Case Study: Development of a Customer Service Request Management System for Gas Utility Company

---

Overview:

The gas utility company has been struggling to manage a high volume of customer service requests, leading to increased wait times and deteriorating service quality. The current system is outdated and unable to scale to meet growing customer demands. In response, a robust Django-based web application is proposed, focusing on providing an efficient platform for customers to submit, track, and manage service requests online.

This case study outlines the proposed solution, detailing the key features of the application and how it addresses the company's needs. The goal of this system is to enhance customer satisfaction by improving the request submission process and enabling real-time tracking, while also optimum support staff efficiency.

Objective:

To develop a scalable, secure, and user-friendly Django web application for the gas utility company that:

1. Allows customers to submit service requests online.
2. Provides customers with real-time tracking of their requests, including submission and resolution timestamps.
3. Equips customer support representatives with tools to manage requests efficiently.

Proposed Solution:

The Django-based application will be designed with two user interfaces:

1. **Customer Portal** for submitting and tracking service requests.
2. **Admin Dashboard** for customer support representatives to manage requests and ensure timely resolution.

Key Features of the Application:

1. Online Service Request Submission

Customers will have the ability to:

- Select the type of service request (e.g., Gas Leak, Meter Issue, Billing Inquiry) from a predefined list.
- Provide a detailed description of the issue.
- Attach relevant documents, images, or videos to support their request (e.g., photos of faulty equipment or gas meter readings).

Implementation:

- **Django Forms:** Django forms will be utilized for customer input, allowing for seamless data entry. Fields will dynamically adjust based on the service request type selected.
- **File Attachments:** The application will allow customers to attach files such as images and documents. File validation will be handled on both the front and backend to ensure secure and proper file uploads.

Security Considerations:

- **Input Validation:** All user inputs, including file uploads, will be validated to prevent injection attacks, while file attachments will be limited to specific file types (e.g., JPEG, PNG, PDF).
- **HTTPS Protocol:** All data transfers, including file uploads, will be secured using HTTPS to encrypt communications.

2. Service Request Tracking

Customers will be able to:

- View the current status of their service request (e.g., Submitted, In Progress, Resolved).
- Access timestamps for the request submission and the resolution.
- Receive automated email or SMS updates when the status of their request changes.

Implementation:

- **Status Updates:** Django's built-in ORM (Object-Relational Mapping) will be used to track the lifecycle of each service request. Customers can check the current status of their request through their personal dashboard, which will show key details like submission time, current status, assigned technician, and expected resolution time.
- **Notifications:** Integration with Twilio (for SMS) and SendGrid (for email) will allow customers to receive real-time updates when their request status changes.

3. Customer Support Interface

Customer support representatives will have access to:

- A dashboard displaying all customer service requests in a queue.
- The ability to assign service requests to technicians, update request statuses, and add internal notes for tracking.
- Filtering options to sort requests by type, urgency, date, and other relevant factors.

Implementation:

- **Admin Interface:** Django’s admin interface will be customized to allow staff to efficiently manage customer requests. Requests can be filtered by category (e.g., meter issues, gas leaks), and escalated requests can be flagged for priority handling.
- **Role-Based Access Control:** Role-based permissions will be implemented to ensure only authorized personnel can update or resolve service requests.

4. Request History and Account Management

Customers will be able to:

- View their past service requests and their resolutions.
- Access their account details, including service history, contact information, and billing status.

Implementation:

- **User Authentication:** Django's built-in authentication system will be used for secure customer logins. Customers can register, log in, and view their personal information and service request history in a secure environment.
- **Profile Management:** Customers can update their profile information (e.g., address, phone number) and manage notification preferences (e.g., SMS or email alerts).

5. Escalation System

The application will include an automated escalation system that triggers when a request remains unresolved for an extended period. This system will:

- Automatically escalate overdue requests to higher-tier management or technicians for faster resolution.
- Notify both the customer and the internal support team about the escalation, ensuring transparency and reducing wait times.

