**A**

**Major Project Report**

**On**

**“CUSTOMER RELATIONSHIP MANAGEMENT”**

Submitted in partial fulfillment of the

Requirements for the award of the degree of

**Bachelor of Technology**

**In**

**Computer Science & Information Technology**

**By**

**KAMMARI RASHMIKA - 20R21A3319**

Under the guidance of

**S. PARVATHI**

**Assistant Professor**

**Department of Computer Science & Information Technology** 

**April, 2024**



**Department of Computer Science & Information Technology**

**CERTIFICATE**

This is to certify that the project entitled **“CUSTOMER RELATIONSHIP MANAGEMENT”** has been submitted by **Kammari Rashmika (20R21A3319),** in partial fulfillment of the requirements for the award ofadegree of Bachelor of Technology in Computer Science and Information Technology from MLR Institute of Technology, Hyderabad. The results embodied in this project have not been submitted to any other University or Institution for the award of any degree or diploma.

**Internal Guide** **Head of the Department**

S.PARVATHI Dr.P.SUBHASHINI

Assistant Professor Professor

**External Examiner**

**i**



**Department of Computer Science & Information Technology**

**DECLARATION**

I hereby declare that the project entitled **“CUSTOMER RELATIONSHIP MANAGEMENT”** is the work done during the periodfrom **July 2023 to April 2024** and is submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Information Technology from MLR Institute of Technology, Hyderabad. The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

**Kammari Rashmika - 20R21A3319**

**ii**



**Department of Computer Science & Information Technology**

**ACKNOWLEDGEMENT**

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that we now have the opportunity to express our guidance to all of them.

First of all, I would like to express my deep gratitude towards our internal guide **S. Parvathi, Assistant Professor, Department of CSIT** for her support in the completion ofour dissertation. I wish to express my sincere thanks to **Dr. P. SUBHASHINI,** HOD, Dept. of CSIT, and also principal **Dr. K. SRINIVAS RAO** for providing the facilities tocomplete the dissertation.

I would like to thank all our faculty and friends for their help and constructive criticism during the project period. Finally, I am very much indebted to our parents for their moral support and encouragement to achieve goals.

**Kammari Rashmika - 20R21A3319**

**iii**



**Department of Computer Science & Information Technology**

**ABSTRACT**

Customer Relationship Management (CRM) is a strategic approach employed by businesses to foster and nurture long-lasting connections with their customers. In today's fiercely competitive market landscape, companies recognize that successful CRM is not merely a transactional process but a holistic strategy encompassing customer acquisition, engagement, retention, and satisfaction. This abstract explores the essential components and benefits of CRM, improving personalized interactions to enhance customer loyalty and drive sustainable business growth. Our SaaS application offers a Quote Service facilitating a quick selection of the best regional services. Acting as a bridge between users and Service providers, it ensures swift, personalized quotes. Service centers respond promptly, providing accurate, competitive quotes tailored to users' needs. This enhances user experience by connecting them efficiently with top-tier service providers in their area. Existing CRM systems often face challenges like complex interfaces and complex automation, and sometimes struggle to adapt to specific industry needs. Some may need more user-friendly features or have a steep learning curve, making it challenging for teams to fully embrace them. The CRM system can be implemented in service centers and revolutionizes the customer experience by providing a seamless and efficient way for clients to stay informed about the status of their device repairs. This initiates a comprehensive tracking system that keeps customers informed about the progress of their device repair journey. This enhances transparency and convenience and ensures that customers can collect their devices at the designated time, leading to increased satisfaction and loyalty. This project represents a powerful and customer-centric approach to managing service center interactions.

**iv**

**LIST OF FIGURES**

| **Figure No** | **Figure Name** | **Page. NO** |
| --- | --- | --- |
| **4.1** | Proposed System Architecture | **12** |
| **4.1.2** | Representing Docker containers |  |
| **4.1.3** | Block Diagram of System Architecture | **14** |
| **4.3.1** | UML Use Case Diagram |  |
| **4.3.2** | UML Class Diagram |  |
| **4.3.3** | UML Sequence Diagram |  |
| **4.3.4** | UML Activity Diagram |  |
| **5.1.1** | Request Response Cycle in Django |  |
| **7.1** |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**LIST OF TABLES**

| **Table No** | **Table Name** | **Page. No** |
| --- | --- | --- |
| **2.4** | Summary of Literature Survey | **7** |
|  |  |  |

**v**

**INDEX**

| **CERTIFICATE** | **i** |
| --- | --- |
| **DECLARATION** | **ii** |
| **ACKNOWLEDGEMENT** | **iii** |
| **ABSTRACT** | **iv** |
| **LIST OF FIGURES** | **v** |
| **LIST OF TABLES** | **v** |

| **CHAPTER 1: INTRODUCTION** | | |
| --- | --- | --- |
| 1.1 | Overview | **1** |
| 1.2 | Purpose of the project | **2** |
| 1.3 | Motivation | **2** |
| **CHAPTER 2: LITERATURE SURVEY** | | |
| 2.1 | Customer Relationship Management Research from 2000 to 2020: An Academic Literature Review and Classification | **3** |
| 2.1.1 | Description | **3** |
| 2.1.2 | Advantages & Disadvantages | **4** |
| 2.2 | A Literature Review on Customer Relationship Management | **4** |
| 2.2.1 | Description | **4** |
| 2.2.2 | Advantages & Disadvantages | **5** |
| 2.3 | [The Importance of Customer Relationship Management Systems to Business Management and Marketing Strategie](https://www.emerald.com/insight/content/doi/10.1108/978-1-83753-686-320241008/full/html)s | **5** |
| 2.3.1 | Description | **6** |
| 2.3.2 | Advantages & Disadvantages | **6** |
| 2.4 | Summary Of Literature Survey | **7** |
| **CHAPTER 3: PROPOSED SYSTEM** | | |
| 3.1 | Introduction | **8** |
| 3.2 | Advantages of Proposed System | **8** |
| 3.3 | System Requirements | **10** |
| 3.3.1 | Software Requirements | **10** |
| 3.3.2 | Hardware Requirements | **11** |

| **CHAPTER 4: SYSTEM DESIGN** | | |
| --- | --- | --- |
| 4.1 | Proposed System Architecture | **15** |
| 4.2 | Proposed System Modules | **17** |
| 4.2.1 | Customer Contact Storage and Management | **18** |
| 4.2.2 | Service Ticket Management and Tracking | **24** |
| 4.2.3 | Quote Generation | **25** |
| 4.2.4 | Automated Communication | **25** |
| 4.2.5 | Reporting and Analytics | **26** |
| 4.3 | UML Diagrams | **27** |
| 4.3.1 | Use Case Diagram | **27** |
| 4.3.2 | Class Diagram | **29** |
| 4.3.3 | Sequence Diagram | **29** |
| 4.3.4 | Activity Diagram |  |
| **CHAPTER 5: IMPLEMENTATION** | | |
| 5.1 | Technologies | **31** |
|  |  |  |
| 5.2 | Source Code | **34** |
| **CHAPTER 6: TESTING** | | **58** |
| **CHAPTER 7: RESULTS** | | |
|  |  | **60** |
|  |  | **61** |
| **CONCLUSION** | | **65** |
| **FUTURE ENHANCEMENT** | | **66** |
| **REFERENCES** | | **67** |

# CHAPTER 1: INTRODUCTION

Customer Relationship Management (CRM) project, where we embark on a journey to revolutionize the way we connect with our valued customers. In an era defined by dynamic market landscapes and evolving consumer preferences, effective CRM is the cornerstone of sustainable business success. This project aims to implement a robust CRM system that goes beyond traditional CRM, fostering meaningful relationships and enhancing customer satisfaction. This project is tailored for service-center organizations. In this initiative, we're focused on enhancing how we interact with our customers to provide better services and build lasting connections. By utilizing this tool, we aim to personalize customer experiences, improve service efficiency, and strengthen overall customer satisfaction. Apart from enhancing between customers and service providers this SaaS also Provides quick quote service which allows the public to receive accurate and competitive quotes tailored to their specific device repair needs.

* 1. **OVERVIEW**

To develop a multi-service web application that focuses on revolutionizing customer relationships through a comprehensive CRM system and a Quick Quote Service. The CRM system aims to foster long-lasting connections that can be utilized in Service Hubs and the customers. This revolutionizes the customer experience by providing a seamless and efficient way to improve customer experience while retaining existing customers. Our CRM system is equipped with an extensive array of features designed to meet diverse business needs and enhance operational efficiency which includes automated communication via SMS or Email, Service Tracking for Customers, E-Billing, business performance Metrics, etc. Ultimately, the project aims to increase Customer satisfaction and retention thereby driving sustainable business growth. Quick quotes service brings leads for service providers by enabling Leads to turn into Customers by requesting quotes for device repair.

* 1. **PURPOSE OF THE PROJECT**

This system aims to enhance how we interact with our customers to provide better services and build lasting connections with Customers. Our goal is to implement a user-friendly CRM system that simplifies communication along with lead generation capability through Quick quotes service which benefits both the service providers and device repair seekers. This helps

to store and organize customer information at one location which helps in easy access to information and organization this also enables users to access data easily regardless of the Interacting device. Enhanced communication capabilities are facilitated through SMS and email services, enabling seamless interaction with customers. Reliable reporting of business performance metrics is provided, Automated workflows are triggered by specific events, optimizing operational efficiency. Vendor contact management functionality ensures smooth collaboration with external partners. Gathering customer feedback post-service delivery allows for continuous improvement. Real-time updates on service timelines and chat support further enhance customer satisfaction.

**1.3 MOTIVATION**

The driving force behind our SaaS application is the aspiration to unite service providers and customers within a single platform. Our motivation lies in creating a space of convenience, efficiency, and connectivity in service interactions. With the introduction of our Quick Quote service, we empower the public to effortlessly discover the best services within their region. Additionally, our platform facilitates collaboration among service providers, enabling the exchange of donor parts for electronic repair. By integrating these functionalities into one SaaS application, our goal is to streamline processes, enhance accessibility, and foster meaningful connections within the service ecosystem.

**CHAPTER 2: LITERATURE SURVEY**

The literature survey presented here offers a thorough examination of existing research, presenting valuable insights into the landscape of Customer Relationship Management (CRM) implementation within service centers. It provides a comprehensive understanding of the current state of knowledge, prevalent trends, challenges, and potential future directions in this domain. Serving as the cornerstone of our CRM project, this survey equips us with a roadmap to navigate through the complexities of CRM systems in service centers, enabling us to capitalize on existing knowledge and address pertinent issues effectively. By synthesizing and critically analyzing relevant literature, we aim to identify key strategies, methodologies, and areas for improvement, thus laying the groundwork for the development and implementation of a customer-centric CRM solution tailored to service centers' specific needs.

**2.1 Customer Relationship Management Research from 2000 to 2020: An Academic Literature Review and Classification**

**2.1.1 DESCRIPTION**

CRM research in recent years, particularly focusing on areas like data mining, CRM software, technology management, knowledge management, artificial intelligence, and social CRM. The review also discusses key findings from selected research articles, including frameworks for dealing with unprofitable customers, the impact of E-CRM on bank-customer relationships, the importance of customer satisfaction, and the role of CRM technology in customer engagement initiatives. Furthermore, the literature review delves into essential aspects of CRM, such as the stages of CRM (initiation, maintenance, termination), factors influencing customer involvement in new relationships with banks, the association between CRM technology and customer engagement initiatives, and the direct impact of CRM on organizational performance. It also addresses critical factors for successful CRM implementation, the significance of customer loyalty in the insurance industry, and the importance of trust and decentralization in blockchain technology. Additionally, the study emphasizes the need for a balanced approach involving people, process, and technology for successful CRM system implementation, the positive effects of commitment and loyalty programs on customer retention, and the necessity of building relationships as a fundamental strategy for enterprises. Overall, the literature review provides valuable insights into the evolving landscape of CRM research, highlighting key trends, challenges, and strategies for effective CRM implementation across various industries and countries.

**2.1.2 ADVANTAGES & DISADVANTAGES**

**The advantage** of the paper is its meticulous classification of research articles based on various parameters like publication year, sources of research articles, country-wise distribution, types of studies, and data collection methods. This detailed categorization allows for a structured and systematic understanding of the CRM research landscape over two decades, aiding researchers and practitioners in identifying trends and gaps in the field. Additionally, the paper's focus on key findings from selected research articles, such as the impact of CRM technology on customer engagement and the factors influencing successful CRM implementation, provides practical insights for businesses looking to enhance their customer relationships.

**The disadvantage** of the paper could be the limited depth of analysis on certain aspects of CRM research. While the classification of articles based on publication year, sources, and data collection methods is thorough, a more in-depth discussion on the implications of the findings or a critical analysis of the methodologies used in the selected research papers could have added further value to the review. Furthermore, the paper could benefit from a more extensive exploration of emerging trends in CRM research beyond the identified categories, such as the integration of AI and machine learning in CRM systems or the impact of evolving customer behaviors on CRM strategies. Despite these limitations, the paper serves as a valuable resource for understanding the evolution and current state of CRM research, offering a foundation for future studies in the field.

**2.2 A Literature Review on Customer Relationship Management**

**2.2.1 DESCRIPTION**

The study examines the literature on Customer Relationship Management (CRM), with a particular emphasis on CRM's effect on client satisfaction and customer loyalty. CRM is a set of methods used by businesses to manage and grow their customer base. Customer research in the CRM database can develop new approaches to leading business strategies. Analytical CRM analyses customer information or interactions using various data mining techniques. CRM optimism was the primary focus of academics and professionals working in information systems. This research to achieve fierce competition in the banking sector, this paper provides an analysis of CRM as well as new research. Competition is constantly waged to attract the greatest number of customers possible taking into account poorly differentiated banking deals.

**2.2.2 ADVANTAGES & DISADVANTAGES**

Customer Relationship Management (CRM) paper offers numerous advantages that can significantly benefit businesses. Firstly, CRM facilitates departmental integration, allowing organizations to consolidate data from various departments, leading to improved customer service through a comprehensive view of customer interactions. Secondly, CRM enhances customer service by serving as a vital tool for auditing, training, and monitoring customer needs, enabling businesses to better understand and provide personalized services to their clients. Lastly, CRM empowers sales and marketing strategies by enabling sales teams to track sales pipelines, monitor performance, and optimize sales efforts, ultimately enhancing productivity and customer engagement.

The paper's **disadvantage** might be that one significant challenge is data recording issues, where ensuring accurate and consistent data entry can be a considerable hurdle in CRM implementation. Additionally, CRM systems require ongoing maintenance and costly upgrades to remain effective and up-to-date, posing a financial burden on businesses. Moreover, integrating CRM with other information management systems can be complex and may necessitate specialized expertise, adding to the challenges of CRM implementation. Training employees on CRM systems and processes is also a time-consuming and resource-intensive task that businesses need to address to ensure successful CRM utilization.

**2.3** [**The Importance of Customer Relationship Management Systems to Business Management and Marketing Strategie**](https://www.emerald.com/insight/content/doi/10.1108/978-1-83753-686-320241008/full/html)**s**

**2.3.1 DESCRIPTION**

This chapter delves into the widespread adoption and impact of Customer Relationship Management (CRM) systems across various industries and organizational sizes. It explores how CRM technologies have become integral to business management and marketing strategies, contributing to the democratization of CRM solutions worldwide. The author presents an empirical study aimed at understanding the relationship between annual investments in CRM solutions and annual net income in a sample of companies. Using time series analysis and autoregressive integrated moving average modeling, the researcher examines data from 10 companies spanning different industries and countries over 20 years, revealing insights into the predictive capabilities of CRM investment on company income.

**2.3.2 ADVANTAGES & DISADVANTAGES**

The **advantage** of widespread adoption of Customer Relationship Management (CRM) systems across various industries and organizational sizes has brought about several notable advantages. Firstly, CRM solutions offer predictive insights into business performance, allowing companies to make informed decisions regarding resource allocation and strategic planning. This predictive capability enables businesses to anticipate customer needs and market trends, facilitating proactive rather than reactive approaches to business management. Additionally, the versatility of CRM systems ensures that they can be seamlessly integrated into organizations of all sizes, from large corporations to small- and medium-sized enterprises, fostering widespread adoption and utilization. Furthermore, by centralizing customer data and streamlining communication channels, CRM systems enhance efficiency and productivity within organizations, leading to improved customer satisfaction and loyalty. Moreover, CRM technologies enable businesses to personalize interactions with customers, thereby fostering stronger relationships and increasing customer retention rates. Overall, the advantages of CRM systems extend beyond mere transactional processes, empowering businesses to foster long-lasting connections with customers, drive sustainable growth, and maintain a competitive edge in today's dynamic market landscape.

Despite their numerous advantages, there are some **disadvantages** like implementation costs can be significant, encompassing expenses such as software licensing, training, and customization. Moreover, concerns regarding data security and privacy compliance arise due to the handling of sensitive customer information within CRM systems. Integrating CRM into existing organizational processes and IT infrastructure may pose challenges, requiring careful planning and coordination to ensure seamless functionality. Additionally, while CRM systems offer predictive insights, they may not capture all factors influencing business performance, limiting the comprehensiveness of analyses. These challenges underscore the importance of strategic planning and ongoing maintenance to maximize the effectiveness of CRM implementations while mitigating associated risks.

**2.4 SUMMARY OF LITERATURE SURVEY**

| **S.NO** | **TITLE & AUTHOR** | **DESCRIPTION** | **PROS** | **CONS** |
| --- | --- | --- | --- | --- |
| **1.** | Customer Relationship Management Research from 2000 to 2020: An Academic Literature Review and Classification  Priyanka Meena and Praveen Sahu | Recent CRM research explores data mining, CRM software, AI, and social CRM. Key findings include managing unprofitable customers, E-CRM's impact on bank-customer relationships, and the importance of satisfaction and engagement. | 1. Meticulous classification aids in understanding CRM research trends.  2. Provides practical insights for enhancing customer relationships. | 1. Limited depth of analysis on certain aspects.  2. Needs more exploration of emerging CRM trends beyond identified categories. |
| **2.** | A Literature Review on Customer Relationship Management  Sumitha K. | CRM's influence on satisfaction and loyalty, emphasizing analytical techniques for strategic development. Academic and professional interest in CRM enhances competitiveness, particularly in sectors like banking, driving ongoing research. | 1. Facilitates departmental integration for a comprehensive view of customer interactions.  2. Enhances customer service through auditing, training, and personalized services. | 1. Data recording issues pose challenges for accurate and consistent data entry.  2. Requires ongoing maintenance, costly upgrades, and integration complexities, adding to implementation challenges. |
| **3.** | [The Importance of Customer Relationship Management Systems to Business Management and Marketing Strategie](https://www.emerald.com/insight/content/doi/10.1108/978-1-83753-686-320241008/full/html)s  Matos-Lopez, Luis | This paper provides a widespread adoption and impact of CRM systems across industries, emphasizing their integration into business management and marketing strategies. It presents an empirical study analyzing the relationship between CRM investments and company income. | 1. CRM enables predictive insights for informed decisions.  Versatile integration enhances efficiency and customer satisfaction. | 1. High implementation costs and data security concerns.  2. Integration challenges require careful planning and maintenance. |

**Table 2.4:** Summary of Literature Survey

**CHAPTER 3: PROPOSED SYSTEM**

**3.1 PROPOSED SYSTEM**

Our proposed system is a Multi-Tenant Software as a Service (SaaS) application that integrates a comprehensive CRM system alongside a quick quote Service. This Multi-Tenant architecture is designed to empower service centers, by providing isolated instances of the CRM system while sharing common services and resources. This innovative approach ensures that each service center can operate independently within its environment.

