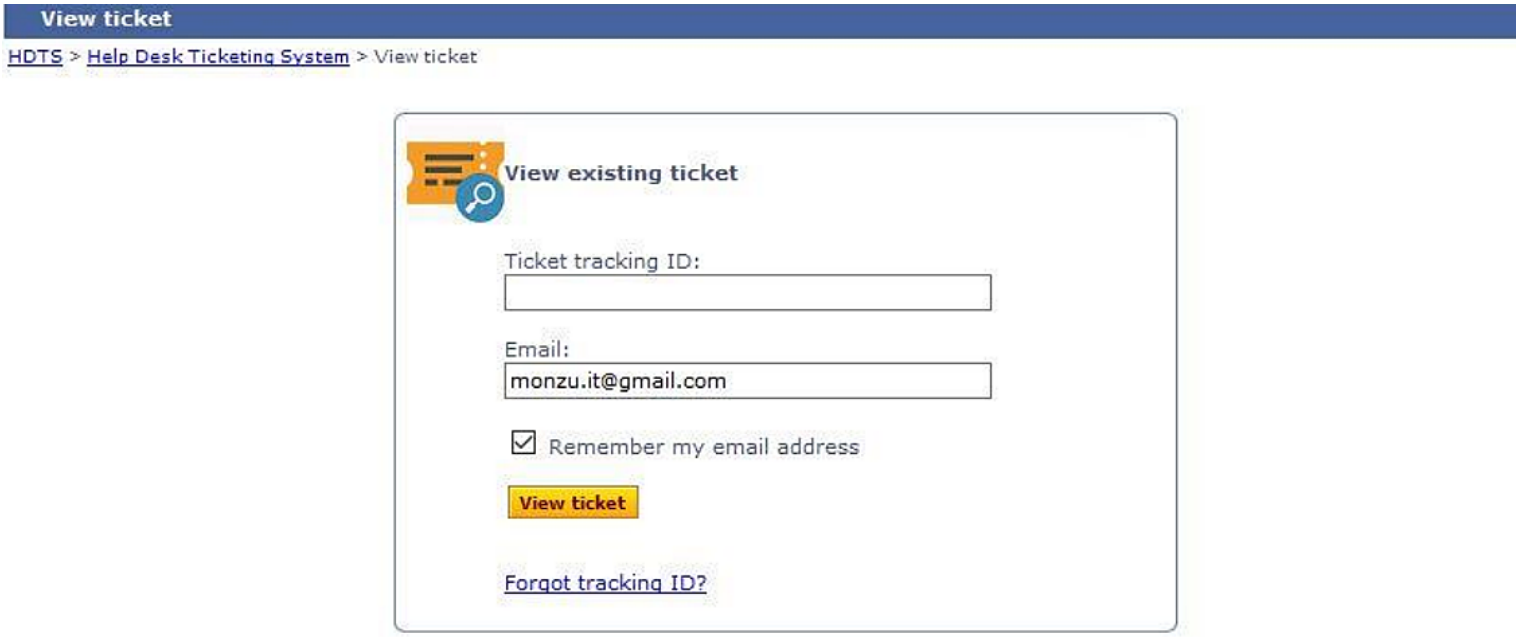
USER DASHBOARD:



