

IT TICKETING SYSTEM: CENTRALIZED INCIDENT MANAGEMENT & RESOLUTION TRACKING

Industry Oriented Mini Project

Submitted To

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

IN PARTIAL FULFILMENT OF THE ACADEMIC REQUIREMENT FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING



SUBMITTED BY:

Redha Zidan

22081A05C9

Syed Abdul Sattar Quadri

22081A05F0

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SHADAN COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institute Accredited with NAAC A+ & NBA)

PEERANCHERU, HYDERABAD-500086

(AFFILIATED TO JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY-HYD)

HYDERABAD-500085

2024 – 2025

**IT TICKETING SYSTEM: CENTRALIZED INCIDENT MANAGEMENT &
RESOLUTION TRACKING**

Industry Oriented Mini Project

Submitted To

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

IN PARTIAL FULFILMENT OF THE ACADEMIC REQUIREMENT FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING



UNDER THE GUIDANCE OF

Dr. G. SRIDHAR

SUBMITTED BY

Redha Zidan

22081A05C9

Syed Abdul Sattar Quadri

22081A05F0

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SHADAN COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institute Accredited with NAAC A+ & NBA)

PEERANCHERU, HYDERABAD-500086

(AFFILIATED TO JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY-HYD)

HYDERABAD-500085

2024 – 2025

CERTIFICATE

This is to certify that the **dissertation** entitled "**IT TICKETING SYSTEM: CENTRALIZED INCIDENT MANAGEMENT & RESOLUTION TRACKING**" is a Bonafide work done.

Submitted by

Redha Zidan - 22081A05C9

Syed Abdul Sattar Quadri - 22081A05F0

In partial fulfilment of requirement for the award of degree of Bachelor of Technology in COMPUTER SCIENCE AND ENGINEERING, SHADAN COLLEGE OF ENGINEERING AND TECHNOLOGY, Affiliated to Jawaharlal Nehru Technological University, Hyderabad, is a record of Bonafide work carried out by them under our guidance and supervision.

INTERNAL EXAMINER

HEAD OF THE DEPARTMENT

EXTERNAL EXAMINER

DECLARATION

I hereby declare that the project report entitled “IT TICKETING SYSTEM: CENTRALIZED INCIDENT MANAGEMENT & RESOLUTION TRACKING” submitted by me to SHADAN COLLEGE OF ENGINEERING & TECHNOLOGY, Hyderabad, in partial fulfillment of the requirement for the award of the degree of B. TECH in COMPUTER SCIENCE AND ENGINEERING is a record of Bonafide.

Project work carried out by our team under the guidance of DR. G. SRIDHAR, further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

DATE:

SIGNATURE OF STUDENT

REDHA ZIDAN

: 22081A05C9 ()

SYED ABDUL SATTAR QUADRI

: 22081A05F0 ()

ACKNOWLEDGEMENT

- I thank GOD almighty for guiding us throughout the project.
- I would like to thank our parents without whose blessings; we would not have been able to accomplish our goal.
- I would thank **Dr. ATEEQ UR RAHMAN**, PRINCIPAL and all staff members of our college and friends for extending their cooperation during our project.
- I would thank **Dr. S.A. MUNEEM**, DIRECTOR and all staff members of our college and friends for extending their cooperation during our project.
- I would like to thank our internal guide **DR. G. SRIDHAR**, for all the help and support he extended to us.
- I am extremely grateful to **Dr. G. SRIDHAR**, HEAD OF THE DEPARTMENT OF CSE, for providing us with best faculties and the atmosphere for the creative work, guidance, and encouragement.
- I would also like to thank all those who have contributed to the completion of the project and help us with valuable suggestions and improvement.

SUBMITTED BY

REDHA ZIDAN : 22081A05C9 ()

SYED ABDUL SATTAR QUADRI : 22081A05F0 ()