The primary goal of the CRM component is to enhance the operational efficiency of IT Service Center Organizations. By centralizing customer data and automating routine tasks, the CRM system enables service centers to deliver superior customer service and support. Furthermore, it facilitates seamless collaboration between businesses by serving as a platform for connecting service centers. This unified approach brings all service centers under one roof. In addition to the CRM system, our proposed system features a quick quote service, which revolutionizes how customers find and engage with competitive services. This service acts as a powerful tool for users, allowing them to quickly obtain personalized quotes from a wide range of service providers within their region.

For service providers, the Quick Quote Service offers a unique opportunity to showcase their offerings and compete effectively in the market. By providing a platform for service providers to submit competitive quotes tailored to each customer's requirements, the Quick Quote Service helps them attract new customers and expand their business reach.

In summary, our Multi-Tenant SaaS application represents a transformative solution for IT Service Center Organizations, offering a powerful CRM system and Quick Quote Service that drive business growth.

**3.2 ADVANTAGES OF PROPOSED SYSTEM**

**1. Efficiency:** By offering a Multi-Tenant SaaS application, This system allows service centers to operate independently while sharing common resources. This enhances operational efficiency by using less Hardware and reducing duplication of efforts.

**2. Scalability:** The Multi-Tenant architecture enables seamless scalability, allowing the system to accommodate the growth of service center organizations without compromising performance or user experience.

**3. Data Security:** Each service center operates within its isolated environment, ensuring data security and privacy. This architecture minimizes the risk of unauthorized access and data breaches, fostering trust among users and customers with the reassuring security of Django.

**4. Enhanced Customer Experience:** The integrated CRM system centralizes customer data and communication processes, enabling service centers to deliver personalized and efficient customer service. This enhances the overall customer experience and fosters long-term relationships.

**5. Transparency and Competition:** The Quick Quote Service facilitates transparent and competitive pricing for services, it enables service providers to showcase their offerings and compete effectively in the market, driving innovation and quality improvement.

**6. Streamlined Operations:** By providing a centralized platform for service management and communication, your system simplifies operations for service centers. This leads to improved productivity, reduced overhead costs, and smoother workflow management.

**7. Collaboration and Partnerships:** The CRM system serves as a platform for connecting service centers and fostering collaborations. This enables service centers to share resources, expertise, and best practices, leading to mutual benefits and growth opportunities.

**8. Business Growth:** Overall, your proposed system facilitates business growth by optimizing operations, enhancing customer satisfaction, and fostering collaboration within the service ecosystem. It positions service centers for success in today's competitive market landscape while driving sustainable growth in the long term.

**9. Automated Communication:** Automated SMS and email functionalities that provide customers with timely updates and service notifications. By leveraging automation, the system sends personalized messages to customers, keeping them informed about service updates.

**10. Vendor Management:** Our CRM system includes robust vendor management capabilities, allowing users to efficiently track received items, payments, returns, and outstanding balances for each vendor. This feature streamlines vendor interactions improves accuracy, and enhances financial management within the system, ultimately contributing to better vendor relationships and operational efficiency.

**11. Reliable reporting of Business Performance:** It monitors key metrics such as customer retention, service success rates, and the source of customers whether it is through quick quotes or not. This involves tracking the number of returning customers and distinguishing between services completed successfully versus those that failed. By analyzing these metrics, service centers can gain valuable insights into customer behavior, optimize service delivery processes, and improve overall customer satisfaction.

**12. Customer Feedback post-delivery of service:** Our CRM system enables businesses to collect and store customer feedback following the completion of service. This feature provides valuable insights into customer satisfaction levels, allowing businesses to identify areas for improvement, address concerns promptly, and enhance overall service quality. By leveraging customer feedback data, businesses can make informed decisions, foster stronger customer relationships, and drive continuous improvement initiatives.

**13. Service-Timeline and real-time updates and Chat support:** Incorporating a service timeline and real-time updates, alongside chat support, enhances customer service. customers can track service progress visually, resulting in improved customer satisfaction and Transparency.

**3.3 SYSTEM REQUIREMENTS**

The system necessitates a cloud platform, such as AWS (Amazon Web Services), that supports Docker containers and is equipped with a Linux-based operating system. This compatibility enables the deployment and management of the software application within Docker containers on a Linux-based environment, ensuring scalability, stability, security, and efficient resource utilization. By meeting these requirements, the software application can be effectively deployed and operated within a cloud environment, leveraging Docker containers and a Linux-based operating system for optimal performance and reliability.

**3.3.1 SOFTWARE REQUIREMENTS**

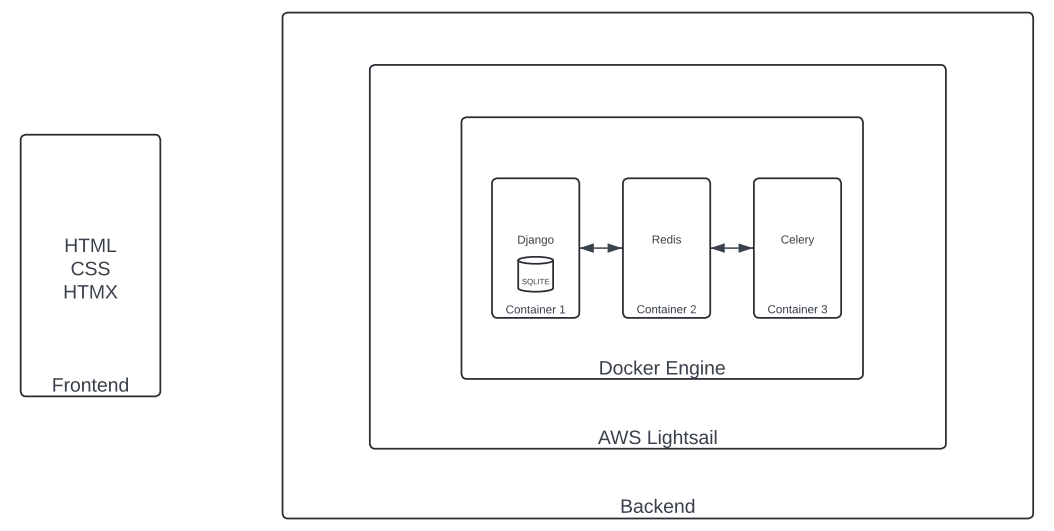
* **Operating System:** Compatible with major operating systems such as Linux, Windows, or macOS.
* **Programming Languages:** Python is used for developing the backend along with HTML, and CSS for developing the web interface.
* **Frameworks and Libraries:** Django, Redis, Celery, BotoS3, Geopy, django\_celery\_results, Django\_channels, Daphne Server, HTMX, Whitenoise. docker-compose.
* **Database Management System:** SQLite, Postgres, Redis
* **User Interface Requirements:** Browser Chrome or Microsoft Edge.
* **Network Connectivity:** Stable internet connection to ensure seamless communication between the web interface and backend processing components with JavaScript enabled in the Browser.
* **Containerization:** Containerization using Docker technology, we encapsulate our application components into three distinct containers: Django, Celery, and Redis. The Django container hosts our web application, providing a robust framework for handling user requests and serving dynamic content. Concurrently, the Celery container manages asynchronous tasks, ensuring efficient background processing and task scheduling. Finally, the Redis container serves as a high-performance caching and message broker system, enhancing the responsiveness and reliability of our application.

**3.3.2 HARDWARE REQUIREMENTS**

* **Processor:** i3 Dual Core CPU for PC / T2.micro (1vcpu, 1 GB RAM) for Cloud
* **Processor Speed:** 2.4GHZ
* **RAM:** 4 GB RAM in Cloud
* **Storage:** 1 GB

**CHAPTER 4: SYSTEM DESIGN**

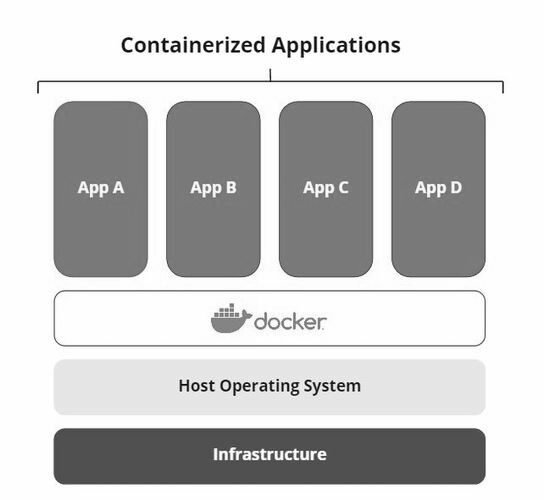
* 1. **PROPOSED SYSTEM ARCHITECTURE**



**Fig 4.1:** Proposed System Architecture

Our system architecture adopts a modern and modular approach, leveraging Docker containers to encapsulate and deploy our application components. The backend of our application comprises three Docker containers: Django, Redis, and Celery.

* The Django container serves as the backbone of our web application, hosting the server-side logic, handling user requests, and generating dynamic content.
* The Redis container acts as a high-performance caching and message broker system, enhancing the responsiveness and reliability of our application by storing frequently accessed data and facilitating communication between Celery and Django.
* The Celery container manages asynchronous task processing, enabling efficient background job execution and task scheduling to enhance the overall performance and scalability of our application.

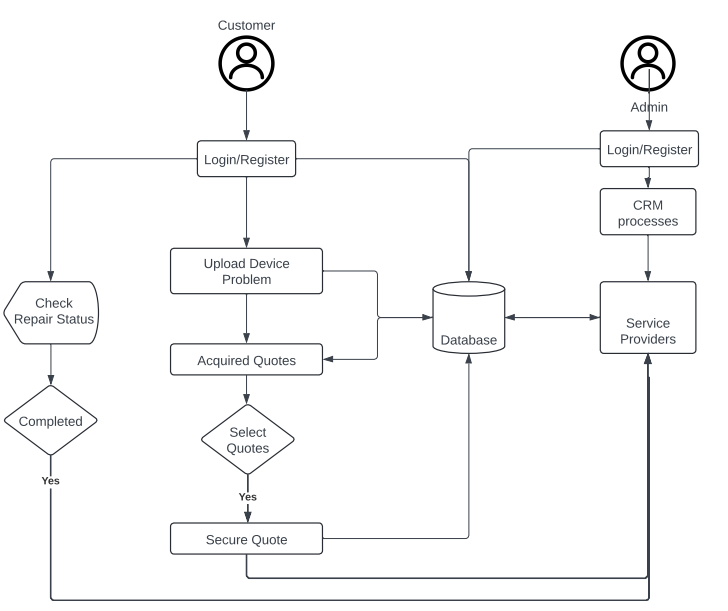
These containers are deployed on AWS Elastic Beanstalk, and provide a scalable and cost-effective cloud infrastructure for hosting our application. 

**Fig 4.1.2** Representing Docker containers

These containers are run with a Docker Compose file, which orchestrates and manages the deployment of multiple containers. Docker Compose allows us to define a multi-container application in a single file, specifying the configuration and relationships between containers. With Docker Compose, we can easily define the services, networks, and volumes required for our application, ensuring consistency and ease of management.

On the front end, we utilize HTML, CSS, and the HTMX library to create a responsive and interactive user interface. HTMX enables seamless AJAX calls, enhancing the user experience by enabling dynamic content updates without full page reloads.

Overall, our proposed system architecture combines the flexibility and scalability of Docker containers with the reliability and performance of AWS Elastic Beanstalk, delivering a robust and efficient solution for deploying and managing our web application.

****

**Fig 4.1.3:** Block Diagram of System Architecture

The system is developed by using Django a Python Web framework that is scalable and used for Backend Logic. For Database Operations, Django ORM helps to communicate and query with Databases. In this project, SQL-liteRelational-Database is used to store Organization information and it is the default database. In Django, views encapsulate the application logic, particularly in Function-Based Views (FBVs), where each view function represents a specific endpoint and handles incoming requests by processing data and rendering appropriate responses. URL resolution maps these views to corresponding URLs, allowing Django to route incoming requests to the appropriate view function based on the requested URL pattern. Django DTL renders our Templates Dynamically with the data provided by the Database. HTMX makes AJAX requests at the front end for a better user experience and reduces Payload for server-side rendering.

**4.2 PROPOSED SYSTEM MODULES**

This project consists of the following modules:

* Customer Contact Storage and Management
* Service Ticket Management and Tracking
* Quote Generation
* Automated Communication
* Reporting and Analytics

**4.2.1. Customer Contact Storage and Management**

This module focuses on capturing and managing customer information, including contact details, service history, and feedback. It enables service centers to maintain a comprehensive database of customers and personalize interactions based on their needs and preferences.

**4.2.2 Service Ticket Management and Tracking**

This module handles the lifecycle of service Tickets, from initialization to completion. It includes features for logging service Tickets, assigning tasks to service technicians, tracking progress, and auto service status updates to customers for better customer experience. Also

Organized ticket management allows for better tracking of ticket status and progress, enabling service providers to monitor the lifecycle of each ticket from submission to resolution.

**4.2.3 Quote Generation**

The Quote Service captures device repair requests from customers or the public, visualizing these inquiries to service providers. Service providers then provide details, which are compiled into quotes and sent back to the customer or device repair seeker.

**4.2.4 Automated Communication**

Automated Communication: Our system incorporates automated communication features, utilizing both SMS and email functionalities. SMS communication is employed to swiftly deliver ticket IDs upon device submission and to remind customers when their device repair is completed and ready for pickup. Meanwhile, email communication is utilized specifically for sending invoices. By seamlessly integrating SMS and email communication, we ensure efficient and timely delivery of essential information to our customers, enhancing their experience throughout the service process.

**4.2.5 Reporting and Analytics**

The reporting and analytics module gathers data from various sources within the CRM system and generates insights into customer behavior, service performance, and operational efficiency. It enables service centers to make data-driven decisions to improve processes and customer satisfaction.

**4.3 UML DIAGRAMS**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form, UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

* Usecase Diagram
* Class Diagram
* Sequence Diagram
* Activity Diagram

**GOALS:**

The Primary goals in the design of the UML are as follows:

1.Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.

2.Provide extendibility and specialization mechanisms to extend the core concepts.

3.Be independent of programming languages and development process.

4.Provide a formal basis for understanding the modeling language.

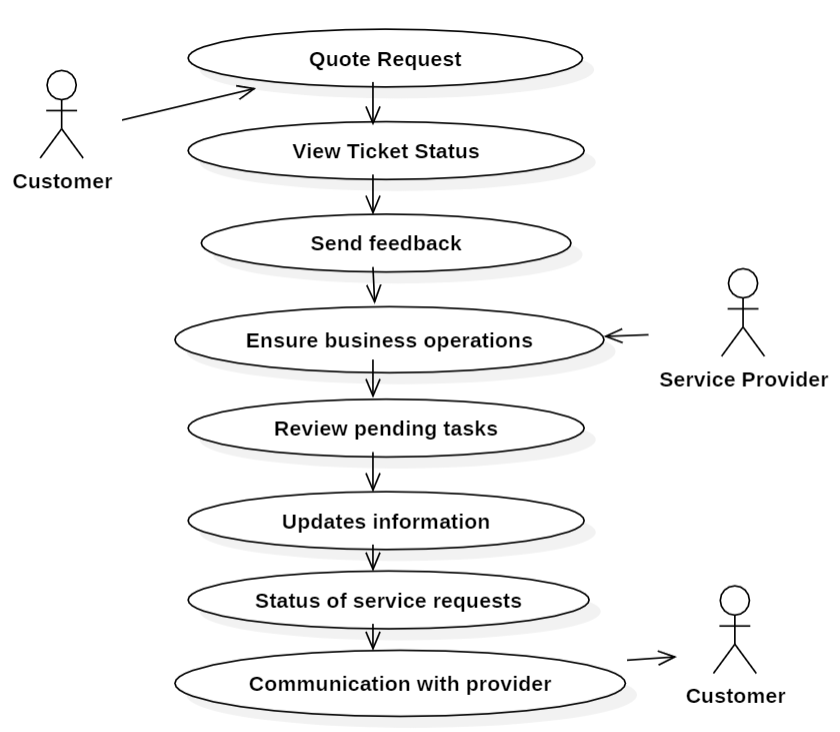
5.Encourage the growth of OO tools market.

6.Support higher level development concepts such as collaborations, frameworks, patterns and components.

7.Integrate best practices.

**4.3.1 USE CASE DIAGRAM:**

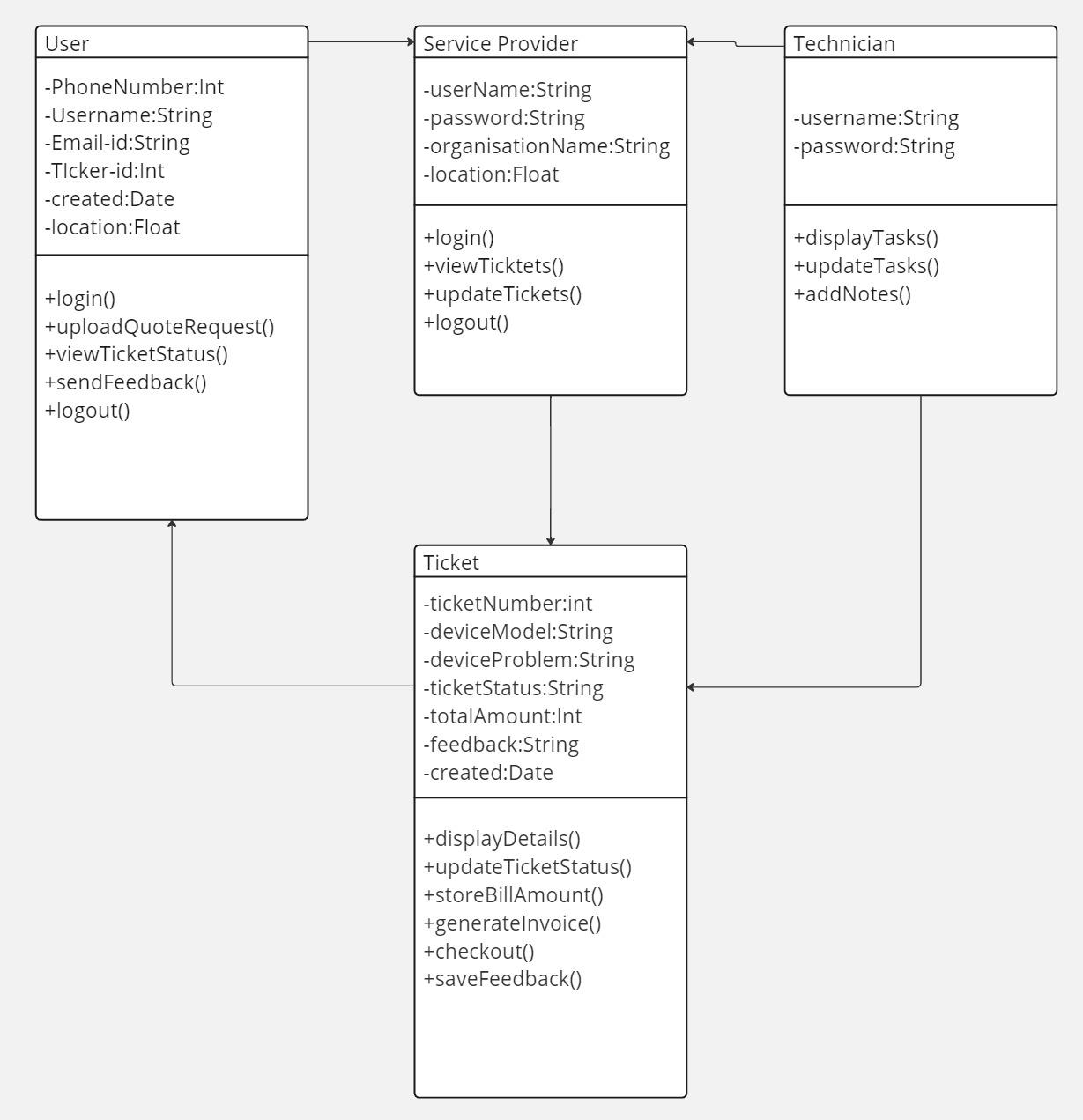
A Use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. The roles of the actors in the system can be depicted.