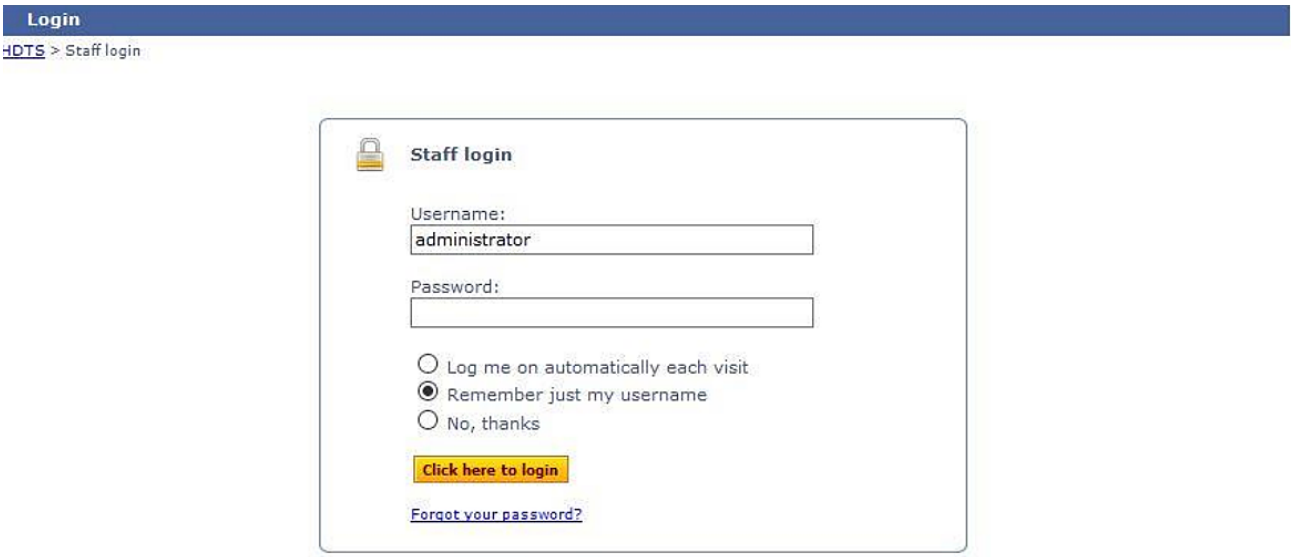
SUBMITTING THE TICKET:



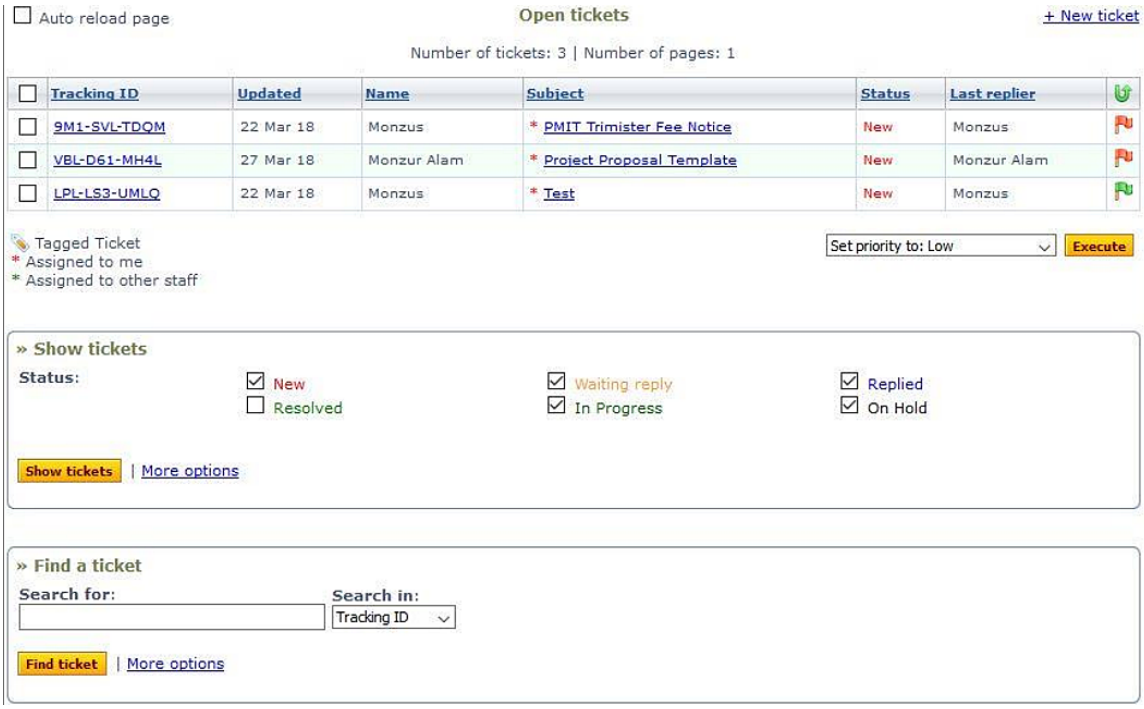
TRACKING THE TICKET:



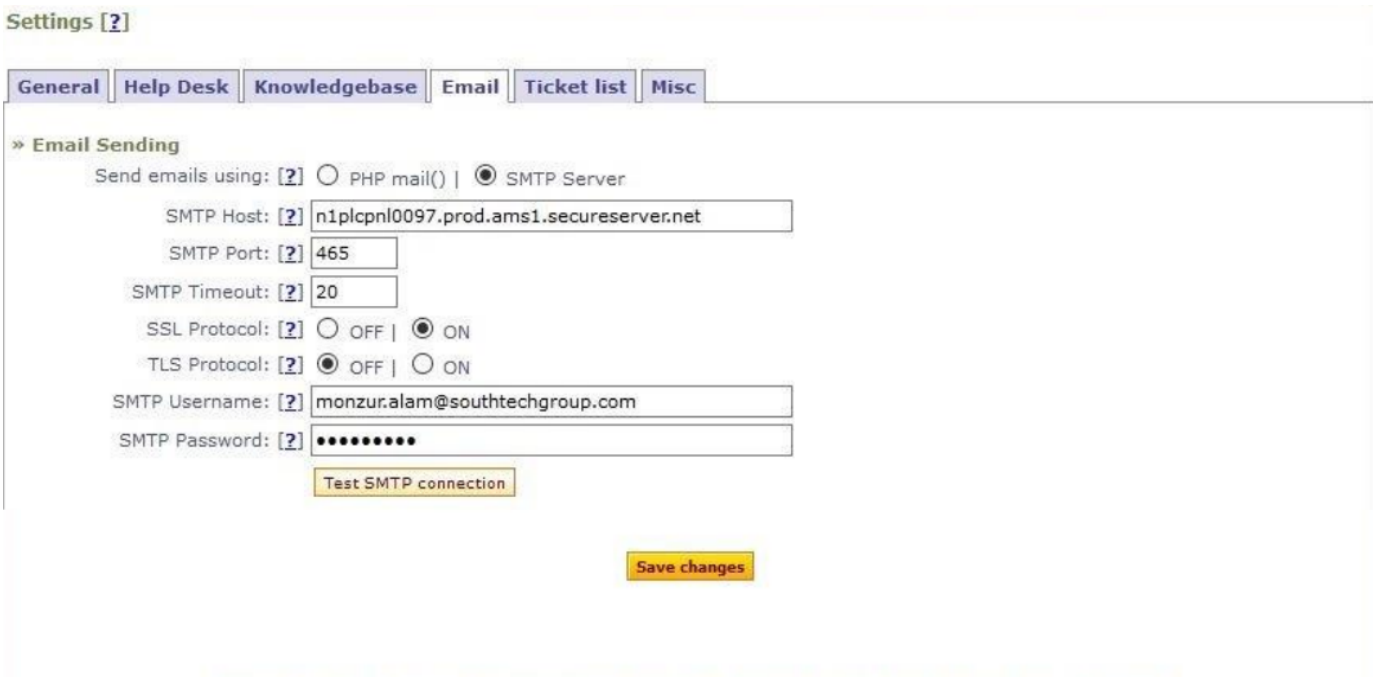
ADMIN LOGIN:



ADMIN DASHBOARD:



EMAIL SETTINGS:



7.3.DATABASE SCHEMA:

8.TESTING

8.1 TEST CASES:

Test case ID	Feature Type	Component	Test Scenario	Pre-Requisite	Steps To Execute	Test Data	Expected Result	Actual Result	Status
LoginPage_TC_OO1	Functional	Home Page	Verify user is able to see the Login/Signup popup when user clicked on My account button		1.Enter URL and click go 2.Click on My Account dropdown button 3.Verify login/Singup popup displayed or not		Login/Signup popup should display	Working as expected	Pass
LoginPage_TC_OO2	UI	Home Page	Verify the UI elements in Login/Signup popup		1.Enter URL and click go 2.Click on My Account dropdown button 3.Verify login/Singup popup with below UI elements: a.email text box b.password text box c.Login button d.New customer? Create account link e.Last password? Recovery password link		Application should show below UI elements: a.email text box b.password text box c.Login button with orange colour d.New customer? Create account link e.Last password? Recovery password link	Working as expected	pass
LoginPage_TC_OO3	Functional	Home page	Verify user is able to log into application with Valid credentials		1.Enter URL and click go 2.Click on My Account dropdown button 3.Enter Valid username/email in Email text box 4.Enter valid password in password text box	Username: manojvandaiyar@gmail.com password:manoj1234	User should navigate to user account homepage	Working as expected	pass
LoginPage_TC_OO4	Functional	Login page	Verify user is able to log into application with InValid credentials		1.Enter URL(https://shoperizer.com/) and click go 2.Click on My Account dropdown button 3.Enter InValid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: manojvandaiyar@gmail password: Testing123	Application should show 'Incorrect email or password ' validation message.	working as expected	pass
LoginPage_TC_OO4	Functional	Login page	Verify user is able to log into application with InValid credentials		1.Enter URL and click go 2.Click on My Account dropdown button 3.Enter Valid username/email in Email text box 4.Enter Invalid password in password text box 5.Click on login button	Username: manojvandaiyar@gmail.com password: manoj1234	Application should show 'Incorrect email or password ' validation message.	working as expected	Pass
RegistrationPage_TC_OO1	Functional	Registration page	Verify user is able to log into application with their personal details		1.Enter URL and click go 2.Click on My Account dropdown button 3.Enter Name 4.Enter Age 5.Enter department,purpose and login	Username: Manoj password: manoj1234 Name:Manoj Age:20 Department:Testing Purpose:Performance testing	Application should show the username,password,name,age,department,purpose	working as expected	pass
ashboardPage_TC_OO	Functional	ashboard pag	Verify user,agent,admin is able to ac		1.As a user,we can create the tickets 2.As a Admin,they can assign the agent to the customer and manage the whole process 3.As a agent,they can solve their customer's problem	User: Create the tickets Agent:Solve the customer's problem Admin:Assigned the agent to the customer	Application should show the customer's ticket and their status of the ticket in the customer's dashboard.Agent can see the customer's complaints in their dashboard.Admin can see the whole	Working as ex	pass

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	10	4	2	3	20
Duplicate	1	1	3	1	6
External	2	3	0	1	6
Fixed	11	2	4	20	37
Not Reproduced	0	0	1	0	1
Suspend	0	0	1	1	2
Won't Fix	0	3	2	1	6
Totals	24	14	13	28	80

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
Print Engine	7	0	1	7
Client Application	51	0	1	51
Security	2	0	2	2
Outsource Shipping	3	0	1	3
Execution Reporting	9	0	1	9
Final Report Output	4	0	1	4
Version Control	2	0	0	2

9.RESULTS

9.1 PERFORMANCE METRICS:

			NFT - Risk Assessment					
S.No	Project Name	Scope/feature	Functional Changes	Hardware Changes	Software Changes	Impact of Downtime	Load/Voluem Changes	Risk Score
1	Customer care regist	Existing	No Changes	No Changes	No Changes	No Downtime imapct seen..!	No Changes	GREEN

NFT - Detailed Test Plan				
S.No	Project Overview	NFT Test approach	umptions/Dependencies/R	Approvals/SignOff

			End Of Test Report					
S.No	Project Overview	NFT Test approach	NFR - Met	Test Outcome	GO/NO-GO decision	Recommendations	Identified Defects (Detected/Closed/Open)	Approvals/SignOff

10.ADVANTAGES & DISADVANTAGES

ADVANTAGES:

➤ In computer system the person has to fill the various forms &number of copies of the forms can be easily generated at a time.

➤ In computer system,it is not necessary to create the manifest but we can directly print it,which saves our time.

