Django Application for Gas Utility Company Customer Service

Mayur Patil

***Abstract*—The rapid increase in the number of customer service requests for a gas utility company has highlighted the limitations of the current service request management system. This paper outlines the development of a Django-based web application designed to improve the management of customer service requests, provide real-time tracking, and streamline customer support workflows. The proposed system allows customers to submit service requests, track their status, and interact with support staff, while enabling representatives to efficiently manage and resolve issues.**

***Index Terms*—Django, web application, customer service, gas utility, service request management, request tracking.**

I. Introduction

The gas utility industry is experiencing a surge in service requests, stemming from billing issues, gas supply disruptions, meter malfunctions, and more. The current manual or outdated systems in place fail to cope with the increased volume of customer inquiries, leading to inefficiencies, long wait times, and dissatisfied customers. In response, this paper proposes a Django-based application that allows customers to submit service requests online, track the progress of their issues, and interact with customer support representatives. The system also includes administrative features for managing and resolving customer requests.

II. System Architecture

The proposed system follows a modular architecture, with a focus on scalability, maintainability, and real-time updates. It includes a backend built with Django, a relational database for persistent storage, and a frontend that presents dynamic content to the user.

A. Frontend

The frontend of the application is built using HTML5 and Django's templating engine to create dynamic and user-friendly pages. It allows customers to submit requests, view their request history, and track the status of open issues. Customer support representatives use a dedicated dashboard to manage and resolve service requests.

B. Backend

The backend of the application uses Django, a powerful Python framework, to implement business logic and data processing. Django's ORM (Object-Relational Mapping) feature facilitates seamless interaction with the relational database, enabling efficient storage and retrieval of customer data and service request details.

C. Database

A relational database such as PostgreSQL or MySQL is used to store the customer data, service requests, and associated actions. The database schema is designed to support high volumes of service requests and related activities, ensuring data integrity and efficient querying.

D. Authentication

Django's built-in authentication system is used for managing user accounts and session data. Customers and support representatives are provided with different levels of access, ensuring that each user has access only to relevant functionality.

III. Features of the Application

The application provides several key features for both customers and customer support representatives.

A. Service Requests

Customers can submit service requests through an easy-to-use web interface. The request form allows customers to choose the type of service (e.g., billing, supply, meter) and describe the issue in detail. Customers may also upload relevant attachments to provide more context.

B. Request Tracking

Once a request is submitted, customers can track its status in real-time. The system provides updates on whether the issue is pending, in progress, or resolved. Customers can view timestamps indicating when the request was submitted and when it was resolved.

C. Support Representative Dashboard

Customer support representatives have access to a dashboard where they can view and manage service requests. They can update the status of requests, escalate issues, and add notes or comments. The dashboard also provides an overview of request statistics, helping representatives prioritize urgent issues.

D. Email Notifications

Automated email notifications are sent to both customers and support representatives when a request is submitted, resolved, or updated. This ensures transparent communication and reduces the chances of requests being overlooked.

IV. Application Structure

The Django application follows the Model-View-Template (MVT) architecture. The system is divided into several components, as described below:

A. Directory Structure

```
gas_utility/
 manage.py
 gas_utility/
    settings.py
    urls.py
    wsgi.py
 customers/
    models.py
    views.py
    forms.py
    urls.py
 support/
    models.py
    views.py
    forms.py
    urls.py
 templates/
     customers/
     support/
```

B. Models

The system includes two main models: `ServiceRequest` and `SupportAction`.

The `ServiceRequest` model tracks customer requests, storing fields such as request type, description, status, and timestamps for submission and resolution.

```
from django.db import models
from django.contrib.auth.models import User

class ServiceRequest(models.Model):
    REQUEST_TYPES = [
        ('BILLING', 'Billing Issue'),
        ('SUPPLY', 'Gas Supply Issue'),
        ('METER', 'Meter Issue'),
        ('OTHER', 'Other'),
    ]

    customer = models.ForeignKey(User, on_delete=mo
    request_type = models.CharField(max_length=10,
    description = models.TextField()
    status = models.CharField(max_length=20, defau
    created_at = models.DateTimeField(auto_now_add=
    resolved_at = models.DateTimeField(null=True, b
    attachment = models.FileField(upload_to='servic

    def __str__(self):
        return f"{self.customer.username} - {self.r
```

The `SupportAction` model allows customer support staff to record actions taken on service requests, such as resolving or escalating issues.

```
from django.db import models
from customers.models import ServiceRequest

class SupportAction(models.Model):
    ACTION_TYPES = [
        ('RESOLVE', 'Resolve Issue'),
        ('ESCALATE', 'Escalate Issue'),
        ('COMMENT', 'Add Comment'),
    ]

    request = models.ForeignKey(ServiceRequest, on_
    action_type = models.CharField(max_length=10, c
    note = models.TextField()
    action_taken_at = models.DateTimeField(auto_now

    def __str__(self):
        return f"{self.request.customer.username} -
```

V. Django Views and Forms

A. Customer Views

Customer views include the functionality for submitting and tracking service requests. The following view handles the submission of new requests:

```
from django.shortcuts import render, redirect
from .models import ServiceRequest
from .forms import ServiceRequestForm

def submit_request(request):
    if request.method == 'POST':
        form = ServiceRequestForm(request.POST, rec
        if form.is_valid():
            form.instance.customer = request.user
            form.save()
            return redirect('customers:request_lis
    else:
        form = ServiceRequestForm()
    return render(request, 'customers/create_request.html', {'form': form})
```

B. Support Views

Support representatives use the following view to manage service requests and resolve customer issues:

```
from django.shortcuts import render, get_object_or_404
from .models import ServiceRequest, SupportAction
from .forms import SupportActionForm

def request_dashboard(request):
    requests = ServiceRequest.objects.filter(status='Pending')  # Only show pending request
    return render(request, 'support/dashboard.html', {'requests': requests})

def request_detail(request, pk):
    service_request = get_object_or_404(ServiceRequest, pk=pk)
    if request.method == 'POST':
        form = SupportActionForm(request.POST)
        if form.is_valid():
            form.instance.request = service_request
            form.save()
            service_request.status = 'Resolved'  # Update request status
            service_request.save()
            return redirect('support:request_dashboard')
    else:
        form = SupportActionForm()
    return render(request, 'support/request_detail.html', {'form': form, 'request': service
```

VI. Conclusion

This Django-based web application improves the efficiency of managing customer service requests for a gas utility company. By automating the request submission and tracking processes, the system reduces customer wait times, enhances transparency, and empowers customer support representatives to resolve issues more effectively. The modular structure of the application allows for easy maintenance and scalability to accommodate future growth in service requests.

Acknowledgment

We would like to acknowledge the contributions of the development team and the support of the gas utility company for providing insights and requirements for this project.