**Fig 4.3.1:** UML Use Case Diagram

**4.3.2 CLASS DIAGRAM:**

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

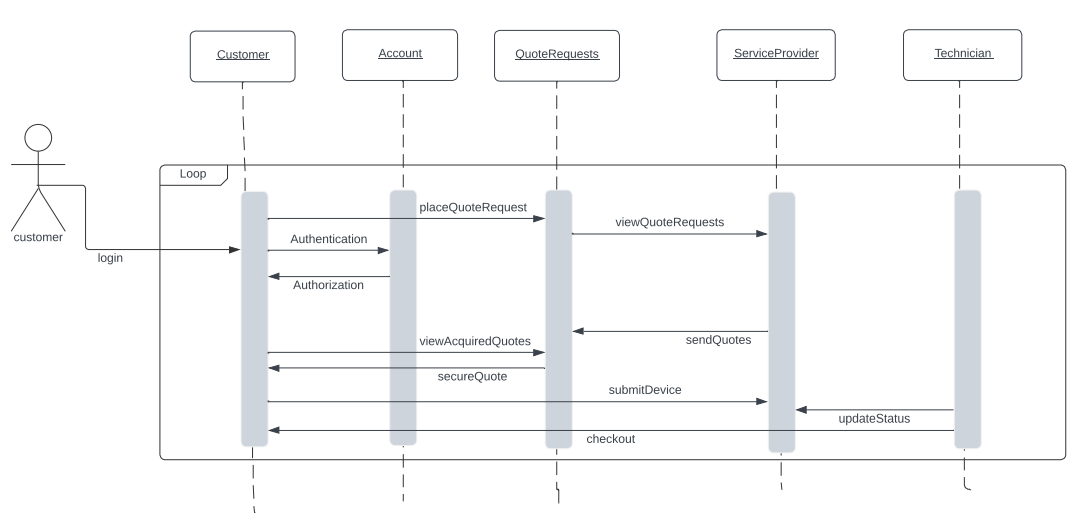




**Fig 4.3.2:** UML Class Diagram

**4.3.3 SEQUENCE DIAGRAM:**

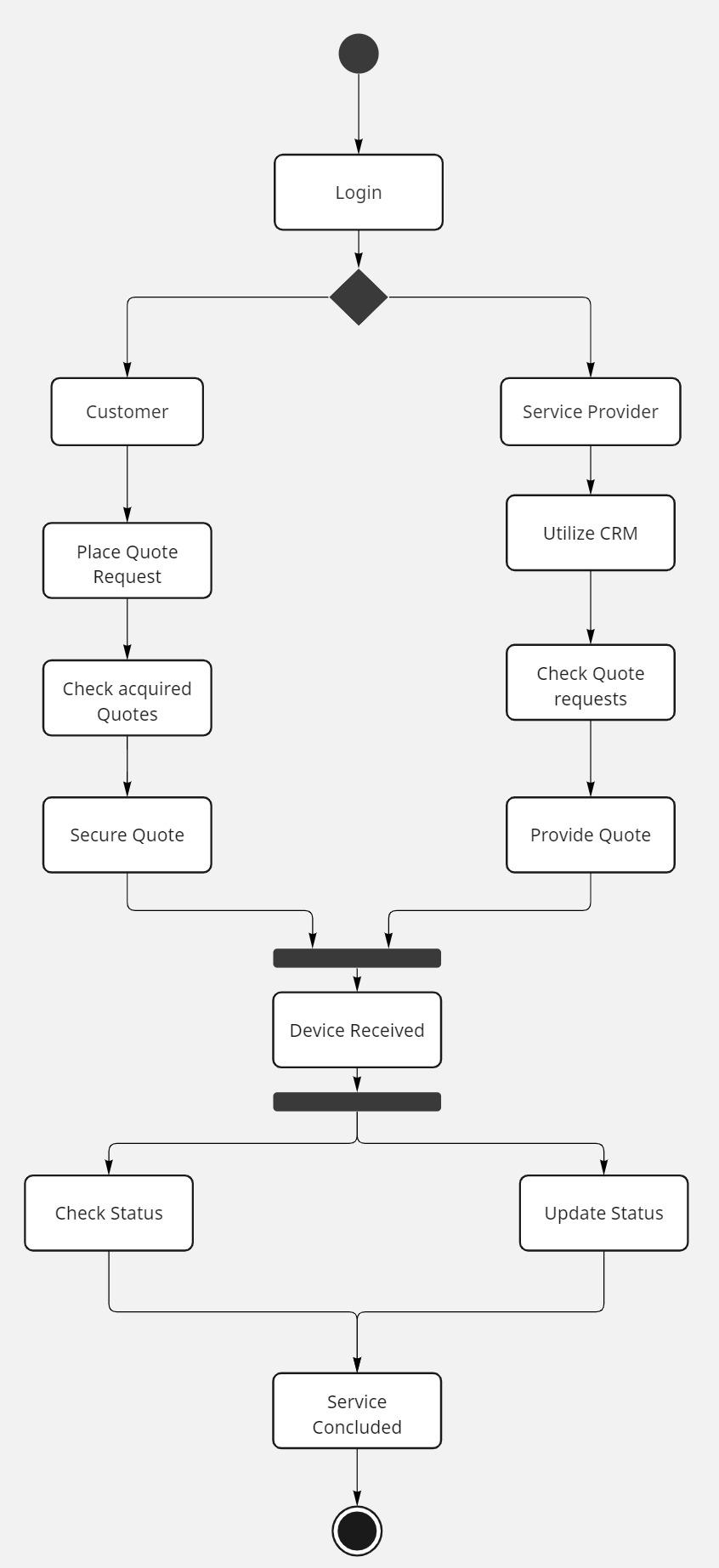
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.



**Fig 4.3.3:** UML Sequence Diagram

**4.3.4 ACTIVITY DIAGRAM:**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration, and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



**Fig 4.3.4:** UML Activity Diagram

**CHAPTER 5: IMPLEMENTATION**

Our implementation journey began with the development of our application locally, where we meticulously crafted its features and functionalities. This initial phase involved extensive work to include dependencies and configurations, ensuring smooth operation within our development environment. However, as our project progressed towards deployment onto the cloud for production, we encountered the complexity of configuring our application to meet production standards. Despite the challenges, our team navigated through intricacies, recognizing the necessity of containerization for streamlined deployment. This realization prompted the adoption of Docker, revolutionizing our deployment process and ensuring consistency across environments. Deployment on a cloud platform, particularly AWS Elastic Beanstalk, marked the culmination of our implementation efforts. With the application running 24/7 on the cloud, we achieved the desired service availability and scalability, meeting the demands of our users with reliability and efficiency. This final phase symbolized the successful transition of our project from development to production.

**5.1 TECHNOLOGIES**

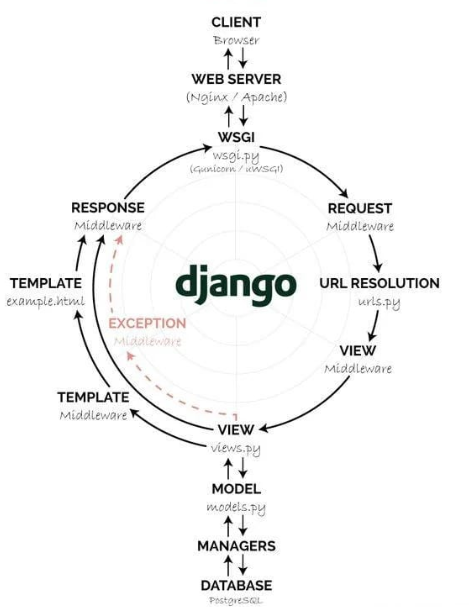
Below are the technologies that collectively comprise the entire application stack, each playing a crucial role in delivering a seamless user experience. Through their combined efforts, they form the backbone of our innovative solution, powering its functionality and performance.

* **PYTHON:**

Python stands as the cornerstone of our SaaS application, chosen for its unparalleled contributions to web development and SaaS solutions. Its simplicity, coupled with its extensive ecosystem of libraries and frameworks, accelerated our development process, empowering us to swiftly bring innovative features to our Application. Python's versatility extends seamlessly to SaaS applications, where its asynchronous programming capabilities and cloud compatibility ensure scalability and responsiveness. Additionally, Python's vibrant community fosters a collaborative environment, providing invaluable support and resources for our development endeavors. In essence, Python's robustness, scalability, and community support align perfectly with our vision for building a cutting-edge SaaS platform that meets the dynamic needs of end users.

* **DJANGO:**

Django serves as the foundational framework powering the backend infrastructure of our SaaS application. Renowned for its "batteries-included" philosophy, Django offers a comprehensive set of tools and functionalities out of the box, enabling rapid development of complex web applications. Its built-in features such as an ORM (Object-Relational Mapping) system, user authentication, and admin panel streamline the development process, allowing us to focus on implementing unique business logic and features. Django's adherence to the DRY (Don't Repeat Yourself) principle and convention-over-configuration approach promotes code reusability and maintainability, facilitating scalability as our application grows.



**Fig 5.1.1** Request Response Cycle in Django

Furthermore, Django's robust security features, including protection against common web vulnerabilities like SQL injection and CSRF (Cross-Site Request Forgery), ensure the integrity and confidentiality of our users' data. Its powerful templating engine and support for internationalization and localization make it suitable for building multilingual and accessible web applications, catering to a diverse global audience. Moreover, Django's scalability is demonstrated by its ability to handle high traffic loads and complex data processing tasks efficiently, making it an ideal choice for building mission-critical SaaS applications. Overall, Django's combination of simplicity, flexibility, and scalability aligns perfectly with our goals for delivering a robust and feature-rich SaaS platform to our users.

* **REDIS:**

Redis serves as a pivotal component within our tech stack, fulfilling multiple critical roles to enhance the functionality and performance of our SaaS application. Primarily utilized as a message broker, Redis facilitates seamless communication and coordination between various components of our application architecture. Its robust pub/sub (publish/subscribe) mechanism enables real-time messaging and event-driven workflows, allowing different parts of our application to communicate asynchronously and efficiently. Moreover, Redis plays a crucial role in data sharing between multiple Django applications within our ecosystem. By leveraging Redis as a high-performance, in-memory data store, we ensure fast and reliable access to shared data across distributed components. This not only enhances data consistency and coherence but also streamlines inter-process communication, enabling our applications to operate cohesively and efficiently. Furthermore, Redis's support for data structures like lists, sets, and sorted sets provides powerful primitives for implementing caching, session management, and other data-intensive operations. Its lightning-fast read and write operations make it ideal for handling high-throughput workloads and real-time analytics, ensuring optimal performance and responsiveness for our users. In summary, Redis serves as a versatile and indispensable component within our architecture, acting as both a message broker and a high-performance data store. Its seamless integration with Django and its ability to facilitate real-time communication and data sharing between distributed components make it an invaluable asset in our quest to deliver a robust and scalable SaaS solution.

* **CELERY:**

Celery plays a pivotal role in our tech stack, providing a powerful and efficient framework for running background tasks in our SaaS application. One of the primary uses of Celery in our architecture is to handle time-consuming tasks such as sending emails, which typically require an average of 4 seconds to complete. By offloading these tasks to Celery, we ensure that the Django request-response cycle remains unaffected by resource-intensive operations, thereby enhancing the overall responsiveness and scalability of our application.In summary, Celery serves as a cornerstone of our background task processing infrastructure, enabling us to execute time-consuming tasks such as sending emails asynchronously and efficiently. Its seamless integration with Django, robust task execution model, and extensible architecture make it an invaluable component in our quest to deliver a responsive, scalable, and reliable SaaS solution.

**Celery Beat:** We employ Celery Beat, a scheduler provided by Celery, to generate and update business metrics daily. By configuring Celery Beat to execute scheduled tasks at specific intervals, such as every morning, we ensure that the metrics are generated and refreshed with the latest data without manual intervention. This automated process enhances efficiency and accuracy, allowing stakeholders to access up-to-date business insights at the start of each day. Additionally, leveraging Celery Beat enables us to define complex scheduling patterns and manage task execution with ease, providing flexibility and reliability in maintaining our business metrics infrastructure.

* **DOCKER:**

Docker plays a crucial role in our SaaS application infrastructure, providing a lightweight and efficient containerization platform that enables us to package, deploy, and manage our application components with ease. At the core of our Docker deployment are three containers: Django, Redis, and Celery, each serving a critical function in our application architecture. Without Docker, the task of deploying our application to production would be significantly more challenging, if not impossible, due to the complex dependencies and configurations required to run our services consistently across different environments. By containerizing our application components with Docker, we benefit from several advantages that streamline our development and deployment processes. Firstly, Docker ensures consistency across development, testing, and production environments, eliminating the "it works on my machine" problem and enabling seamless collaboration among team members. Each container encapsulates all the dependencies and configurations required to run the respective service, providing a self-contained and portable environment that can be easily replicated and deployed anywhere. Additionally, Docker simplifies the management and orchestration of our application services through tools like Docker Compose With Docker Compose, we can define and manage multi-container applications using a single YAML file, facilitating the configuration of complex service dependencies and deployment workflows. In summary, Docker's containerization technology plays a crucial role in our application infrastructure, enabling us to deploy and manage our Django, Redis, and Celery services efficiently.

* **DJANGO CHANNELS:**

We've utilized Django Channels to seamlessly integrate WebSocket functionality into our Django application, enabling real-time communication and dynamic updates between clients and servers. By leveraging Django Channels, we've extended the capabilities of our application beyond traditional HTTP requests, allowing for bidirectional communication and asynchronous event handling.

* **DAPHNE:**

Daphne is an HTTP, HTTP2, and WebSocket protocol server for ASGI and ASGI-HTTP, developed as part of the Django Channels project. In the context of Django, Daphne serves as the interface between Django Channels and the web server, facilitating the handling of WebSocket connections and asynchronous HTTP requests. One of the key features of Daphne is its support for ASGI (Asynchronous Server Gateway Interface), which allows Django applications to handle long-lived connections, such as WebSocket connections, alongside traditional HTTP requests. This enables developers to build real-time features, such as chat applications and live updates, within their Django projects. Daphne is designed for high performance and scalability, capable of handling thousands of concurrent connections efficiently. It supports both HTTP and WebSocket protocols, making it suitable for a wide range of applications requiring real-time communication between clients and servers.

* **AWS ELASTIC BEANSTALK:**

We leverage AWS Elastic Beanstalk to deploy our application using Docker, streamlining the deployment process and ensuring scalability, reliability, and ease of management. By utilizing Docker, we encapsulate our application and its dependencies into containers, providing consistency across different environments and simplifying deployment across AWS infrastructure. Elastic Beanstalk automates the provisioning of resources, load balancing, auto-scaling, and monitoring, allowing us to focus on developing our application while ensuring it runs smoothly in a managed environment. This integration of Elastic Beanstalk with Docker enables us to deploy and manage our application efficiently, while taking advantage of AWS's robust infrastructure and services.

* **HTMX:**

HTMX is a lightweight JavaScript library that enables seamless AJAX interactions in web applications, enhancing user experience and interactivity. By adding dynamic behavior to HTML elements, HTMX allows developers to update parts of a web page without full-page reloads, resulting in faster and smoother interactions for users. Its simplicity and ease of integration make it a valuable tool for enhancing the responsiveness and usability of web applications, particularly in scenarios where real-time updates and asynchronous operations are required. HTMX further provides the advantage of updating only the required parts on the client browser instead of the entire frontend, reducing bandwidth usage and improving performance. Additionally, HTMX eased our development process by providing a simpler alternative to complex development libraries such as React library. With HTMX, accomplishing similar functionality as React does require only 3 to 4 lines of code at the frontend, compared to the more involved setup and configuration required with React. This streamlined approach enabled us to focus on implementing features and delivering value to our users more efficiently. Overall, HTMX's compatibility with Django and its ability to simplify AJAX interactions make it an invaluable addition to our tech stack, empowering us to build dynamic and interactive web applications with ease.

* **HTMX WEB SOCKET EXTENSION:**

Using the HTMX WebSocket Extension, we've implemented a messaging feature that enables real-time communication between users. This functionality allows users to send and receive messages instantly without the need for page refreshes. By leveraging WebSocket technology, our messaging feature provides a seamless and responsive user experience, facilitating dynamic and interactive conversations between users. Additionally, the HTMX WebSocket Extension simplifies the implementation process, enabling us to seamlessly integrate real-time messaging capabilities into our web application with minimal effort.

* **BOTOS3:**

We utilize Boto3, a Python library, to communicate with AWS services seamlessly. By leveraging Boto3, we can programmatically interact with a wide range of AWS services, including Amazon S3, SNS and more. This allows us to automate tasks such as uploading and downloading files to and from Amazon S3 buckets, managing SNS service. Boto3 simplifies the integration of AWS services into our applications.

* **DJANGO CELERY RESULTS:**

Django Celery Results is an extension of Celery, a distributed task queue for Python, that provides support for storing task results in a persistent backend such as a database. This extension enables developers to track the status and outcome of asynchronous tasks executed with Celery, ensuring reliable task execution and facilitating error handling and monitoring.

**5.2 SOURCE CODE**

**Django.py:**

"""

Django settings for Servicejobs project.

Generated by 'django-admin startproject' using Django 4.2.