IT TICKETING SYSTEM: CENTRALIZED INCIDENT MANAGEMENT & RESOLUTION TRACKING

TABLE OF CONTENTS

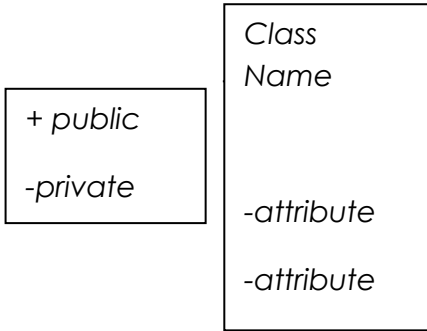
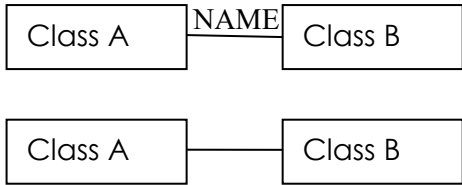
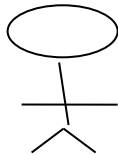
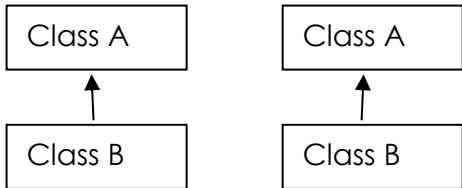
TITLE	PAGE NO.
LIST OF FIGURES	iii
LIST OF SYMBOLS	iv-vi
LIST OF ABBREVIATION	vii
ABSTRACT	viii
 CHAPTER 1: INTRODUCTION	 1-6
1.1 GENERAL	2
1.2 SCOPE OF THE PROJECT	3
1.3 OBJECTIVE	3
1.4 LITERATURE SURVEY	4-5
1.5 EXISTING SYSTEM	6
1.5.1 EXISTINGSYSTEM DISADVANTAGES	6
1.6 PROPOSED SYSTEM	6
1.6.1 PROPOSED SYSTEM ADVANTAGES	6
 CHAPTER 2: PROJECT DESCRIPTION	 7-12
2.1 GENERAL	7
2.2 PROBLEM DEFINATION	7
2.3 METHODOLOGIES	8
2.3.1 MODULES	8
2.3.2 MODULES EXPLANATION	8-9
2.3.3 MODULE DIAGRAM	9
2.3.4 GIVEN INPUT AND EXPECTED OUTPUT	10-11
2.4 TECHNIQUE / ALGORITHM	11-12
 CHAPTER 3: REQUIREMENTS	 18-19
3.1 GENERAL	13
3.2 HARDWARE REQUIREMENTS	14
3.3 SOFTWARE REQUIREMENTS	15
3.4 FUNCTIONAL REQUIREMENTS	16
3.5 NON-FUNCTIONAL REQUIREMENTS	17
3.6 DOMAIN REQUIREMENTS	18-19

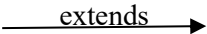


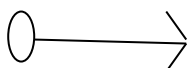
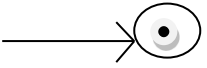
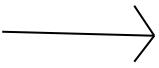
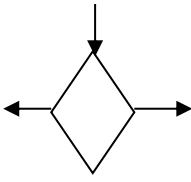
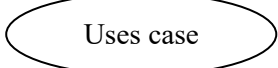
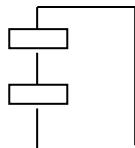
CHAPTER 4: SYSTEM DESIGN	20-40
4.1 GENERAL	20
4.2 SYSTEM ARCHETECTURE	21
4.3 UML DIAGRAMS	22
4.3.1 USE CASE DIAGRAM	22
4.3.2 CLASS DIAGRAM	23
4.3.3 OBJECT DIAGRAM	24
4.3.4 COMPONENT DIAGRAM	25
4.3.5 SEQUENCE DIAGRAM	26
4.3.6 COLLABORATION DIAGRAM	27
4.3.7 STATE DIAGRAM	28
4.3.8 ACTIVITY DIAGRAM	29
4.3.9 DEPLOYMENT DIAGRAM	30
4.4 DATA FLOW DIAGRAM	31-32
4.5 ER DIAGRAM	33
4.6 GUI DESIGN	34
4.6.1 COMPONENTS OF GUI	35-38
4.6.2 FEATURES OF GUI	39-40
 CHAPTER 5: IMPLEMENTATION	 41-69
5.1 GENERAL	41
5.2 IMPLEMENTATION	41-69
 CHAPTER 6: SNAPSHOTS	 70-76
6.1 GENERAL	70
6.2 OUTPUT SNAPSHOTS	71-75
 CHAPTER 7: SOFTWARE TESTING	 76-77
7.1 GENERAL	76
7.2 DEVELOPING METHODOLOGIES	76
7.3 TYPES OF TESTING	77
 CHAPTER 8:	 78-79
8.1 FUTURE ENHANCEMENTS	78
8.2 CONCLUSION	79
8.3 REFERENCES	79