➤ It satisfy the user requirement.

➤ Be easy to understnd by the user and operator.

➤ Be easy to operate.

➤ Have a good user interface.

DISADVANTAGES:

➤ Excel export has not been developed for help desk,issues due to some critically.

➤ The transactions are executed in off-line mode,hence on-line data for ticket, e-mail capture and modified is not possible.

➤ Off-line reports of help desk, network, ticket cannot be generated due to batch mode execution.

11.CONCLUSION

Our project is only a humble venture to satisfy the needs to manage their project work. Several user friendly coding have also adopted. This package shall prove to be a powerfull package in satisfying all the requirements of the school. The objective of software planning is to provide a frame work that enables the manager to make reasonable estimates made within a limitted time frame at the beginning of the software project and should be updated requarly as the project progresses.

12.FUTURE SCOPE

In a nutshell,it can be summmarized that the future scope of the project circles around maintainance information regarding:

➤ We can add printer in future.

➤ We can give more advance software for help desk ticketing system incldes\ing more facilities.

➤ We will host the platform on online servers to make it assessible worldwide.

➤ Integrate multiple load balancers to distribute the loads of the system.

➤ Create the master and slave database structure to reduce the overload of the database queries

➤ implement the backups mechanism for taking backup of codebase and databases on regular basis on diiferent servers.

The above mentioned points are the enhacements which can be done to increase the applicability and usage of this project. Here we can maintain the record of help desk and issues. Also, as it can be seen that now-a-days the players are versatile.i.e, so there is ascope for introducing a methods to maintain the help desk,isssues,ticketing system. enchancement can be done to maintain all the help desk,issues, ticketing,e-mail,network.

13.APPENDIX

SOURCE CODE:

APP.PY:

```
# importing the modules
from flask import Flask, render_template, request, redirect, session, url_for
from flask_mail import Mail, Message
from flask_mysqldb import MySQL
import MySQLdb.cursors
from passlib.hash import pbkdf2_sha256
import config
# app config
app = Flask(_name_)
app.config['MYSQL_HOST'] = config.sql_server
app.config['MYSQL_USER'] = config.mysql_username
app.config['MYSQL_PASSWORD'] = config.sql_password
app.config['MYSQL_DB'] = config.mysql_username
app.config['MAIL_SERVER'] = 'smtp.gmail.com'
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] = config.email
app.config['MAIL_PASSWORD'] = config.password
app.config['MAIL_USE_SSL'] = True
app.config['MAIL_USE_TLS'] = False
mysql = MySQL(app)
app.secret_key = 'returnzero'
mail = Mail(app)
# routes
# home
@app.route("/",methods=['GET',"POST"])
def home():
if ('user' not in session.keys()) or (session['user'] == None):
return redirect(url_for('login'))
```

```
else:
cursor = mysql.connection.cursor()
cursor.execute("SELECT * FROM User WHERE id = % s",[session['user']])
userdetails = cursor.fetchone()
if userdetails[3] == 2:
return render_template("home.html",user=userdetails)
elif userdetails[3] == 1:
cursor.execute("SELECT * FROM Tickets WHERE agent=%s",[session['user']])
tickets = cursor.fetchall()
return render_template("home.html",user=userdetails,tickets=tickets)
else:
if request.method == "POST":
title = request.form['title']
description = request.form['description']
cust_id = session['user']
cursor = mysql.connection.cursor()
cursor.execute("INSERT INTO Tickets(customer,title,description)
VALUES(%s,%s,%s)",(cust_id,title,description))
mysql.connection.commit()
cursor.execute("SELECT * FROM User WHERE id = % s",[session['user']])
userdetails = cursor.fetchone()
cursor.execute("SELECT * FROM Tickets WHERE customer = %s",[session['user']])
tickets = cursor.fetchall()
return render_template("home.html",msg="Ticket Filed",user=userdetails,tickets=tickets)
cursor = mysql.connection.cursor()
cursor.execute("SELECT * FROM User WHERE id = % s",[session['user']])
userdetails = cursor.fetchone()
cursor.execute("SELECT * FROM Tickets WHERE customer = %s",[session['user']])
tickets = cursor.fetchall()
return render_template("home.html",user=userdetails,tickets=tickets)
# user account registration
@app.route("/register",methods=["GET","POST"])
```

```
def register_account():
if request.method == "POST":
username = request.form['username']
email = request.form['email']
password = request.form['password']
hashed_password = pbkdf2_sha256.hash(password)
cursor = mysql.connection.cursor()
cursor.execute("INSERT INTO User(username,email,password,role)
VALUES(%s,%s,%s,%s)",(username,email,hashed_password,0))
mysql.connection.commit()
msg = Message('registration customer care',sender=config.email,
recipients=[email]
)
msg.body = '''
Account creation in customer care registry was successful.
for raising tickets, login with your email id and password.
Thank You
'''
mail.send(msg)
return redirect(url_for("login"))
return render_template("register.html")
# login
@app.route('/login',methods=["GET","POST"])
def login():
if request.method == "POST":
email = request.form['email']
password = request.form['password']
cursor = mysql.connection.cursor()
cursor.execute("SELECT * FROM User WHERE email = % s",[email])
userdetails = cursor.fetchone()
if userdetails:
if pbkdf2_sha256.verify(password,userdetails[2]):
```

```
session['user'] = userdetails[4]
return redirect(url_for("home"))
else:
msg = "Incorrect Password"
else:
msg = "User does not exist"
return render_template("login.html",msg=msg)
return render_template("login.html")
# logout
@app.route("/logout")
def logout():
session['user'] = None
return redirect(url_for("home"))
# ticket detail
@app.route("/ticket/<int:id>",methods=["GET","POST"])
def ticket_detail(id):
cursor = mysql.connection.cursor()
cursor.execute("SELECT * FROM Tickets WHERE id=%s",[id])
ticket = cursor.fetchone()
cursor.execute("SELECT * FROM User WHERE id=%s",[ticket[1]])
customer = cursor.fetchone()
cursor.execute("SELECT * FROM User WHERE id=%s",[session['user']])
user = cursor.fetchone()
cursor.execute("SELECT * FROM User WHERE role=1")
all_users = cursor.fetchall()
cursor.execute("SELECT * FROM User WHERE id=%s",[ticket[2]])
agent = cursor.fetchone()
if agent is None:
agent = [None,None]
if user is None:
return redirect(url_for("login"))
if request.method == "POST":
```

```
agent = request.form['agent']
cursor.execute("UPDATE Tickets SET agent= %s WHERE id = %s",(agent,id))
cursor.execute("UPDATE Tickets SET progress='assigned' WHERE id = %s",[id])