For more information on this file, see

https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/4.2/ref/settings/

"""

from pathlib import Path

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'django-insecure-ip6b)5q^6fr0k+kj2$9\_2xeksq\*lry3kn678mc4rf=lic1vwi='

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = ['\*'] # manage.py runserver 192.168.29.47:8000

# INTERNAL\_IPS = ["127.0.0.1"]

# Application definition

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'django\_celery\_results',

'django\_filters',

# "debug\_toolbar",

'baseapp',

'console',

'sendresponse',

'transactdb',

'connect',

]

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

'whitenoise.middleware.WhiteNoiseMiddleware',

# "debug\_toolbar.middleware.DebugToolbarMiddleware",

]

ROOT\_URLCONF = 'Servicejobs.urls'

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [BASE\_DIR / 'templates'],

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.debug',

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

],

},

},

]

WSGI\_APPLICATION = 'Servicejobs.wsgi.application'

# Database

# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

}

}

# Password validation

# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

# Internationalization

# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'Asia/Calcutta'

USE\_I18N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC\_URL = 'static/'

STATIC\_ROOT = BASE\_DIR / 'production-files'

STATICFILES\_DIRS = [BASE\_DIR / 'mystatic']

# Default primary key field type

# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

LOGIN\_URL = "login"

AUTH\_USER\_MODEL = "baseapp.MyCustomUserModel"

SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE = True

SESSION\_COOKIE\_AGE = 50400 # equivalent to 14 hours

# EMAIL CONFIG

EMAIL\_BACKEND = "django.core.mail.backends.smtp.EmailBackend"

EMAIL\_HOST = 'smtp.gmail.com'

EMAIL\_PORT = 587

EMAIL\_USE\_TLS = True

EMAIL\_HOST\_USER = 'zoltakconnect@gmail.com'

EMAIL\_HOST\_PASSWORD = 'xxxx xxxx xxxx xxxx'

# REDIS CONFIGURATION

CACHES = {

"default": {

"BACKEND": "django.core.cache.backends.redis.RedisCache",

"LOCATION": "redis://redis\_development\_container:6379",

}

}

# CELERY SETTINGS

CELERY\_BROKER\_URL= "redis://redis\_development\_container:6379/"

CELERY\_RESULT\_BACKEND = 'django-db'

CELERY\_CACHE\_BACKEND = 'default'

CELERY\_RESULT\_EXTENDED = True

# CELERY\_TASK\_TRACK\_STARTED = True

#CELERY\_TIMEZONE = "Asia/Calcutta"

#CELERY\_ENABLE\_UTC = True

**DjangoURLs.py:**

"""

URL configuration for Servicejobs project.

The `urlpatterns` list routes URLs to views. For more information please see:

https://docs.djangoproject.com/en/4.2/topics/http/urls/

Examples:

Function views

1. Add an import: from my\_app import views

2. Add a URL to urlpatterns: path('', views.home, name='home')

Class-based views

1. Add an import: from other\_app.views import Home

2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')

Including another URLconf

1. Import the include() function: from django.urls import include, path

2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

"""

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

# path("\_\_debug\_\_/", include("debug\_toolbar.urls")),

path('admin/', admin.site.urls),

path('', include('baseapp.urls')),

path('cs/', include('console.urls')),

path('sr/', include('sendresponse.urls')),

path('db/', include('transactdb.urls')),

path('tm/', include('team.urls')),

path('connect/', include('connect.urls')),

]

**asgi.py:**

"""

ASGI config for weebsockets project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see

https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/

"""

import os

from channels.auth import AuthMiddlewareStack

from channels.routing import ProtocolTypeRouter, URLRouter

from channels.security.websocket import AllowedHostsOriginValidator

from django.core.asgi import get\_asgi\_application

import chat.routing

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'weebsockets.settings')

django\_asgi\_app = get\_asgi\_application()

application = ProtocolTypeRouter({

"http": django\_asgi\_app,

'websocket': AllowedHostsOriginValidator(

AuthMiddlewareStack(URLRouter(chat.routing.websocket\_urlpatterns))

)

})

**models.py:**

from django.db import models

from django.contrib.auth.models import AbstractUser, Group

from django.contrib.sessions.models import Session

# Create your models here.

Status\_choices = (('pending', 'pending'), # modelValue, userReadableValue

('Ready', 'Ready'),

('Failed', 'Failed'),

('Delivered', 'Delivered'),

('Returned', 'Returned'),)

Type\_choices = (('adapter', 'adapter'), ('mobile', 'mobile'),

('printer', 'printer'), ('tablet', 'tablet'),

('smps', 'smps'), ('ups', 'ups'),

('motherboard', 'motherboard'), ('monitor', 'monitor'),

('laptop', 'laptop'), ('desktop', 'desktop'),

('audio', 'audio'), ('StorageMedia', 'StorageMedia'),

('DataRecovery', 'DataRecovery'), ('Toner/Catridge', 'Toner/Catridge'),

('GPU', 'GPU'), ('Part-Order', 'Part-Order'),

('?', '?'))

payment\_choices = (('Pending', 'Pending'),

('Paid', 'Paid'),

('N/A', 'N/A'),

('Deposit', 'Deposit'))

Engagement\_status = (('NA', 'Not Assigned'),

('Elite', 'Elite'),

('Regular', 'Regular'),

('Basic', 'Basic'),

('Caution', 'Caution'))

BillPaymentChoices = (('CASH', 'CASH'),

('CARD', 'CARD'),

('UPI', 'UPI'))

WarrantyPeriodChoices = (('Days', 'Days'),

('Months', 'Months'),

('Years', 'Years'))

class MyCustomUserModel(AbstractUser):

ORG = models.PositiveSmallIntegerField(null=True)

ACT = models.CharField(null=True, max\_length=6, choices=(('Master', 'Admin-level Access'),

('Slave', 'Employee-level Access'),

('Shared', 'Shared-level Access')))

class Profile(models.Model):

User = models.OneToOneField(MyCustomUserModel, on\_delete=models.CASCADE)

Phone = models.PositiveIntegerField(null=True)

Landline = models.CharField(max\_length=15, null=True)

Subscription = models.BooleanField(default=True)

Login\_Attempts\_Failed = models.IntegerField(default=0)

Organization = models.CharField(max\_length=80, null=True)

Location = models.CharField(max\_length=20, null=True)

Latitude = models.FloatField(null=True, blank=True)

Longitude = models.FloatField(null=True, blank=True)

def \_\_str\_\_(self):

return self.User.username

class Visitor(models.Model):

TargetUser = models.OneToOneField(MyCustomUserModel, on\_delete=models.CASCADE)

SessionKey = models.CharField(max\_length=50, null=True, blank=True)

class CustomerRegistration(models.Model):

id = models.SmallIntegerField()

CIDN = models.IntegerField(primary\_key=True)

CustomerName = models.CharField(max\_length=40)

Phone = models.PositiveIntegerField(null=True, blank=True)

Orders = models.PositiveIntegerField(default=0)

EngagementTier = models.CharField(max\_length=10, default='NA', choices=Engagement\_status,)

Created = models.DateTimeField(null=True, auto\_now\_add=True,)

CORG = models.PositiveSmallIntegerField(null=True)

# Fields During Data Restoration

# Created = models.DateTimeField(auto\_now\_add=False, null=True)

def \_\_str\_\_(self):

return self.CustomerName

class Job(models.Model):

id = models.SmallIntegerField()

SIDN = models.IntegerField(primary\_key=True)

SORG = models.PositiveSmallIntegerField(null=True)

Name = models.ForeignKey(CustomerRegistration, on\_delete=models.CASCADE, related\_name='ticket')

Problem = models.CharField(max\_length=30, null=True, blank=True, default='(empty)')

Status = models.CharField(max\_length=10, choices=Status\_choices, default="pending")

Payment = models.CharField(max\_length=7, choices=payment\_choices, default='Pending')

Type = models.CharField(max\_length=15, choices=Type\_choices, default='?')

Model = models.CharField(max\_length=15, null=True)

Price = models.IntegerField(default=0, blank=True)

Description = models.TextField(null=True, blank=True)

Phone2 = models.PositiveIntegerField(blank=True, null=True,)

RequireUpdate = models.BooleanField(default=False)

SN = models.CharField(max\_length=15, null=True, blank=True)

Deposit = models.IntegerField(default=0, blank=True)

Created = models.DateTimeField(auto\_now\_add=True)

Updated = models.DateTimeField(auto\_now=True)

def \_\_str\_\_(self):

return f"{self.SIDN} - {self.Model}"

# Fields During Data Restoration

# Created = models.DateTimeField(auto\_now\_add=False, null=True)

# Updated = models.DateTimeField(auto\_now=False, null=True)

class Bill(models.Model): # Note this just holds one Bill item not an Invoice

BSIDN = models.IntegerField(null=True)

BORG = models.PositiveSmallIntegerField(null=True)

Description = models.CharField(max\_length=80, default='empty')

Details = models.CharField(max\_length=300, null=True)

Amount = models.PositiveSmallIntegerField(default=0, blank=True)

Quantity = models.PositiveSmallIntegerField(default=1, blank=True)

Created = models.DateTimeField(auto\_now\_add=True, null=True)

def \_\_str\_\_(self):

return self.Description

class Invoice(models.Model):

ISIDN = models.PositiveSmallIntegerField(null=True)

Discount = models.PositiveSmallIntegerField(default=0)

Created = models.DateField(auto\_now\_add=True)

PaymentMode = models.CharField(max\_length=4, null=True, choices=BillPaymentChoices)

class ServiceWarranties(models.Model):

WORG = models.PositiveSmallIntegerField()

ServiceFor = models.OneToOneField(Job, on\_delete=models.CASCADE)

Includes = models.CharField(null=True, max\_length=255, blank=True)

Excludes = models.CharField(null=True, max\_length=255, blank=True)

Duration = models.PositiveSmallIntegerField()

DurationType = models.CharField(default='Months', max\_length=10, choices=WarrantyPeriodChoices)

EndDate = models.DateTimeField(null=True)

Created = models.DateTimeField(auto\_now\_add=True)

**App1.py:**

from django.contrib.auth import authenticate, login, logout

from django.contrib.auth.decorators import login\_required

from django.shortcuts import render, redirect

from django.views.decorators.csrf import csrf\_exempt

from django.core.paginator import Paginator

from django.utils import timezone

from baseapp.models import Job, CustomerRegistration, MyCustomUserModel, Visitor, Profile, Bill

from sendresponse.models import Response

from django.http import HttpResponse

from .filters import CustomerFilter

from datetime import datetime, timedelta

from dateutil.relativedelta import relativedelta

from django.contrib.sessions.models import Session

from django.db.models import Sum

import random

from baseapp.tasks import add, emailDispatchtask

from django.core.mail import send\_mail

# import boto3

# Create your views here.

customers\_queryset\_cache = CustomerRegistration.objects.all()

GATE\_rgs = False

storage = {

'stats': {

# 555 : {

# 'TC': customers\_queryset\_cache,

# 'PJ': jobs\_queryset\_cache.filter(Status='pending'),

# 'RJ': jobs\_queryset\_cache.filter(Status='Ready'),

# 'SC': jobs\_queryset\_cache.exclude(Status\_\_in=['Delivered', 'Returned']),

# }

},

# 555: Paginator(Job.objects.filter(SORG=555).order\_by('-Updated', '-id'), 20),

}

def loginPage(request):

context = {'loginStatus': False}

# User Authenticated already!

if request.user.is\_authenticated:

context['loginStatus'] = True

return render(request, 'baseapp/loginPage.html', context=context)

def authenticate\_user(request):

if request.method == 'POST':

# Input Validation

usr, pwd = request.POST['user-name'].strip(), request.POST['pass-word'].strip()

ERROR = None

if not usr:

ERROR = "Enter UserName"

elif not pwd:

ERROR = "Enter password"

elif not MyCustomUserModel.objects.filter(username=usr).exists():

ERROR = "INVALID USER"

elif not MyCustomUserModel.objects.get(username=usr).is\_active:

ERROR = 'Account Disabled'

def error\_response(message="Unknown Error"):

print(f'Login attempt failed input contained [{usr}]')

e\_response = render(request, 'baseapp/snippets/loginAttemptErrors.html', context={'errorMessage': message})

e\_response['HX-Retarget'] = '#login\_container'

e\_response['HX-Reswap'] = 'beforeend'

return e\_response

if ERROR:

return error\_response(ERROR)

user\_validated = authenticate(username=usr, password=pwd)

if user\_validated:

login(request, user\_validated)

print(f'user [{request.user}] logged-in')

# Allow one session per User ----------------------------------------------------------

new\_session\_key = request.session.session\_key

if hasattr(request.user, 'visitor'):

if Session.objects.filter(session\_key=request.user.visitor.SessionKey).exists():

# Discard previous session if exists

old\_session = Session.objects.get(session\_key=request.user.visitor.SessionKey)

old\_session.delete()

# Discard Visitor object

request.user.visitor.delete()

query = Visitor(TargetUser=request.user, SessionKey=new\_session\_key)

query.save()

# ------------------------------------------------------------------------------------

response = HttpResponse()

userType = request.user.ACT

if userType == 'Slave':

response['HX-Redirect'] = '/tm'

else:

response['HX-Redirect'] = '/'

return response

return error\_response("CHECK PASSWORD")

# During Get Method

return render(request, 'baseapp/partials/loginForm.html')

def logOut(request):

print(f'user [{request.user}] logged-out')

logout(request)

return redirect("home-page")

def registerPage(request):

return render(request, 'baseapp/partials/registerForm.html')

# New Business-Entity

def registerNewUser(request):

form = request.POST

ERROR = None

# form-data

userName, p1, p2 = form['username'].strip(), form['pass1'].strip(), form['pass2'].strip()

userEmail = form['email'].strip()

# Check Gates

if not GATE\_rgs:

ERROR = "Platform under Maintenance"

elif p1 != p2: # Form Validation

ERROR = 'Password Match Error'

elif MyCustomUserModel.objects.filter(username=userName).exists():

ERROR = 'Username not available!'

# ------------------------------------------------------------------------

def error\_response(message, success=False, org=None):

if success:

print(f'Registration success welcome orgID {org}')

e\_response = render(request, 'baseapp/snippets/platformRegisterErrors.html',

context={'errorMessage': message, 'success': success})

e\_response['HX-Retarget'] = '#register\_container'

e\_response['HX-Reswap'] = 'beforeend'

return e\_response

existing\_orgs = MyCustomUserModel.objects.values\_list('ORG', flat=True)

new\_org\_number = None

# --------------------------------------------

for i in range(3):

new\_org\_number = random.randint(500, 777)

if new\_org\_number not in existing\_orgs:

break

else:

ERROR = 'Internal Server Error!'

# ---------------------------------------------

# ------------------------------------------------------------------------

if ERROR:

return error\_response(ERROR)

user = MyCustomUserModel.objects.create\_user(username=userName,

password=p1,

email=userEmail,

ACT='Master',

ORG=new\_org\_number)

user.save()

user.refresh\_from\_db()

userProfile = Profile(User=user,

Organization=form['org\_name'].strip(),

Phone=form['phn'])

userProfile.save()

# Load storage Prevents Key Errors

storage['stats'][new\_org\_number] = {'TC': 0, 'PJ': 0, 'RJ': 0, 'SC': 0}

storage[new\_org\_number] = Paginator([], 1)

# -------------------------------------------------------------------------

# Caution Below is Non-Error message

return error\_response("Registration Success", success=True, org=new\_org\_number)

@login\_required

def homePage(request):

if request.user.ACT == "Slave":

return HttpResponse("Unauthorized access!")

db\_count = storage['stats'][request.user.ORG]['TC']

db\_process = storage['stats'][request.user.ORG]['PJ']

db\_completed = storage['stats'][request.user.ORG]['RJ']

db\_atStore = storage['stats'][request.user.ORG]['SC']

context = {'rows': storage[request.user.ORG].page(1),

'db\_count': db\_count, 'db\_atStore': db\_atStore,

'db\_today': today\_new\_entries(request),

'db\_process': db\_process, 'db\_completed': db\_completed}

return render(request, 'baseapp/index.html', context=context)

@csrf\_exempt

def sendPages(request, pg):

if storage[request.user.ORG].num\_pages < pg:

return HttpResponse('<tr><td colspan="11" style="color:red"> End of the Records </td></tr>')

return render(request, 'baseapp/partials/hmpgTableRenderHTMX.html',

context={'page': pg + 1,

'rows': storage[request.user.ORG].page(pg)})

def firstUpdatedPage(request): # Updating user page after Updated-Record

return render(request, 'baseapp/partials/pageOneHome.html',

context={'rows': storage[request.user.ORG].page(1)})

def updatePaginator(ORG\_number): # new-Paginator with new-Create Operation on DataBase

storage[ORG\_number] = Paginator(Job.objects.filter(SORG=ORG\_number).order\_by('-Updated', '-id'), 20)

def djFilterResultPage(request):

djfilter = CustomerFilter(request.POST,

queryset=Job.objects.filter(SORG=request.user.ORG).order\_by('-Updated'))

return render(request, 'baseapp/partials/djfilterResult.html', context={'rows': djfilter.qs})

def widget\_data\_request(request, category):

if category == 1:

query = Job.objects.filter(Status='pending', SORG=request.user.ORG).order\_by('Updated')

return render(request, 'baseapp/partials/widgetSimpleRender.html', context={'rows': query})

elif category == 2:

query = Job.objects.filter(Status='Ready', SORG=request.user.ORG).order\_by('-Updated')

return render(request, 'baseapp/partials/widgetSimpleRender.html', context={'rows': query})

elif category == 3:

date = datetime.strftime(datetime.now(), '%Y-%m-%d')

query = Job.objects.filter(Created\_\_icontains=date, SORG=request.user.ORG).order\_by('-Created')

return render(request, 'baseapp/partials/widgetSimpleRender.html', context={'rows': query})

return HttpResponse('Invalid request !!!')