Implementation:

- **Automated Escalation:** This feature will be built using Celery, a distributed task queue, to monitor service requests that exceed the defined service-level agreement (SLA). Once a request breaches the SLA, notifications are automatically triggered, and the request is escalated to a priority queue.

User Experience

The application offers an intuitive user interface that guides users through the process of submitting service requests and managing their profiles. Key elements include:

- **Responsive Design**: The application is designed to work seamlessly on both desktop and mobile devices.
- **Error Handling**: Users receive clear messages when errors occur, such as missing customer profiles, ensuring a smooth experience.
- **Styling**: CSS and Bootstrap are used to enhance the visual appeal of forms and messages, improving overall usability.

Technical Architecture:

1. **Frontend:**

   - **HTML/CSS (Bootstrap)** for responsive design.
   - **JavaScript (AJAX)** for dynamic content loading without refreshing pages, improving user experience.

2. **Backend:**

   - **Django Framework (Python)** for handling business logic, database interactions, and request routing.
   - **Django REST Framework (DRF)** for any future API needs (e.g., if mobile apps are required for field technicians or customers).
   - **Celery + Redis** for task management, including notifications and escalation workflows.

3. **Database:**

   - **PostgreSQL** will serve as the primary database, offering scalability, ACID compliance, and robust support for concurrent operations.

4. **Hosting:**

   - **AWS EC2** or **Heroku** for deployment, with **S3** for secure file storage (attachments).
   - **Nginx** for reverse proxy and load balancing.

5. **Security Features:**

   - **OAuth 2.0 or JWT** for secure user authentication and session management.
   - **CSRF Protection** and **XSS Prevention** to ensure secure forms and inputs.
   - **SSL/TLS Encryption** to secure all user data, especially during login, file uploads, and communications.

Feasibility and Scalability:

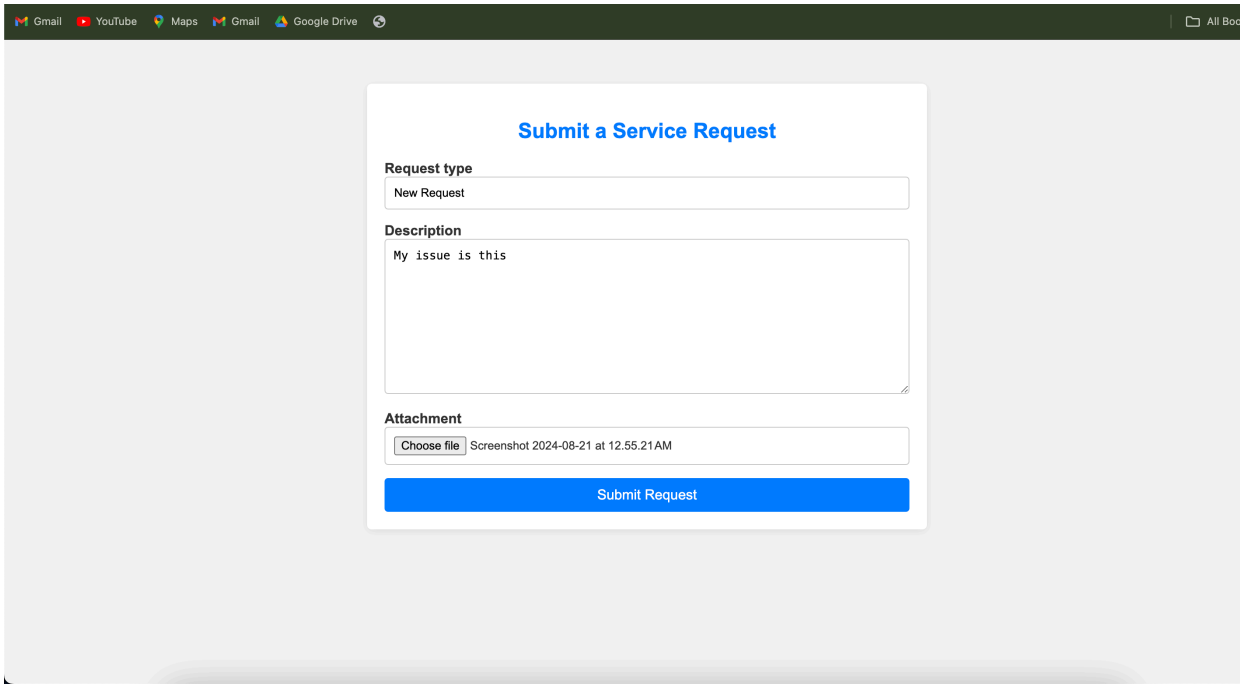
- **Performance Optimizations:**
  - Using **Redis** as a cache backend ensures faster request retrieval.
  - **Database indexing** on frequently queried fields (e.g., request status) optimizes performance.
- **Scalability:**
  - The app can scale horizontally by using Django with **Gunicorn** or **uWSGI** behind an **NGINX** load balancer.
  - **Docker** containers can be employed for easy deployment and scaling in cloud environments like AWS or Azure.
- **Security Measures:**
  - **OAuth 2.0** or **JWT** authentication can be added for API security.
  - Django's built-in protection against CSRF (Cross-Site Request Forgery) and XSS (Cross-Site Scripting) ensures data integrity.