mysql.connection.commit()
cursor.execute("SELECT email FROM User WHERE id=%s",[agent])
agent_mail = cursor.fetchone()[0]
msg = Message('Assigned Ticket',sender=config.email,
recipients=[agent_mail]
)
# send mail to agent
msg = Message('Assigned Ticket',sender=config.email,
recipients=[agent_mail]
)
cursor.execute("SELECT email FROM User WHERE id=%s",[ticket[1]])
customer = cursor.fetchone()[0]
msg.body = f'''
You have been assigned a ticket.
Ticket Title: {ticket[3]}
posted by: {customer}
'''
mail.send(msg)
# send mail to customer
msg = Message('Ticked Progress',sender=config.email,
recipients=[customer]
)
msg.body = f'''
Dear Customer,
Your Ticket progress has been Updated and
Assigned to an Agent of ours.
Agent : {agent_mail}
'''
mail.send(msg)
```

```
return redirect(url_for("panel"))
return
render_template("details.html",ticket=ticket,agent=agent,customer=customer,user=user,all_users=all_
users)
# admin register
@app.route("/admin/register",methods=["GET","POST"])
def admin_register():
if request.method == "POST":
username = request.form['username']
email = request.form['email']
password = request.form['password']
secret_key = request.form['secret']
if secret_key == "12345":
hashed_password = pbkdf2_sha256.hash(password)
cursor = mysql.connection.cursor()
cursor.execute("INSERT INTO User(username,email,password,role)
VALUES(%s,%s,%s,%s)",(username,email,hashed_password,2))
mysql.connection.commit()
return redirect(url_for("login"))
else:
return render_template("admin_register.html",msg="Invlaid Secret")
return render_template("admin_register.html")
# promote agent
@app.route("/panel",methods=['GET','POST'])
def panel():
id = session['user']
if id is None:
return redirect("login")
cursor = mysql.connection.cursor()
cursor.execute("SELECT * FROM User WHERE id=%s",[id])
user_details = cursor.fetchone()
if user_details[3] != 2:
```

```
return "You do not have administrator privileges"
else:
cursor.execute("SELECT * FROM User WHERE role=0")
all_users = cursor.fetchall()
cursor.execute("SELECT * FROM Tickets WHERE progress IS NULL")
tickets = cursor.fetchall()
if request.method == "POST":
user_id = request.form['admin-candidate']
cursor = mysql.connection.cursor()
cursor.execute("UPDATE User SET role=1 WHERE id = %s",[user_id])
mysql.connection.commit()
cursor.execute("SELECT * FROM User WHERE id = %s",[user_id])
promoted_agent = cursor.fetchone()
msg = Message('Promoted to Agent',sender=config.email,recipients=[promoted_agent[1]])
msg.body = """
Dear User,
You have been promoted to an Agent in the Customer-Care-Registry.
You will be able to handle tickets for the customer from now on.
Congratulations.
"""
mail.send(msg)
return redirect(url_for("panel"))
return render_template("panel.html",all_users=all_users,user=user_details,tickets=tickets)
# accept ticket
@app.route("/accept/<int:ticket_id>/<int:user_id>")
def accept(ticket_id,user_id):
cursor = mysql.connection.cursor()
cursor.execute("SELECT * FROM User WHERE id = %s",[user_id])
agent = cursor.fetchone()
cursor.execute("SELECT * FROM Tickets WHERE id=%s",[ticket_id])
ticket = cursor.fetchone()
cursor.execute("SELECT email FROM User WHERE id=%s",[ticket[1]])
```

```
customer = cursor.fetchone()
if agent[4] == ticket[2]:
cursor.execute("UPDATE Tickets SET progress='accepted' WHERE id=%s",[ticket_id])
mysql.connection.commit()
msg = Message('Ticket Progress',sender=config.email,recipients=[customer[0]])
msg.body = f"""
Dear User,
Your Ticket has been accepted by {agent[1]}
"""
mail.send(msg)
return redirect(url_for("home"))
# close ticket
@app.route("/delete/<int:ticket_id>/<int:user_id>")
def delete(ticket_id,user_id):
cursor = mysql.connection.cursor()
cursor.execute("SELECT * FROM User WHERE id = %s",[user_id])
agent = cursor.fetchone()
cursor.execute("SELECT * FROM Tickets WHERE id=%s",[ticket_id])
ticket = cursor.fetchone()
if agent[4] == ticket[2]:
cursor.execute("DELETE FROM Tickets WHERE id=%s",[ticket_id])
mysql.connection.commit()
cursor.execute("SELECT * FROM User WHERE id=%s",[ticket[1]])
customer = cursor.fetchone()
msg = Message('Ticket Progress',sender=config.email,recipients=[customer[1]])
msg.body = f"""
Dear User,
Your Ticket has been Closed by {agent[1]}
Thanks For using Customer Care Registry.
"""
mail.send(msg)
return redirect(url_for("home"))
```

```
# run server
if _name_ == "_main_":
app.run(debug=True,host='0.0.0.0',port='8080')
```

GITHUB AND PROJECT DEMO LINK:

GITHUB LINK :
https://github.com/IBM-EPBL/IBM-Project-4274-1658727164

PROJECT DEMO LINK:
https://drive.google.com/file/d/1A_SWw-JA4EVxIGwJ_zQJZwLlwelbZB3t/view?usp=drivesdk