# ------------------------------------------------------------------------------------------

@login\_required

def search\_clients\_Stage(request): # returns a page to add customers

x = Job.objects.all().order\_by('-Created')[:3] # recent 3 Clients

return render(request, 'baseapp/selectMyClient.html', context={'recent\_items': x})

@csrf\_exempt

def search\_existing\_customers(request): # returns matching registered customers records

user\_input = request.POST.get('searchBox').split()

filtered\_qs = None

if not user\_input:

pass # user has nothing in searchBox

else:

filtered\_qs = CustomerRegistration.objects.filter(CustomerName\_\_icontains=user\_input[0],

CORG=request.user.ORG) | \

CustomerRegistration.objects.filter(

Phone\_\_icontains=user\_input[1] if len(user\_input) > 1 else 'zzz', CORG=request.user.ORG) | \

CustomerRegistration.objects.filter(Phone\_\_icontains=user\_input[0], CORG=request.user.ORG)

if not filtered\_qs:

# search-box didn't bring any results and search-box is not empty

response = render(request, 'baseapp/partials/customerNotFound.html', context={'reuse': user\_input[0]})

return response

context = {'foundUsers': filtered\_qs}

return render(request, 'baseapp/partials/found-results.html', context=context)

@csrf\_exempt

def register\_customer(request, reuse=None): # new customer registering to database

if request.method == 'GET':

context = {}

if reuse and reuse.isnumeric():

context['prefilled'] = reuse

return render(request, 'baseapp/snippets/registerCustomerInputBoxes.html', context=context)

name, phone = request.POST['name'], request.POST['phone']

if len(name) < 3 or len(phone) > 10 or len(phone) < 10 or not phone.isnumeric():

return HttpResponse('<p id="error\_input">Invalid phone Number</p>')

if CustomerRegistration.objects.filter(Phone=phone, CORG=request.user.ORG).exists():

query = CustomerRegistration.objects.get(Phone=phone, CORG=request.user.ORG)

return render(request, 'baseapp/snippets/alreadyExists.html', context={'cidn': query.CIDN})

nums = get\_new\_cidn\_id()

creating\_user = CustomerRegistration(id=nums[0], CIDN=nums[1],

CustomerName=name, Phone=phone,

CORG=request.user.ORG)

creating\_user.save()

# Updating Dashboard

storage['stats'][request.user.ORG]['TC'] += 1

return render(request, 'baseapp/snippets/successRegCust.html', context={'cidn': nums[1]})

def cidn\_serviceInitiation\_stage(request, cidn, page=None): # page to add jobs to customer after choosing customer

query = CustomerRegistration.objects.get(CIDN=cidn)

context = {'user\_obj': query, 'cidn': query.CIDN}

if not page: # search result swap problem

response = HttpResponse()

response['HX-redirect'] = f'job-to-customer/{cidn}/1'

return response

return render(request, 'baseapp/jobToCustomer.html', context=context)

@csrf\_exempt

def Handle\_ServiceForm(request, fcidn=None):

if request.method == 'GET': # returns an Empty Form

context = {'cidn': fcidn}

return render(request, 'baseapp/partials/tiny\_job\_form.html', context=context)

# Let's Register Service to System

parent\_obj = CustomerRegistration.objects.get(CIDN=fcidn, CORG=request.user.ORG)

sequence, sidn = get\_new\_sidn\_id()

payload = Job(id=sequence,

SIDN=sidn,

Name=parent\_obj,

Type=request.POST['Type'],

Model=request.POST['Model'],

SN=request.POST['SN'],

Problem=request.POST['Problem'],

Price=0 if not request.POST['Price'] else request.POST['Price'],

Deposit=0 if not request.POST['Deposit'] else request.POST['Deposit'],

Phone2=None if not request.POST['Phone2'] else request.POST['Phone2'],

Payment=request.POST['Payment'],

Description=request.POST['Description'],

SORG=request.user.ORG)

parent\_obj.Orders += 1

parent\_obj.save()

payload.save()

# Updating Dashboard

storage['stats'][request.user.ORG]['PJ'] += 1

storage['stats'][request.user.ORG]['SC'] += 1

updatePaginator(request.user.ORG)

context = {'data': payload}

# response\_model('Service Requested', sidn)

return render(request, 'baseapp/partials/postSaveDetailCard.html', context=context)

@csrf\_exempt

def edit\_saved\_service\_form(request, sidn): # Immediate Service Only

query = Job.objects.get(SIDN=sidn, SORG=request.user.ORG)

if request.method == 'POST':

query.Type = request.POST['Type']

query.Model = request.POST['Model']

query.SN = request.POST['SN']

query.Problem = request.POST['Problem']

query.Price = 0 if not request.POST['Price'] else request.POST['Price']

query.Phone2 = None if not request.POST['Phone2'] else request.POST['Phone2']

query.Deposit = 0 if not request.POST['Deposit'] else request.POST['Deposit']

query.Payment = request.POST['Payment']

query.Description = request.POST['Description']

query.save()

return render(request, 'baseapp/partials/postSaveDetailCard.html', context={'data': query})

return render(request, 'baseapp/partials/edit\_tiny\_job\_form.html', context={'data': query})

def today\_new\_entries(request):

date = datetime.strftime(timezone.localtime(), '%Y-%m-%d')

x = Job.objects.filter(Created\_\_icontains=date, SORG=request.user.ORG).count()

return x

def serviceReceipt(request, sidn):

query = Job.objects.get(SIDN=sidn)

Business = request.user.profile

return render(request, "baseapp/receipt.html", context={'data': query,

'biz': Business})

def POS(request):

return render(request, 'baseapp/POS.html')

def develop\_invoice(request):

# url1 = 'https://harnishdesign.net/demo/html/koice/index-invoice-recharge.html'

# url2 = 'https://harnishdesign.net/demo/html/koice/index-invoice-movie.html'

my\_context = {'items': ['Lenovo g560', 'Micro tech-ups', 'Samsung G7 monitor', 'Hare-Krishna']}

return render(request, 'baseapp/develop\_invoice.html', context=my\_context)

def get\_new\_cidn\_id():

query = CustomerRegistration.objects.last()

if not query: # This line is for Empty Database

generated\_sequence = [1, 1000157]

return generated\_sequence

grab\_seconds = datetime.now().strftime('%S')

generated\_sequence = [query.id + 1, int(str(query.CIDN + 100)[:5] + grab\_seconds)]

return generated\_sequence

def get\_new\_sidn\_id():

query = Job.objects.last()

if not query: # This line is for Empty Database

generated\_sequence = [1, 1000157]

return generated\_sequence

grab\_seconds = datetime.now().strftime('%S')

generated\_sequence = [query.id + 1, int(str(query.SIDN + 100)[:5] + grab\_seconds)]

return generated\_sequence

def response\_model(message, sidn):

query = Response(CIDN=sidn, Status=message)

query.save()

# ------------------------------------------- Dashboard-Operations ----------------------------------------------------

@csrf\_exempt

def recordDetail(request, sidn=None, pageRefresh=False): # data to Dashboard

user\_obj = Job.objects.get(SIDN=sidn, SORG=request.user.ORG)

context = {'data': user\_obj, 'pageRefresh': pageRefresh}

return render(request, 'baseapp/dashboard.html', context=context)

def floatingDashboard(request, sidn=None, model=None):

query = Job.objects.get(SIDN=sidn, SORG=request.user.ORG)

if model == 'viewUpdateService':

return render(request, 'baseapp/partials/floatingDashboardViewUpdate.html',

context={'sidn': sidn, 'data': query})

elif model == 'moreActions':

return render(request, 'baseapp/partials/floatingMoreActionsPerRecord.html',

context={'sidn': sidn, 'cidn': query.Name.CIDN})

else:

return HttpResponse('Invalid response!')

@csrf\_exempt

def markStatus(request, action=None, sidn=None):

instance = Job.objects.get(SIDN=sidn, SORG=request.user.ORG)

credential = request.user.ORG

if action == 'ready':

instance.Status = 'Ready'

instance.RequireUpdate = False

instance.save()

response\_model('Ready for Pick up', sidn)

# Updating Dashboard

storage['stats'][credential]['PJ'] -= 1

storage['stats'][credential]['RJ'] += 1

return render(request, 'baseapp/snippets/postReadyNowDeliverable.html', context={'data': instance})

elif action == 'failed':

instance.Status = 'Failed'

instance.RequireUpdate = False

instance.save()

response\_model('Service Unsuccessful', sidn)

# Updating Dashboard

storage['stats'][credential]['PJ'] -= 1

return render(request, 'baseapp/snippets/postFailedNowReturnable.html', context={'data': instance})

elif action == 'delivered':

instance.Status = 'Delivered'

instance.Payment = 'Paid'

instance.RequireUpdate = False

instance.save()

response\_model('Service Concluded', sidn)

# Updating Dashboard

storage['stats'][credential]['RJ'] -= 1

storage['stats'][credential]['SC'] -= 1

return render(request, 'baseapp/snippets/stateComplete.html', context={'data': instance})

elif action == 'returned':

instance.Status = 'Returned'

instance.RequireUpdate = False

instance.save()

response\_model('Return Completed', sidn)

# Updating Dashboard

storage['stats'][credential]['SC'] -= 1

return render(request, 'baseapp/snippets/stateReturned.html', context={'data': instance})

else:

return HttpResponse('Invalid Response!!!')

@csrf\_exempt

def edit\_existing\_service(request, sidn=None):

query = Job.objects.get(SIDN=sidn, SORG=request.user.ORG)

if request.POST['price']:

query.Price = request.POST['price']

if request.POST['deposit']:

query.Deposit = request.POST['deposit']

if request.POST['phone2']:

query.Phone2 = request.POST['phone2']

query.Description = request.POST['description']

query.save()

new\_query = Job.objects.get(SIDN=sidn, SORG=request.user.ORG)

return render(request, 'baseapp/dashboard.html', context={'data': new\_query, 'pageRefresh': True})

@csrf\_exempt

def billPanel(request, bsidn, action):

query1 = Bill.objects.filter(BORG=request.user.ORG, BSIDN=bsidn)

query2 = Job.objects.get(SORG=request.user.ORG, SIDN=bsidn)

if action == 'viewBills':

bills\_sum = query1.aggregate(Sum('Amount'))['Amount\_\_sum']

if not bills\_sum:

bills\_sum = 0

context = {'bills': query1, 'service': query2,

'Total': bills\_sum}

return render(request, 'baseapp/partials/viewBills.html', context=context)

elif action == 'getSum':

new\_sum = query1.aggregate(Sum('Amount'))['Amount\_\_sum']

return render(request, 'baseapp/snippets/updatingSum.html',

context={'new\_sum': new\_sum, 'BSIDN': bsidn})

elif action == 'getDashSum':

if not query1:

print('No bills for', bsidn)

return HttpResponse('<div id="dashSum"></div>')

new\_sum = query1.aggregate(Sum('Amount'))['Amount\_\_sum']

return render(request, 'baseapp/snippets/dashTotalAmount.html',

context={'new\_sum': new\_sum, 'sidn': bsidn})

elif action == 'refreshDashSum':

new\_sum = query1.aggregate(Sum('Amount'))['Amount\_\_sum']

if not new\_sum:

new\_sum = 0

return render(request, 'baseapp/snippets/dashTotalAmount.html',

context={'new\_sum': new\_sum, 'sidn': bsidn})

else:

print("Invalid Request")

return HttpResponse("Invalid Request")

def manageInvoice(request):

pass

@csrf\_exempt

def setEngLevel(request, cidn):

query = CustomerRegistration.objects.get(CIDN=cidn)

if request.method == 'GET':

return render(request, 'baseapp/snippets/engLevelOptions.html',

context={'data': query})

elif request.method == 'POST':

new\_level = request.POST['englvl']

if new\_level == 'setCaution':

query.EngagementTier = 'Caution'

elif new\_level == 'setElite':

query.EngagementTier = 'Elite'

elif new\_level == 'setRegular':

query.EngagementTier = 'Regular'

query.save()

return HttpResponse(f'<div class="notch" id="notch{new\_level[3:]}"> </div>')

@csrf\_exempt

def resetService(request, sidn=None):

if request.method == 'POST':

query = Job.objects.get(SIDN=sidn)

query.Status = 'pending'

query.save()

return render(request, 'baseapp/dashboard.html',

context={'data': query, 'pageRefresh': True})

return render(request, 'baseapp/snippets/pendingOption.html',

context={'sidn': sidn})

@csrf\_exempt

def Remove\_service(request, sidn): # Deletion of existing Service/Job

query = Job.objects.get(SIDN=sidn, SORG=request.user.ORG)

parentObj = CustomerRegistration.objects.get(CIDN=query.Name.CIDN, CORG=request.user.ORG)

delta = timezone.localtime() - query.Created

if delta.seconds < 330: # 5.5 minutes

parentObj.Orders -= 1

parentObj.save()

query.delete()

# Updating Dashboard

storage['stats'][request.user.ORG]['PJ'] -= 1

storage['stats'][request.user.ORG]['SC'] -= 1

age = delta.seconds // 60

response = HttpResponse()

response['HX-Redirect'] = '/'

return response

# ------------------------ Billing operations ----------- >

def billOperations(request, action, bsidn, recordID=None):

if action == 'addBill':

post = request.POST

payload = Bill(BSIDN=bsidn, BORG=request.user.ORG,

Description=post['description'] if post['description'] else 'empty',

Details=post['details'],

Quantity=int(post['quantity']),

Amount=int(post['amount'])\*int(post['quantity']))

payload.save()

return render(request, 'baseapp/snippets/newStoredBill.html',

context={'bill': payload, 'SIDN': bsidn})

elif action == 'appendInputBoxes':

return render(request, 'baseapp/snippets/newBillInputs.html', context={'SIDN': bsidn})

elif action == 'editBill':

query = Bill.objects.get(BORG=request.user.ORG, id=recordID)

return render(request, 'baseapp/snippets/editStoredBill.html',

context={'bill': query})

elif action == 'saveEditedBill':

post = request.POST

print(post, recordID)

query = Bill.objects.get(BORG=request.user.ORG, id=recordID)

query.Description = post['description']

query.Details = post['details']

query.Quantity = post['quantity']

query.Amount = int(post['quantity']) \* int(post['amount'])

query.save()

return render(request, 'baseapp/snippets/storedBillUpdated.html', context={'bill': query})

elif action == 'deleteBill':

query = Bill.objects.get(BORG=request.user.ORG, id=recordID, BSIDN=bsidn)

query.delete()

new\_query = Bill.objects.filter(BORG=request.user.ORG, BSIDN=bsidn)

sum\_post\_delete = new\_query.aggregate(Sum('Amount'))['Amount\_\_sum']

if not sum\_post\_delete:

sum\_post\_delete = 0

return render(request, 'baseapp/snippets/billAmountUpdateNdelete.html',

context={'amount': sum\_post\_delete, 'recordID': recordID, 'SIDN': bsidn})

else:

print('reporting error from line 645')

return HttpResponse('Invalid request!')

# ------------------------ EMAIL -------------------------->

def dispatchEmail(request):

# observed 3 seconds delay

# calling Celery Task

emailDispatchtask.delay(['mohanakrishna329@gmail.com'])

return HttpResponse('Email')

# ------------------------ & & ------------------------------->

# ------------------------- Miscellaneous -------- >

def Dash\_Info\_Reset(credential):

storage['stats'][credential] = {

'TC': CustomerRegistration.objects.filter(CORG=credential).count(),

'PJ': Job.objects.filter(Status='pending', SORG=credential).count(),

'RJ': Job.objects.filter(Status='Ready', SORG=credential).count(),

'SC': Job.objects.filter(SORG=credential).exclude(Status\_\_in=['Delivered', 'Returned']).count()

}

def control\_Gates(request, gate\_id):

global GATE\_rgs

if gate\_id == 1:

GATE\_rgs = not GATE\_rgs

message = f' GATE rgs set to {GATE\_rgs}'

else:

message = f' GATE rgs live Status ({GATE\_rgs})'

print(message)

return HttpResponse(message)

def AjaxRedirect(request, action=None): # Mixed Redirects for HTMX

response = HttpResponse()

if action == 'home':

if request.user.ACT == 'Slave':

response['HX-Redirect'] = '/tm'

else:

response['HX-Redirect'] = '/'

elif action == 'kickuser':

logout(request)

response['HX-Redirect'] = '/'

return response

def empty\_response(request, origin):

if origin == 'vuFloatBoard':

# Hidden DIV prevents Duplicates HTMX swaps

return HttpResponse('<div id="receiveFloatBoard"></div>')

elif origin == 'akaSTATERESET':

# Hidden DIV prevents Duplicates HTMX swaps

return HttpResponse('<div id="receiveFloatBoard"></div>')

return HttpResponse()

def spinPaginatorStorage():

# get\_all\_ORG = MyCustomUserModel.objects.filter(ACT="Master") # old One

get\_all\_ORG = MyCustomUserModel.objects.values\_list("ORG", flat=True).distinct()

for item in get\_all\_ORG:

storage[item] = Paginator(Job.objects.filter(SORG=item).order\_by('-Updated', '-id'), 20)

def spinStatsStorage():

get\_all\_ORG = MyCustomUserModel.objects.values\_list("ORG", flat=True).distinct()

for item in get\_all\_ORG:

storage['stats'][item] = {

'TC': CustomerRegistration.objects.filter(CORG=item).count(),

'PJ': Job.objects.filter(Status='pending', SORG=item).count(),

'RJ': Job.objects.filter(Status='Ready', SORG=item).count(),

'SC': Job.objects.filter(SORG=item).exclude(Status\_\_in=['Delivered', 'Returned']).count()

}

# ------------------------------------------- Debug & Testing ---------------------------------------------------------

def run\_Script(request):

# Load scripts Here

client = boto3.client('sns',

aws\_access\_key\_id='AKIAV3V2LVTKGBY2KZPX',

aws\_secret\_access\_key='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',

region\_name="ap-south-1")

client.publish(PhoneNumber='+919676317703', Message='Bollo World')

return HttpResponse('Script Ran Successfully !!!')

def TroubleShoot(request):

# Empty Job

# add.delay(10)

return HttpResponse("Troubleshooter empty!")

@csrf\_exempt

def testPage(request):

x = timezone.now()

y = timezone.now() + relativedelta(days=-1)

z = y-x

print('x:', x.strftime(r'%a %d %B %Y, %I:%M %p'))

print('y:', y.strftime(r'%a %d %B %Y, %I:%M %p'))

print('z:', z.days)

return render(request, 'baseapp/test.html', context={})

def testPage\_2(request):

return render(request, 'baseapp/test2.html', context={})

# PreLoaders

spinPaginatorStorage()

spinStatsStorage()

**App2.py:**

from django.shortcuts import render, redirect

from django.http import HttpResponse

from django.views.decorators.csrf import csrf\_exempt

from geopy.distance import geodesic

from connect.models import Lead, QuoteRequest, Quote, Feedback

from baseapp.models import Profile

import random

# Create your views here.

storage = {

'bussiness\_locations': Profile.objects.only('Latitude', 'Longitude')

}

def checkSession(func):

def wrapper(request, \*args, \*\*kwargs):

# ----- Create Session -----

if not request.session.session\_key:

# creates session

request.session.set\_expiry(2400)

'''

session expires due to inactivity.