Future Enhancements:

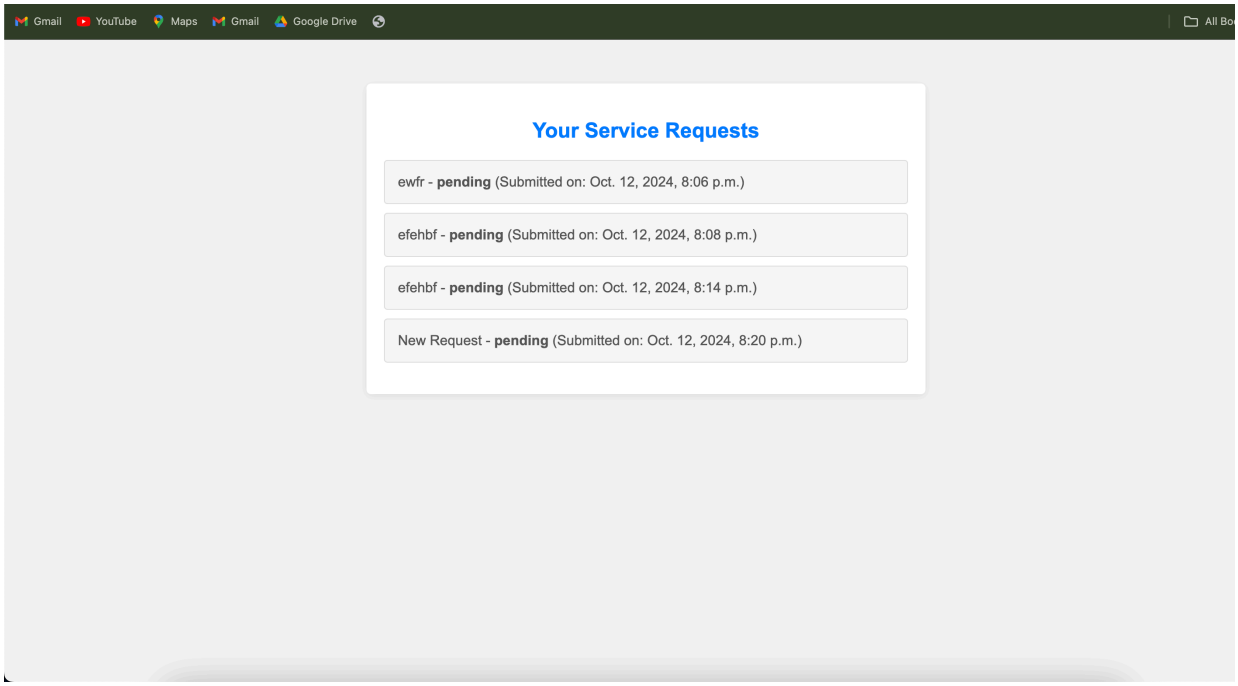
- **Mobile App Integration**: A mobile app can be developed for both customers and field technicians to enhance accessibility and real-time service tracking.
- **AI-Powered Chatbot**: Integration of a chatbot for handling common service queries and automating simple requests.
- **Data Analytics and Reporting**: A reporting tool can be developed to generate insights on service efficiency, customer satisfaction, and technician performance.