Modifying Session is considered as activity not reading,

applies when Integer as argument to set\_expiry

'''

request.session['Authenticated'] = False

request.session['userCredential'] = None

request.session['userPhone'] = None

request.session['OTP'] = None

request.session['current\_location'] = None

print('session Created')

elif not request.session.get('Authenticated'):

request.session['Authenticated'] = False

request.session['current\_location'] = None

return func(request, \*args, \*\*kwargs)

return wrapper

@checkSession

def main(request):

session = request.session

context = {'Authenticated': session.get('Authenticated', False),

'displayName': session.get('name', ''),

'displayPhone': session.get('userPhone', '')}

if session.get('pushNotification', False):

context['pushNotification'] = session['pushNotification']

session['pushNotification'] = None

# ----------------------------------------------------------

if session['Authenticated']:

context['rows'] = QuoteRequest.objects.filter(Owner=session['userCredential']).order\_by('-Created')

context['requestCount'] = QuoteRequest.objects.filter(Owner=session['userCredential']).count()

# ----------------------------------------------------------

return render(request, 'connect/index.html', context=context)

@checkSession

def loginLead(request, action=None):

remote\_address = request.META['REMOTE\_ADDR']

sessionKey = request.session.session\_key

session = request.session

if request.method == 'POST':

form = request.POST

if action == 'checkUser':

#-- Input Validation ------------------

error = None

if not form['phoneEmail'].isnumeric():

error = 'Invalid input!'

elif len(form['phoneEmail']) < 10 or len(form['phoneEmail']) > 10:

error = 'Invalid phone number'

if error:

response = HttpResponse(error)

response['HX-retarget'] = "#daBox"

return response

#--------------------------------------

if Lead.objects.filter(Phone=form['phoneEmail']).exists():

# User account Match!

# Get OTP ready

session['userCredential'] = form['phoneEmail']

session['name'] = Lead.objects.get(Phone=form['phoneEmail']).Name

session['OTP'] = '1234'

return render(request, 'connect/snippets/nextLoginInputs.html',

context={'existingUser': True, 'givenInput': form['phoneEmail']})

else:

# user not Found

# Get OTP ready

session['userCredential'] = form['phoneEmail']

session['OTP'] = '8595'

return render(request, 'connect/snippets/nextLoginInputs.html',

context={'existingUser': False, 'givenInput': form['phoneEmail']})

elif action == 'confirmOTP':

if session['OTP'] and session['OTP'] == form['OTP']:

session['Authenticated'] = True

# Greet existing user

session['pushNotification'] = f"welcome back {session['name']}"

response = HttpResponse()

response['HX-redirect'] = "/connect"

return response

else:

response = HttpResponse('Incorrect OTP')

response['HX-retarget'] = "#daBox"

return response

elif action == 'createAccount':

x, y, z = session['userCredential'], form['newname'], form['OTP']

# if OTP valid create account

if session['OTP'] == z:

query = Lead(Phone=x, Name=y)

query.save()

session['Authenticated'] = True

session['name'] = form['newname']

# Welcoming New user

session['pushNotification'] = f"Welcome {y}"

response = HttpResponse()

response['HX-redirect'] = "/connect"

return response

elif action == 'logout':

request.session.clear()

response = HttpResponse()

response['HX-Redirect'] = '/connect'

return response

return render(request, 'connect/partials/loginSignUp.html')

# ALL Request & Quote Operations

@checkSession

def quoteRequestForm(request):

session = request.session

# -- Authentication ------------------------>

if not request.session['Authenticated']:

session['pushNotification'] = "Account login Required!"

response = HttpResponse()

response['HX-Redirect'] = '/connect'

return response

# ------------------------------------------>

if request.method == 'POST':

form = request.POST

x, y, z = form['model'], form['problem'], form['preferences']

parentObj = Lead.objects.get(Phone=session['userCredential'])

new\_query = QuoteRequest(Owner=parentObj,

Model=x, Problem=y, Preferences=z)

new\_query.save()

return render(request, 'connect/snippets/requestFormSuccess.html')

return render(request, 'connect/partials/requestForm.html')

def quoteRequestOps(request, action, qrid):

query = QuoteRequest.objects.get(id=qrid)

if action == 'showQuotes':

found\_quotes = query.qquotes.all()

# calculating distances

for item in found\_quotes:

user\_location = request.session['current\_location']

if not user\_location:break

service\_provider = item.Organization.Latitude, item.Organization.Longitude

item.ddistance = round(geodesic(user\_location, (service\_provider[0], service\_provider[1])).km, 2)

return render(request, 'connect/partials/acquiredQuotes.html',

context={'quotes': found\_quotes})

elif action == 'showOptions':

return render(request, 'connect/snippets/quoteRequestOptions.html',

context={'qrid': qrid})

return HttpResponse()

# && existing request Operations

def quoteOps(request, action, qid):

if action == 'showOptions':

query = Quote.objects.get(id=qid)

return render(request, 'connect/snippets/quoteOptions.html',

context={'quote': query})

elif action == 'lockQuote':

query = Quote.objects.get(id=qid)

query.Locked = not query.Locked

query.save()

return render(request, 'connect/snippets/refreshQuote.html',

context={'quote': query})

def userLocation(request):

# Implement User location name with below Link, May not be accurate

# https://www.geeksforgeeks.org/get-the-city-state-and-country-names-from-latitude-and-longitude-using-python/

req = request.GET

# print('\n Client-Browser \n Latitude:', req['latitude'], 'Longitude:', req['longitude'], '\n')

user\_location = 17.47063, 78.40593

calculated\_results = {}

# ------------------------------------------------------------

for provider in storage['bussiness\_locations']:

if not provider.Latitude:continue

distance = geodesic(user\_location, (provider.Latitude, provider.Longitude)).km

calculated\_results[distance] = provider.Organization

# ------------------------------------------------------------

# -----------------------------------------------------------

sorted\_distances = list(calculated\_results.keys())

sorted\_distances.sort()

result = {keey: calculated\_results[keey] for keey in sorted\_distances}

# print('\n calculated distances ASC', result, '\n')

# -----------------------------------------------------------

if int(float(req['latitude'])):

request.session['current\_location'] = req['latitude'], req['longitude']

else:

response = render(request, 'connect/partials/newPushNotification.html',

context={'notificationMessage': 'location unavailable!',

'color': 'red', 'bgcolor': 'white'})

response['HX-Reswap'] = 'outerHTML'

return response

return HttpResponse()

def storeFeedback(request):

if request.method == 'POST':

form = request.POST

if form['message']:

query = Feedback(Message=form['message'])

query.save()

return render(request, 'connect/snippets/feedbackSuccess.html')

print('Feedback empty!')

return render(request, 'connect/snippets/feedbackSuccess.html')

def clrSessions(request):

# clears all expired sessions regardless of current-user-browser

request.session.clear\_expired()

return HttpResponse("")

def experiments(request):

return HttpResponse()

**App3.py**

from django.shortcuts import render

from django.contrib.auth.decorators import login\_required

from django.views.decorators.csrf import csrf\_exempt

from django.http import HttpResponse

from django.db.models import Sum

from django.utils import timezone

from dateutil.relativedelta import relativedelta

from baseapp.models import (Job, CustomerRegistration, MyCustomUserModel, Profile,

Bill, ServiceWarranties)

# Create your views here.

console\_storage = {

'stats': {

}

}

@login\_required

def console(request):

# Authorization

if request.user.ACT != "Master":

return HttpResponse("Unauthorized access!")

# ----------------------- Update Stats

x = console\_storage['stats'][request.user.ORG]['updated']

y = timezone.localtime()

if (y - x).seconds > 60: # seconds older

updateOrgStats(request.user.ORG)

# -----------------------

context = {'dash': console\_storage['stats'][request.user.ORG]}

return render(request, 'console/Base\_console.html', context=context)

@login\_required

def consoleSections(request, section=None): # invokes matching Console Sectional-Pages

if section == 'customers':

return CustomerDB(request)

elif section == 'employees':

return manageEmployees(request)

elif section == 'accountSettings':

return accountSettings(request)

elif section == 'billsDB':

return BillsDB(request)

elif section == 'warranties':

return serviceWarranties(request)

else:

return HttpResponse('<div style="color:red; text-align:center;"> Resource Unavailable!!! </div>')

# ---------------Return-Sectional-Rendering---------------- #

def CustomerDB(request):

query = CustomerRegistration.objects.filter(CORG=request.user.ORG)

return render(request, 'console/console\_customers.html',

context={'records': query})

def manageEmployees(request):

query = MyCustomUserModel.objects.filter(ACT\_\_in=['Slave', 'Shared'], ORG=request.user.ORG)

return render(request, 'console/console\_employees.html',

context={'records': query})

def accountSettings(request):

query1 = request.user

query2 = request. user.profile

return render(request, 'console/console\_account.html',

context={'dataU': query1,

'dataP': query2})

def BillsDB(request):

query = Bill.objects.filter(BORG=request.user.ORG).order\_by('-Created')

return render(request, 'console/console\_bills.html',

context={'rows': query})

def serviceWarranties(request):

query = ServiceWarranties.objects.filter(WORG=request.user.ORG)

# add warranty Active status attribute

for item in query:

x = item.Created

y = item.EndDate

z = y - x

if int(z.days) > 0:

item.Active = 'Active'

else:

item.Active = 'Expired'

print('remaining days:', z.days, type(z.days))

return render(request, 'console/console\_warranties.html',

context={'rows': query})

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Mixed-Short-Processing \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ #

# -----------------------------------------------Customer-DB------------------------------------------------------------

def viewClientInfo(request, cidn=None): # Single Client record info sent

query = CustomerRegistration.objects.get(CIDN=cidn, CORG=request.user.ORG)

return render(request, 'console/partials/viewSingleClient.html', context={'data': query})

def editClientInfo(request, cidn=None):

query = CustomerRegistration.objects.get(CIDN=cidn, CORG=request.user.ORG)

if request.method == 'POST': # saving Data

extract = request.POST

query.CustomerName = extract['ClientName']

query.Phone = extract['Phone']

query.EngagementTier = extract['Eng\_lvl']

query.save()

return render(request, 'console/partials/editStatusPostSubmit.html')

return render(request, 'console/partials/editSingleClient.html', context={'data': query})

def deleteClient(request, cidn=None):

query = CustomerRegistration.objects.get(CIDN=cidn)

query.delete()

return HttpResponse(f'<p>customer <strong>{query.CustomerName}</strong> and records Deleted Successfully !!! </p>')

# ----------------------------------------------- Manage Employees -----------------------------------------------------

@csrf\_exempt

def viewEmployeeInfo(request, pk):

query = MyCustomUserModel.objects.get(id=pk)

return render(request, 'console/partials/viewSingleEmployee.html', context={'data': query})

@csrf\_exempt

def deactivateEmployee(request, pk):

query = MyCustomUserModel.objects.get(id=pk)

query.is\_active = not query.is\_active

query.save()

return manageEmployees(request)

@csrf\_exempt

def resetEmployeePassWord(request, pk):

if request.method == 'GET':

return render(request, 'console/snippets/resetPasswordInputBox.html', context={'pk': pk})

if request.method == 'POST':

new\_password = request.POST['newPassword'].strip()

query = MyCustomUserModel.objects.get(id=pk)

query.set\_password(new\_password)

query.save()

return manageEmployees(request)

@csrf\_exempt

def removeEmployeeAccount(request, pk):

query = MyCustomUserModel.objects.get(id=pk)

query.delete()

return manageEmployees(request)

@csrf\_exempt

def addEmployee(request):

if request.method == 'GET':

return render(request, 'console/snippets/newEmployeeInputBox.html')

userName, userPassword = request.POST.get('userName').strip(), request.POST.get('userPassword').strip()

if userName and userPassword and not MyCustomUserModel.objects.filter(username=userName).exists():

new\_user = MyCustomUserModel.objects.create\_user(username=userName, password=userPassword,

ACT='Slave', ORG=request.user.ORG)

new\_user.save()

new\_user.refresh\_from\_db()

# create low Profile

low\_profile = Profile(User=new\_user,

Organization=request.user.profile.Organization,

Phone=request.user.profile.Phone,

Landline=request.user.profile.Landline,

Location=request.user.profile.Location)

low\_profile.save()

else:

return HttpResponse("Error Occurred!")

return manageEmployees(request)

# ------------------------------------------------ WarranTies --------------------------------------------------------#

def warrantyOps(request, action, wid):

def helperFun(choice, number):

if choice == 'Days':

return relativedelta(days=number)

elif choice == 'Months':

return relativedelta(months=number)

elif choice == 'Years':

return relativedelta(years=number)

if action == 'viewDetails':

query = ServiceWarranties.objects.get(id=wid)

return render(request, 'console/partials/warranties/viewWarrantyDetails.html',

context={'data': query})

elif action == 'editWarranty':

query = ServiceWarranties.objects.get(id=wid)

return render(request, 'console/partials/warranties/editWarranty.html',

context={'data': query})

elif action == 'saveEditedWarranty':

query = ServiceWarranties.objects.get(id=wid)

req = request.POST

query.Includes = req['includes']

query.Excludes = req['excludes']

query.Duration = req['duration']

query.DurationType = req['durationType']

query.EndDate = query.Created + helperFun(req['durationType'], int(req['duration']))

query.save()

return HttpResponse('Edit Success')

elif action == 'deleteWarranty':

query = ServiceWarranties.objects.get(id=wid)

query.delete()

return HttpResponse('Deleted Sucessfully!')

elif action == 'newForm':

return render(request, 'console/partials/warranties/newWarrantyForm.html')

elif action == 'saveForm':

req = request.POST

# check SIDN

if not Job.objects.filter(SORG=request.user.ORG, SIDN=req['SIDN']).exists():

return HttpResponse("Service ID not Found")

future\_date = timezone.now() + helperFun(req['durationType'], int(req['duration']))

query1 = Job.objects.get(SORG=request.user.ORG, SIDN=req['SIDN'])

query2 = ServiceWarranties(WORG=request.user.ORG, ServiceFor=query1,

Includes=req['includes'], Excludes=req['excludes'],

Duration=req['duration'], DurationType=req['durationType'],

EndDate=future\_date)

query2.save()

return HttpResponse('Sucess')

return HttpResponse('Invalid')

# ------------------------------------------------ Miscellaneous ------------------------------------------------------

def updateOrgStats(org):

console\_storage['stats'][org] = {

'TotalCustomers': CustomerRegistration.objects.filter(CORG=org).count(),

'BilledAmount': Bill.objects.filter(BORG=org).aggregate(Sum("Amount"))['Amount\_\_sum'],

'updated': timezone.localtime()

}

def spinConsoleStorage():

query = MyCustomUserModel.objects.values\_list("ORG", flat=True).distinct()

for item in query:

console\_storage['stats'][item] = {

'TotalCustomers': CustomerRegistration.objects.filter(CORG=item).count(),

'BilledAmount': Bill.objects.filter(BORG=item).aggregate(Sum("Amount"))['Amount\_\_sum'],

'updated': timezone.localtime()}

# preLoaders

spinConsoleStorage()

**Dockerfile:**

FROM python:3.10-slim-buster

ENV PYTHONUNBUFFERED=1

WORKDIR /django

COPY requirements.txt requirements.txt

RUN pip3 install -r requirements.txt

docker-compose.yml

version: '3.8'

services:

web:

build: .

volumes:

- .:/django

ports:

- 8000:8000

container\_name: django\_development\_container

command: python manage.py runserver 0.0.0.0:8000

redis:

image: redis:alpine

container\_name: redis\_development\_container

ports:

- 6379:6379

celery:

restart: always

build:

context: .

command: celery -A Servicejobs worker -l INFO

volumes:

- .:/django

container\_name: celery\_development\_container

depends\_on:

- redis

# flower:

# image: mher/flower

# restart: always

# ports:

# - 5555:5555

# environment:

# - CELERY\_BROKER\_URL=redis://redis\_development\_container:6379/

**CHAPTER 6: TESTING**

Testing is an essential phase in the development of our CRM aimed at ensuring its functionality, accuracy, and security. The testing process involves a comprehensive evaluation of the system's components and workflows to verify that they meet specified requirements and perform as expected. Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property function as a unit. The test data should be chosen such that it passes through all possible conditions.Testing is the verification of the CRM software's functionality, reliability, and performance to ensure that it meets the business requirements and user needs. It includes testing the usability, data accuracy, compatibility issues, documentation, and reporting of the CRM system.

**6.1 UNIT TESTING**

For the CRM system, unit testing involves testing individual components or modules to ensure they function correctly in isolation. This includes testing functions, methods, or classes within the system, and verifying that they produce the expected outputs for various inputs. Unit tests for the CRM system could involve testing functionalities such as user authentication, quote generation, and status tracking. This ensures that each component performs as intended and helps identify any bugs or errors early in the development process.

**6.2 INTEGRATION TESTING**

Integration testing is the testing of the CRM system's integration with other third-party applications or systems, such as marketing or sales tools, to ensure a seamless flow of information and data between the CRM and other tools or platforms. It includes testing data accuracy and quality, compatibility issues, documentation, and reporting. Integration testing focuses on testing the interactions between different components or modules within the CRM system. It verifies that these components work together seamlessly to achieve the desired functionality. Integration tests for the CRM system could involve testing interactions between the quote service module, user authentication module, and status tracking module. This ensures that data flows correctly between components and that any dependencies are handled appropriately.

**6.3 SYSTEM TESTING**

System testing evaluates the entire CRM system as a whole to ensure it functions correctly in its intended environment. This testing phase assesses the system's behavior and performance under various conditions, including normal usage scenarios and edge cases. System tests for the CRM system could involve testing end-to-end workflows, such as the process of requesting a quote, tracking the status of a repair, and receiving notifications. This ensures that the system behaves as expected and meets overall requirements. System testing is the testing of the entire CRM system as a whole to ensure that it meets the specified requirements and functions correctly. It includes testing the system's performance, security, and scalability.

**6.4 MODULE TESTING**

Module testing involves testing individual functional units or modules within the CRM system to verify their correctness and reliability. This testing phase focuses on validating the behavior of specific components and ensuring they perform as intended. Module tests for the CRM system could involve testing functionalities such as user registration, quote generation algorithms, and notification systems. This helps identify and address any issues within individual modules before they impact the overall system. Module testing is the testing of a complete module or subsystem of the CRM system to ensure that it functions correctly and meets the specified requirements. It includes testing the integration of software, especially the APIs, to ensure that they work as expected.

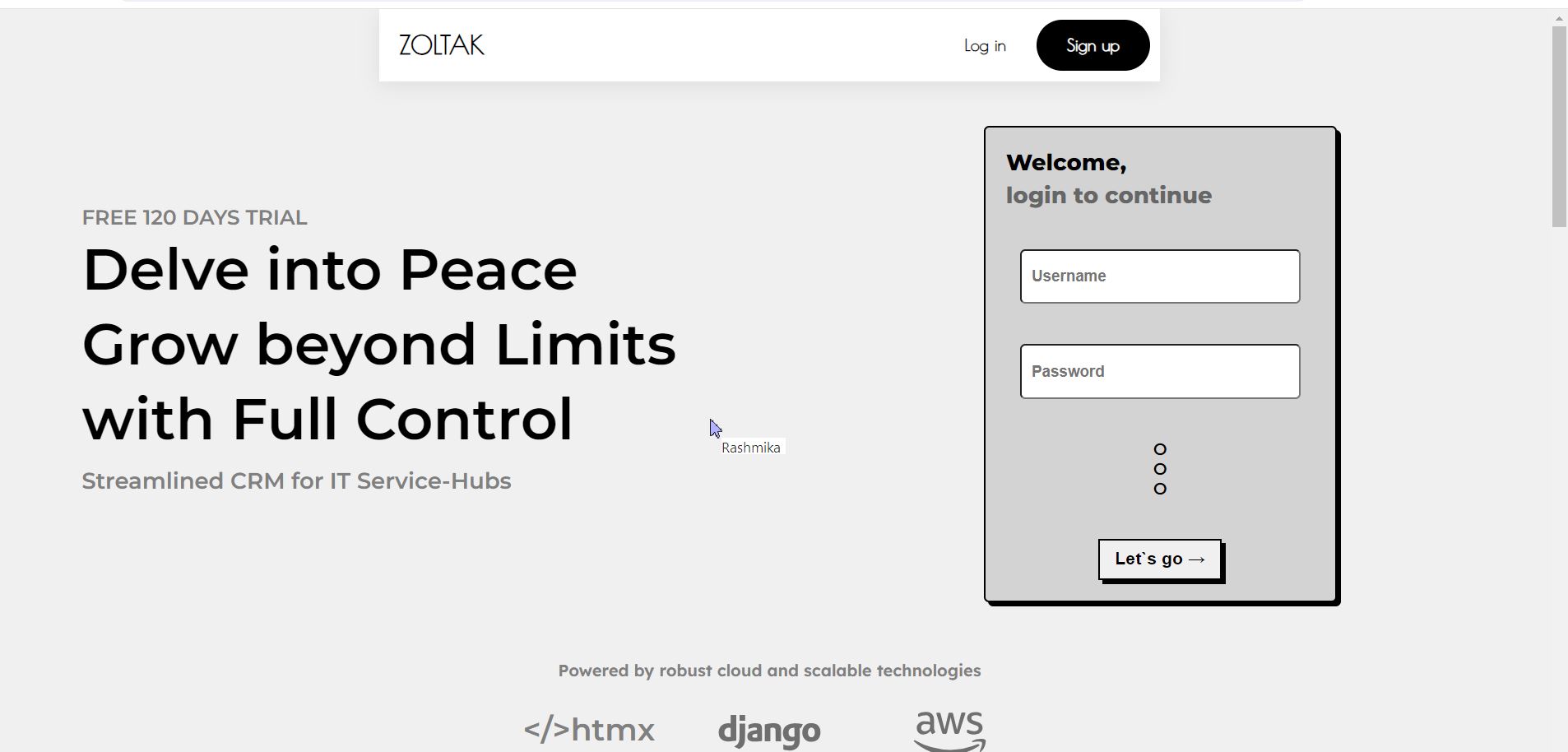
**6.5 ACCEPTANCE TESTING**

Acceptance testing involves verifying that the CRM system meets the requirements and expectations of stakeholders. This testing phase evaluates the system's functionality, usability, and performance against predefined criteria. In the context of the CRM system, acceptance testing could involve testing features such as quote service usability, responsiveness, and accuracy. Stakeholders would interact with the system to ensure it meets their needs and provides a satisfactory user experience. Acceptance testing is the testing of the CRM system by the end-users or stakeholders to ensure that it meets their requirements and is ready for deployment. It includes testing the system's usability, functionality, and performance.

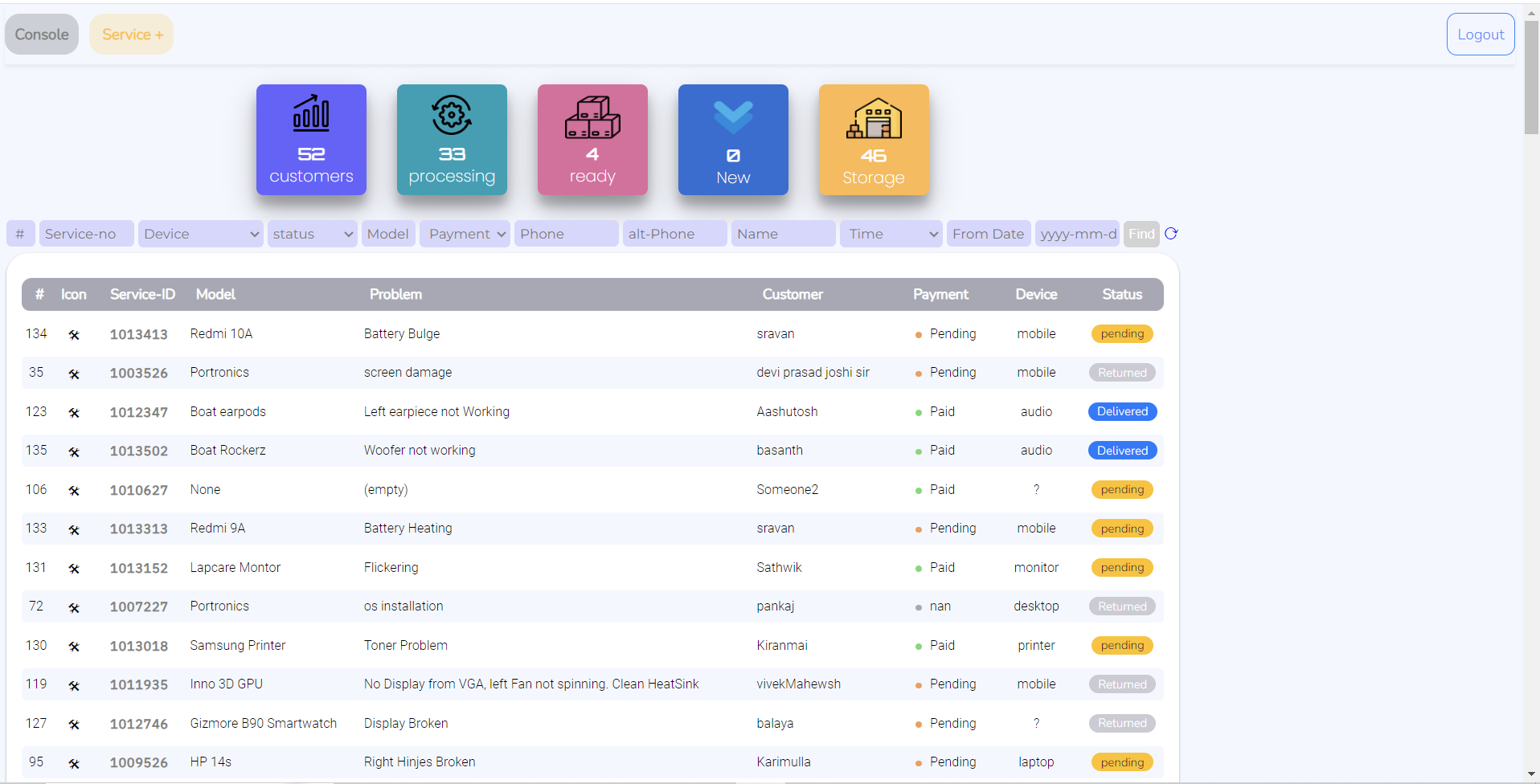
**6.6 PERFORMANCE TESTING**

Performance testing assesses the CRM system's responsiveness, scalability, and stability under various load conditions. This testing phase evaluates how the system performs under normal and peak usage scenarios to ensure it can handle the expected workload. Performance tests for the CRM system could involve simulating multiple users accessing the system simultaneously, generating a large number of quote requests, and tracking the system's response times. This helps identify any bottlenecks or performance issues that may affect user experience and allows for optimization before deployment. Performance testing is the testing of the CRM system's performance under various loads and stresses to ensure that it performs optimally and meets the specified performance requirements. It includes testing the system's response time, throughput, and scalability.