Application Interface

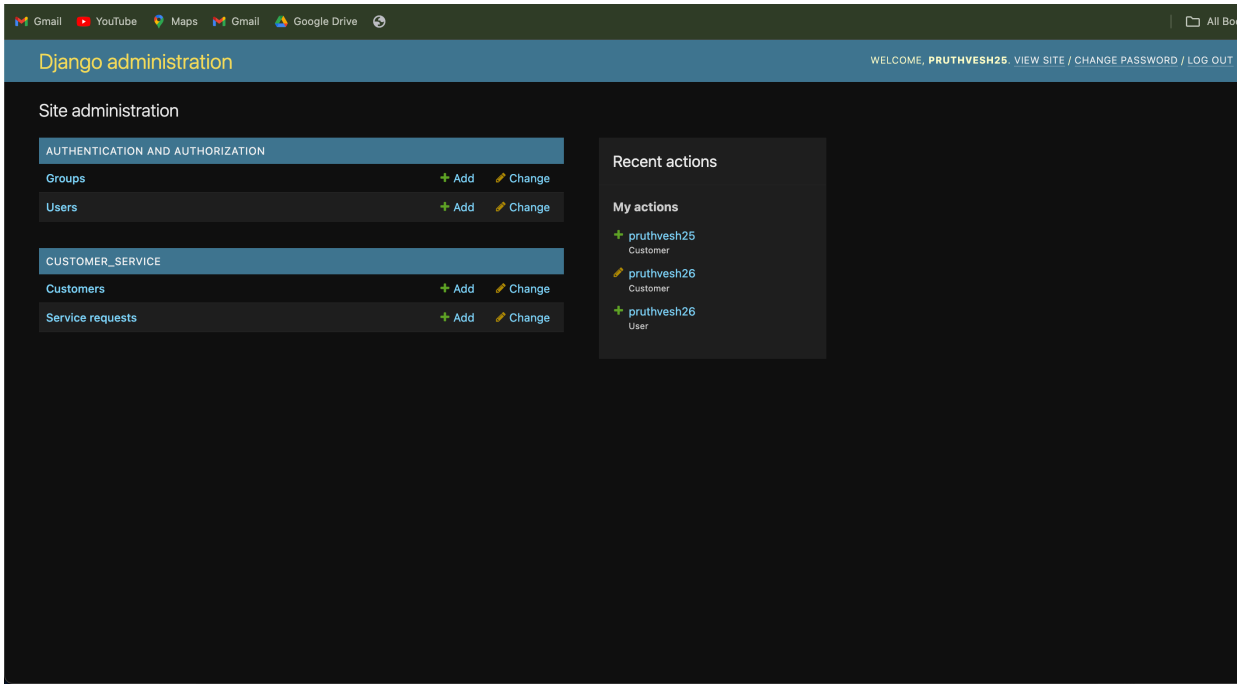
Submit Page :



After Submission :



Admin Controls :



Challenges Faced

During the development of the Gas Utility Management System, several challenges were encountered:

- **User Authentication**: Implementing secure user authentication while ensuring ease of access.
- **Error Handling**: Developing a robust error handling mechanism to provide meaningful feedback to users.
- **Database Management**: Configuring the database to handle user data and service requests efficiently.

Conclusion:

The proposed Django application for the gas utility company will significantly streamline service request management and improve customer satisfaction by providing a seamless platform for request submission and tracking. The system's robust architecture, coupled with automated workflows and real-time updates, ensures that the company can handle increasing service demands without compromising on quality.

By enhancing transparency through request tracking, automating the escalation of unresolved issues, and providing efficient tools for customer support representatives, this solution addresses the company's current pain points while laying the foundation for future growth and innovation.

This complete solution is designed to be scalable, secure, and easy to use, ensuring that the company can meet its service-level agreements and enhance its reputation for excellent customer service.