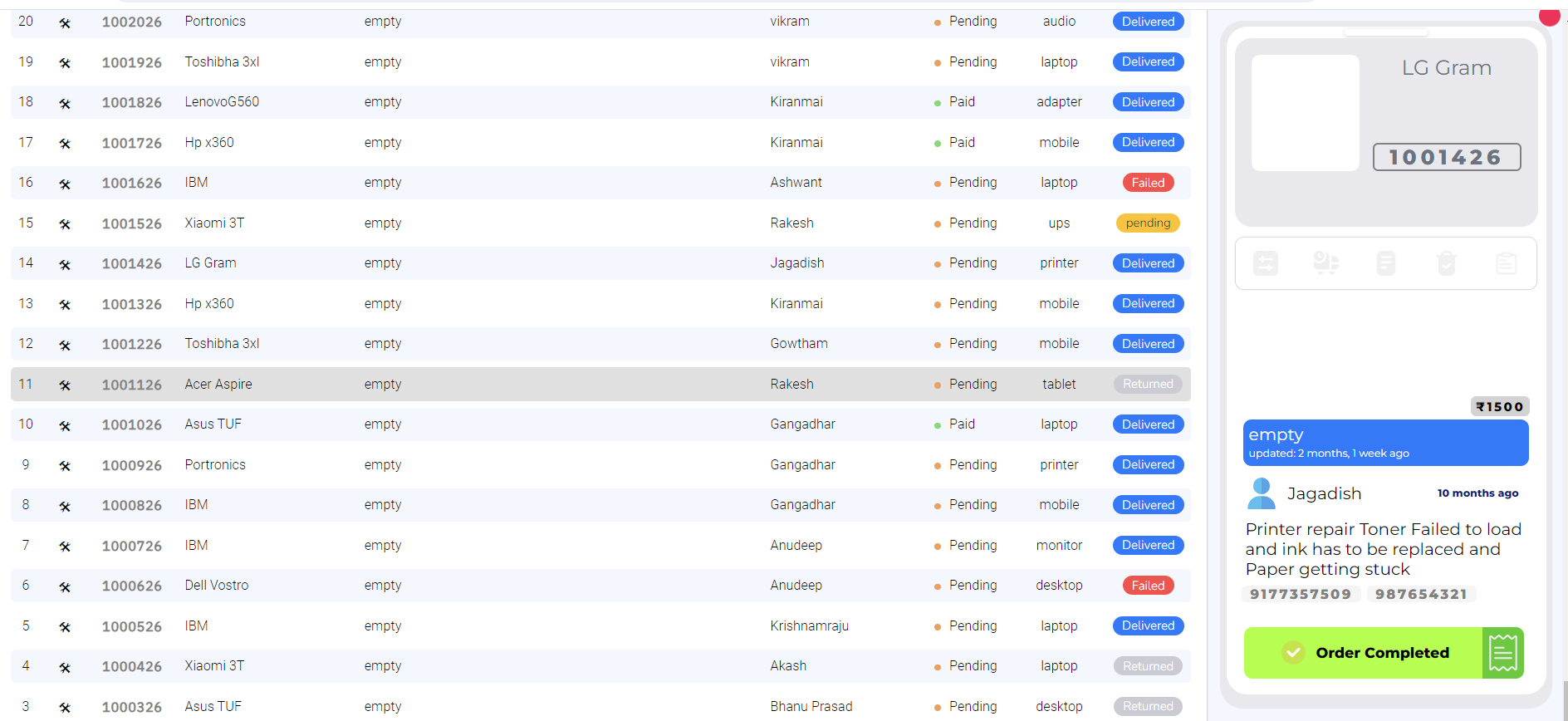
**CHAPTER 7: RESULTS**

****

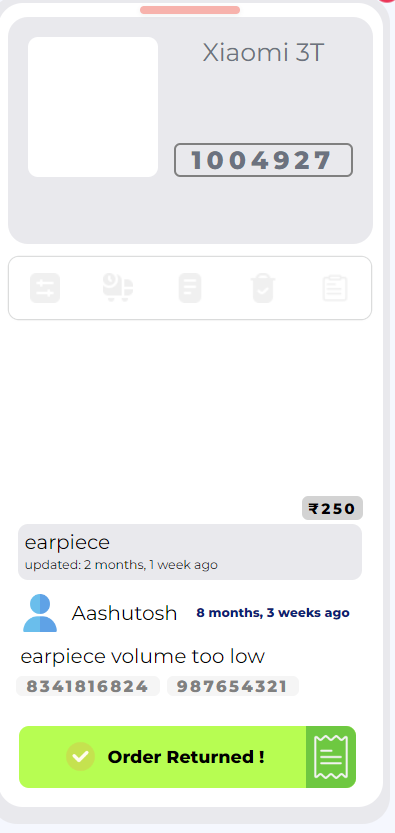
**Fig 7.1:** Login page

****

**Fig 7.2:**

****

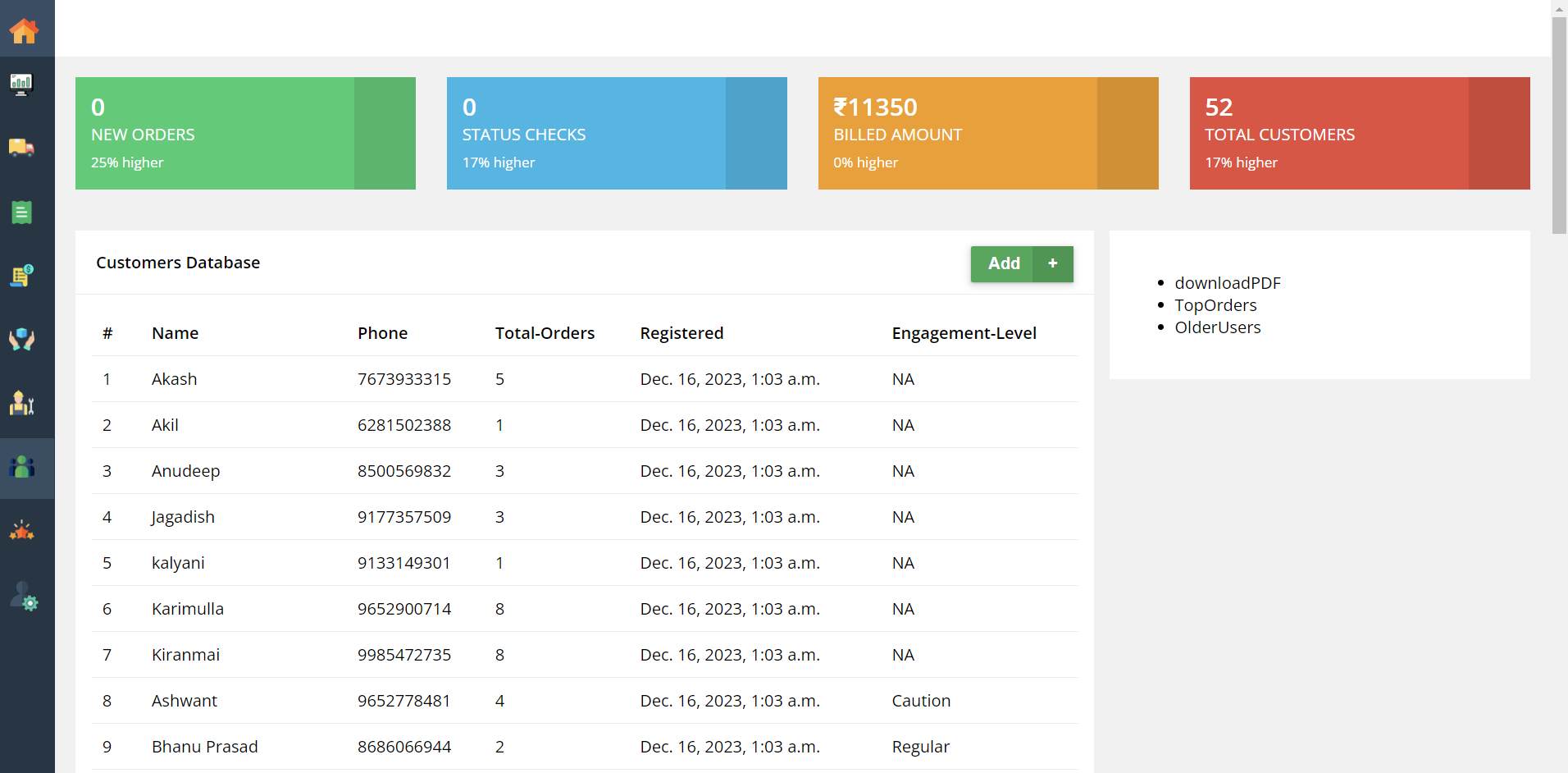
**Fig 7.3:**

****

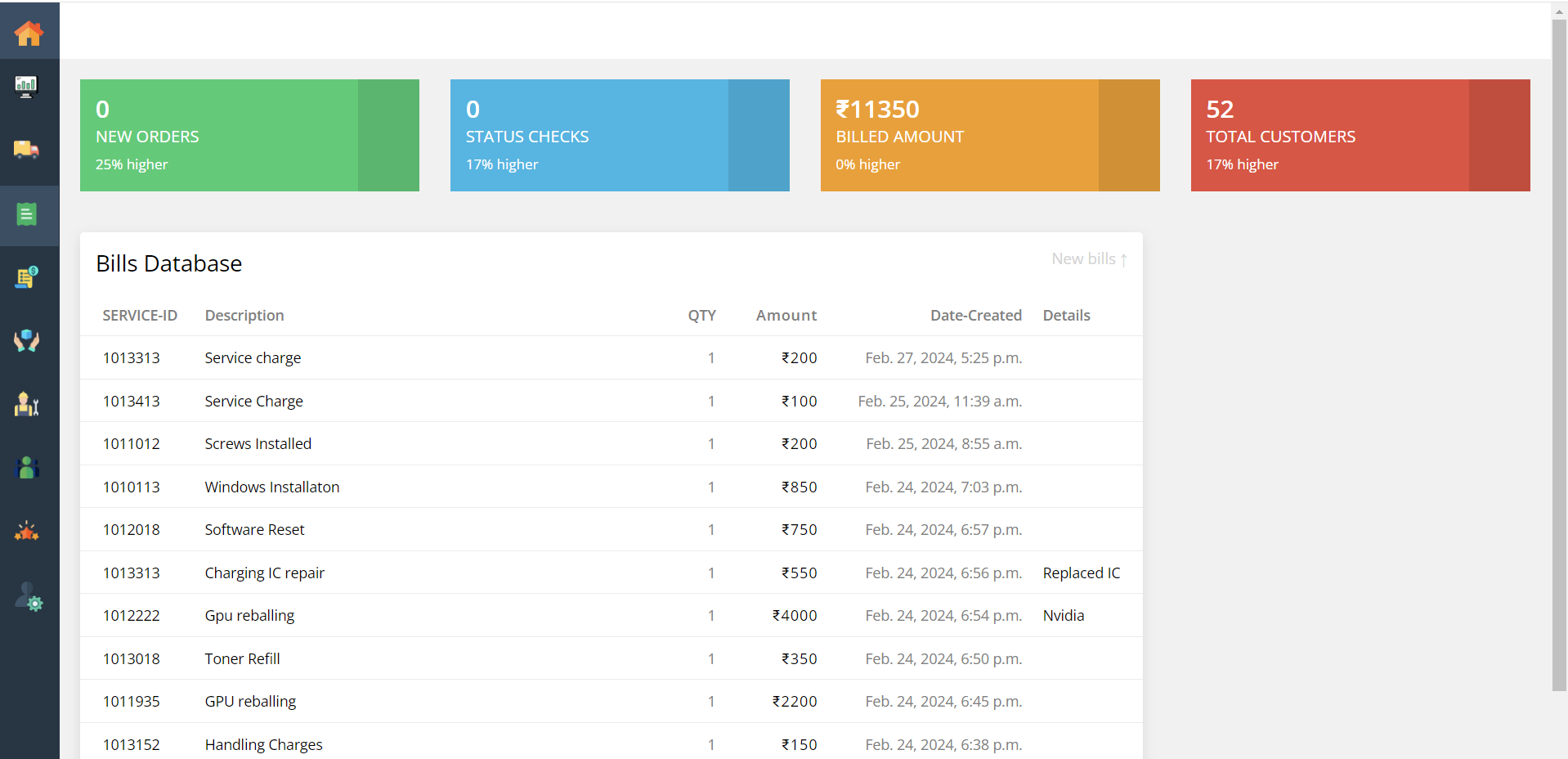
**Fig 7.4:**

****

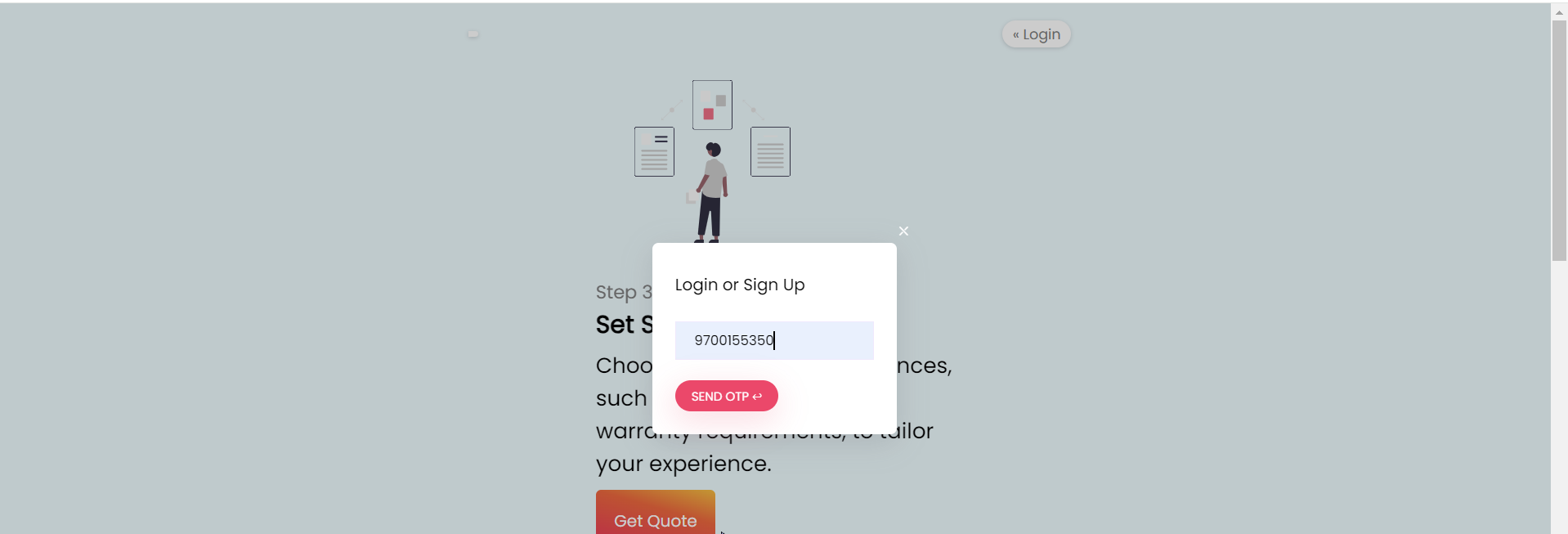
**Fig 7.5:**

****

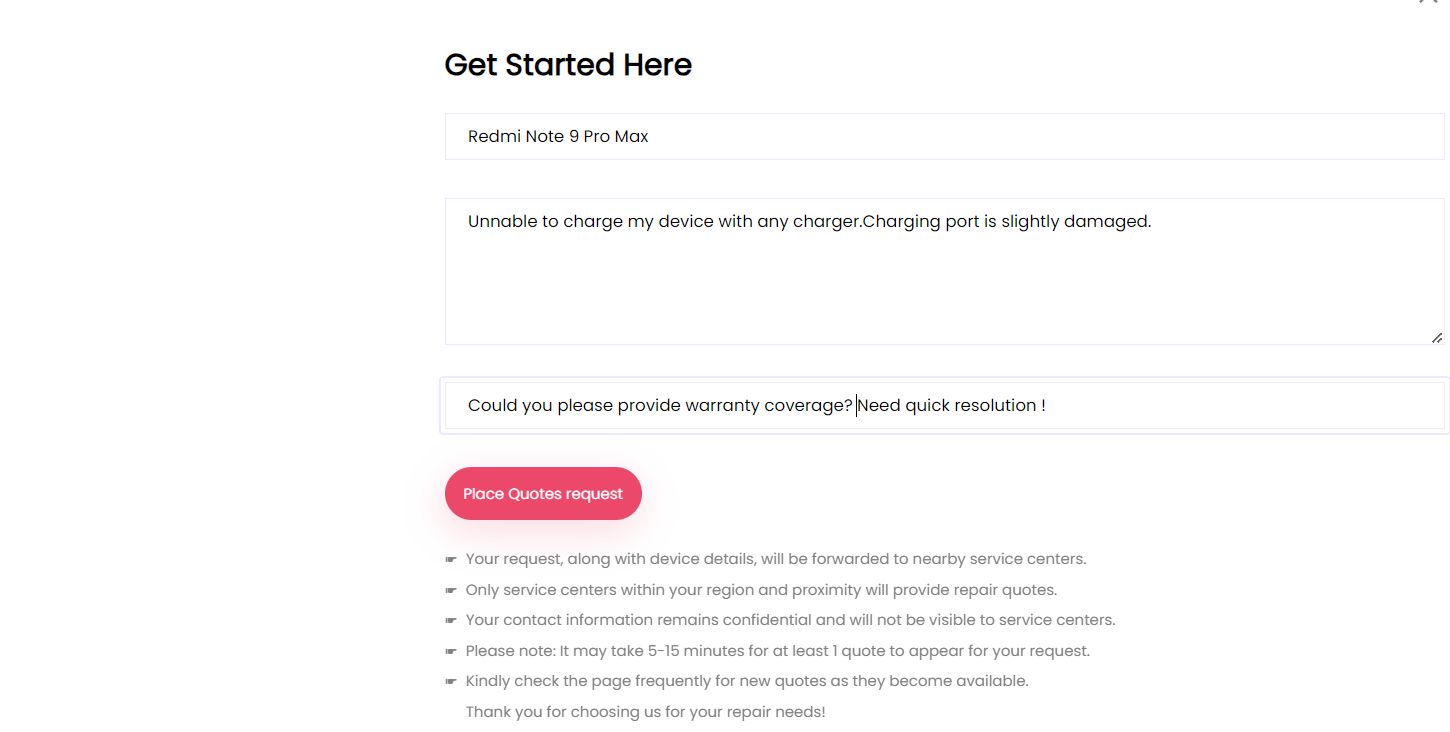
**Fig7.6:** B

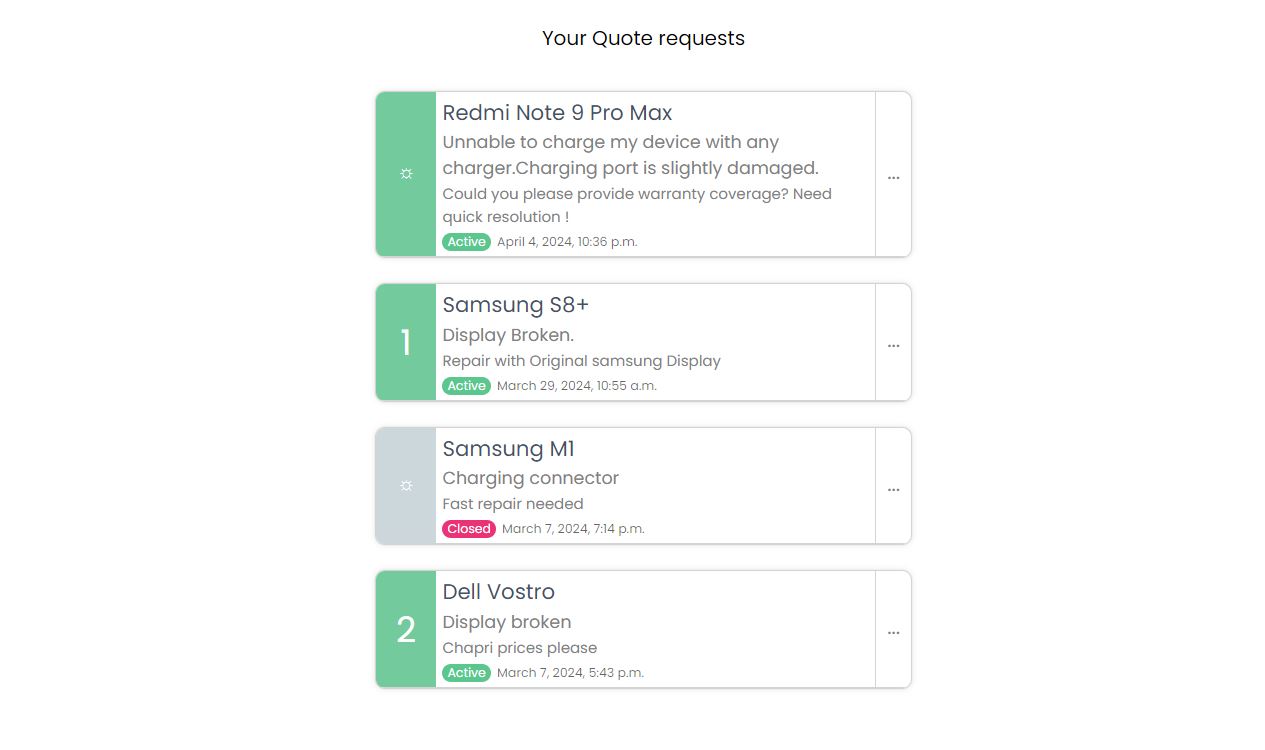
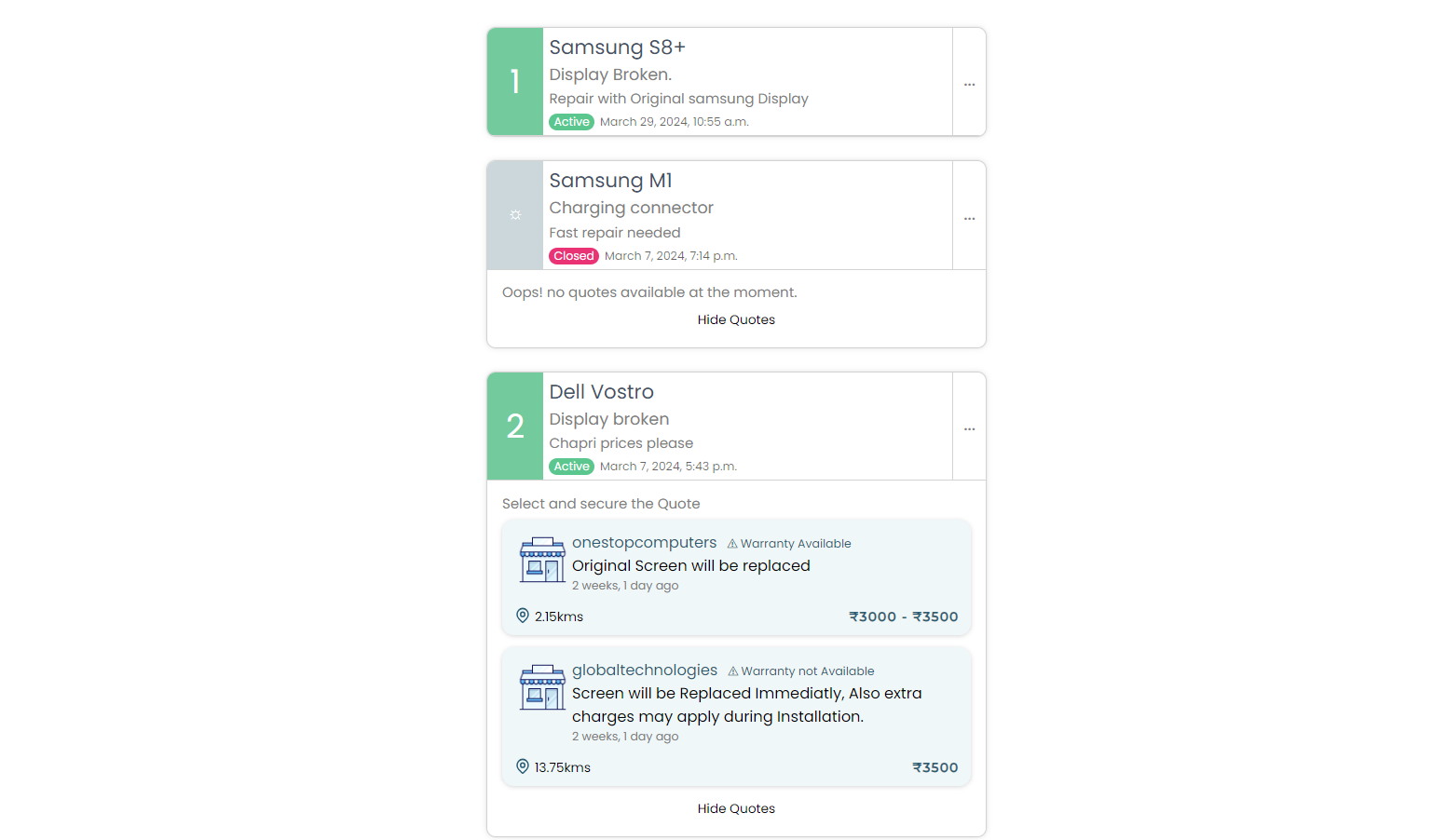
****

**Fig 7.7:** B

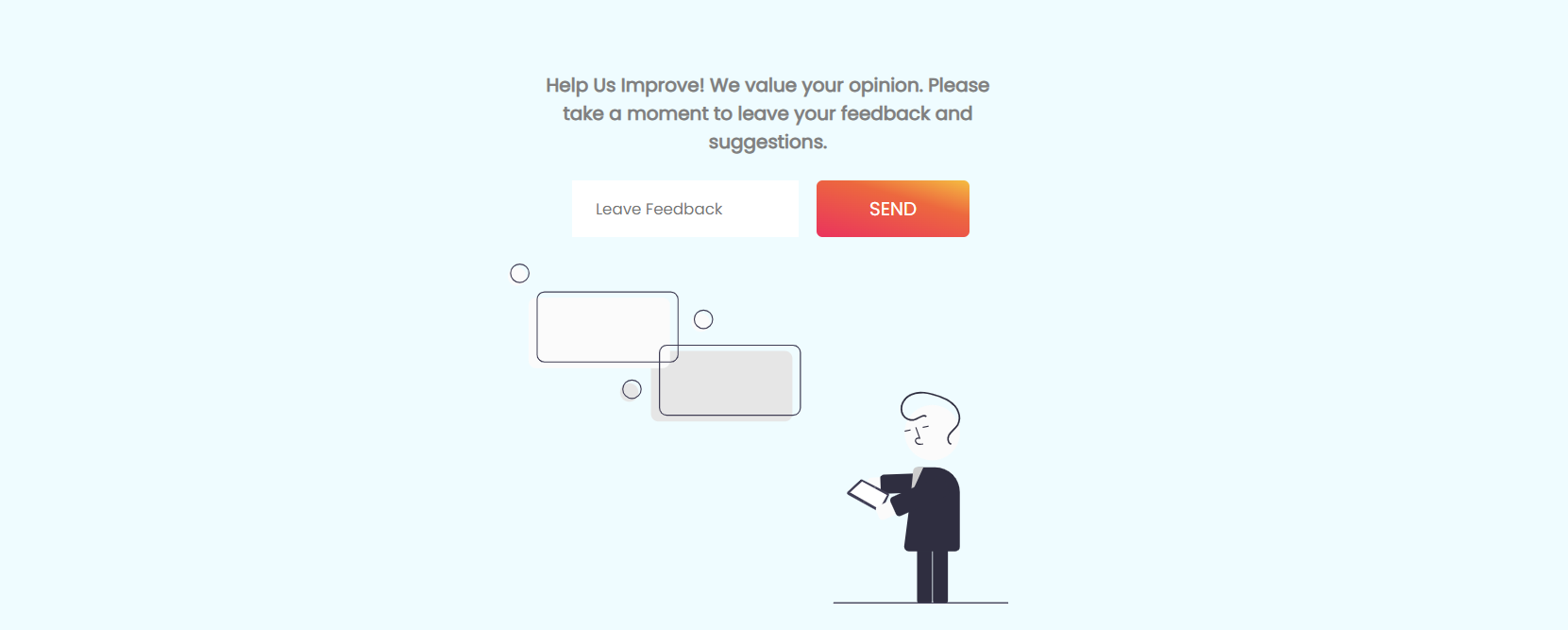
****

**Fig 7.8:** B



****

**Fig 7.9:** Bl

****

**Fig 7.10:** Customer Feedback

**CONCLUSION**

In conclusion, our SaaS application represents a transformative solution for IT service centers, enabling them to operate efficiently and grow their business in a competitive market landscape. By leveraging our comprehensive platform, service centers can streamline their operations, enhance customer interactions, and ultimately drive business growth. Through innovative features such as automated communication, vendor management, and service ticket organization, our application empowers service providers to optimize their workflow and deliver exceptional service experiences to their customers.

Moreover, our Quick Quote Service offers a powerful tool for service centers to attract and retain customers within their region. By providing a convenient and efficient way for users to obtain repair quotes from multiple service providers, our platform enables service centers to showcase their offerings and compete effectively in the market. This not only expands their customer base but also fosters loyalty and trust among existing customers, driving repeat business and referrals.

Ultimately, our SaaS application benefits both customers and service providers by bringing them together under one roof. By centralizing service discovery and provision, we create a dynamic ecosystem where users can easily access top-tier services while service providers can efficiently manage their operations and grow their business. Through our platform, we facilitate seamless interactions, enhance transparency, and foster long-lasting relationships between customers and service providers, shaping the future of service delivery in the digital age.

The Quick Quote Service within our SaaS application revolutionizes the process of accessing electronics device repairs, offering significant time-saving benefits for users. By eliminating the need for physical travel to multiple service providers, users can conveniently request and receive repair quotes from the comfort of their own homes. This not only saves valuable time but also reduces expenses associated with transportation, contributing to greater financial efficiency. Additionally, by minimizing reliance on motor vehicles for service exploration, the Quick Quote Service has a positive environmental impact, reducing fuel consumption and emissions. Overall, our platform provides a streamlined and eco-friendly solution for accessing repair services, aligning with modern values of sustainability and responsible consumption while enhancing convenience and efficiency for users.

**FUTURE ENHANCEMENT**

In our quest to enhance the application's functionality, several future enhancements have been identified. First and foremost, we plan to introduce a marketplace platform, enabling service providers to sell both new and used items exclusively. This expansion will broaden the application's offerings and provide users with a wider range of products to choose from. Additionally, we aim to enhance user experience by introducing multi-currency support, allowing transactions in various currencies beyond the current INR support. Lastly, sentiment analysis will be integrated to analyze customer feedback, providing valuable insights into user sentiment and improving overall user satisfaction. These enhancements reflect our commitment to continuously improving and evolving our application to better serve our users' needs and preferences.

**REFERENCES**

[1] Azvine, B., Nauck, D., Ho, C., Broszat, K., & Lim, J. (2006). Intelligent process analytics for CRM. BT technology journal, 24, 60– 69.

[2] Öztaysi, B., Kaya, T., &Kahraman, C. (2011). Performance comparison based on customer relationship management using analytic network process. Expert Systems with Applications, 38, 9788–9798.

[3] Torkzadeh, G., Chang, J. C.-J., & Hansen, G. W. (2006). Identifying issues in customer relationship management at Merck-Medco. Decision Support Systems, 42, 1116–1130.

[4] Phan, D. D., & Vogel, D. R. (2010). A model of customer relationship management and business intelligence systems for catalog and online retailers. Information & Management, 47, 69–77.

[5] King, S. F., & Burgess, T. F. (2008). Understanding success and failure in customer relationship management. Industrial Marketing Management, 37, 421–431.

[6] Richards, K. A., & Jones, E. (2008). Customer relationship management: Finding value drivers. Industrial Marketing Management, 37, 120–130.

[7] Malthouse, E. C., Haenlein, M., Skiera, B., Wege, E., & Zhang, M. (2013). Managing customer relationships in the social media era: Introducing the social CRM house. Journal of Interactive Marketing, 27(4), 270–280. https://doi.org/10.1016/j.intmar. 2013.09.008

[8] Shukla, M. K., & Pattnaik, P. N. (2018). Managing customer relations in a modern business environment: Towards an ecosystem-based sustainable CRM model. Journal of Relationship Marketing, 18(1), 17–33. https://doi.org/10.108 0/15332667.2018.1534057

[9] Simanjuntak, M., Putri, N. E., Yuliati, L. N., & Sabri, M. F. (2020). Enhancing customer retention using a customer relationship management approach in the car loan business. Cogent Business and Management

[10] Smith, M., & Chang, C. (2010). Improving customer outcomes through the implementation of customer relationship management: Evidence from Taiwan. Asian Review of Accounting, 18(3), 260–285.