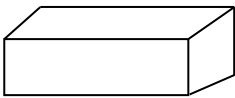
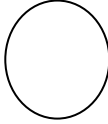
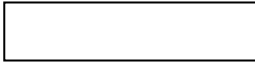

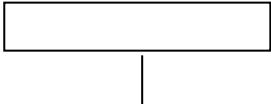
LIST OF FIGURES

FIGURE NO	NAME OF THE FIGURE	PAGE NO.
2.3.3	Module Diagram	9
4.3.1	Use case Diagram	22
4.3.2	Class diagram	23
4.3.3	Object diagram	24
4.3.4	Component Diagram	25
4.3.5	Deployment Diagram	26
4.3.6	Sequence diagram	27
4.3.7	Collaboration diagram	28
4.3.8	State Diagram	29
4.3.9	Activity Diagram	30
4.4	Data flow diagram	31
4.5	ER Diagram	33
4.6	GUI Design	34

LIST OF SYMBOLS

S.NO	NOTATION NAME	NOTATION	DESCRIPTION
1.	Class		Represents a collection of similar entities grouped together.
2.	Association		Associations represent static relationships between classes. Roles represents the way the two classes see each other.
3.	Actor		It aggregates several classes into a single class.
4.	Aggregation		Interaction between the system and external environment

5.	Relation (uses)	uses	Used for additional process communication.
6.	Relation (extends)		Extends relationship is used when one use case is similar to another use case but does a bit more.
7.	Communication		Communication between various use cases.
8.	State		State of the processes.
9.	Initial State		Initial state of the object
10.	Final state		Final state of the object
11.	Control flow		Represents various control flow between the states.
12.	Decision box		Represents decision making process from a constraint
13.	Use case		Interact ion between the system and external environment.
14.	Component		Represents physical modules which are a collection of components.

15.	Node		Represents physical modules which are a collection of components.
16.	Data Process/State		A circle in DFD represents a state or process which has been triggered due to some event or action.
17.	External entity		Represents external entities such as keyboard, sensors, etc.
18.	Transition		Represents communication that occurs between processes.
19.	Object Lifeline		Represents the vertical dimensions that the object communications.

ABSTRACT

In today's fast-paced digital environments, efficient management of technical issues is crucial for ensuring smooth organizational operations. The **IT Ticketing System: Centralized Incident Management & Resolution Tracking** aims to streamline the process of reporting, tracking, and resolving IT-related issues within an organization or institution. This project focuses on the development of an automated ticketing system that simplifies issue management by enabling users to report problems, and administrators or support teams to manage and resolve them effectively.

The system allows users to create tickets for various IT concerns such as hardware malfunctions, software errors, network issues, and general technical queries. Each ticket includes relevant details like the issue, description, date, and status. Once a ticket is submitted, it is automatically logged into the system's database and categorized for quick allocation to the appropriate support personnel.

To ensure efficient issue resolution, the system includes features such as ticket status tracking. Admins can assign tickets to support agents, monitor their progress, and close them upon successful resolution. The system also maintains a history of all tickets, enabling analysis of recurring problems and overall performance metrics.

The goal of this project is to improve the responsiveness and organization of IT support teams by reducing manual effort, minimizing resolution time, and enhancing communication between users and technical staff. The IT Ticketing System is designed to be scalable, reliable, and adaptable to various organizational needs, making it a vital tool for maintaining technological infrastructure.

CHAPTER-1

INTRODUCTION

In today's digital age, IT infrastructure forms the backbone of every organization—be it educational institutions, corporate offices, or government departments. As dependency on technology grows, so does the demand for efficient handling of technical issues. However, many organizations still rely on outdated methods such as emails, phone calls, or manual logs to manage IT-related problems. These approaches often lead to confusion, delays, and poor issue tracking.

To address these challenges, our project introduces an **IT Ticketing System: Centralized Incident Management & Resolution Tracking**, a centralized platform designed to streamline the reporting, assignment, tracking, and resolution of technical issues. The system allows users to raise tickets whenever they encounter an IT-related problem, such as hardware failures, software errors, or network disruptions. Each ticket is logged into the system with relevant details like, description, and status.

The main goal of this project is to enhance the efficiency of IT support teams by automating the issue management process. The system enables administrators to assign tickets to specific staff members, monitor progress, and maintain clear communication with users. It also includes features such as ticket history and status updates, ensuring transparency and accountability throughout the resolution process.

1.1 GENERAL

In the modern world, organizations heavily rely on IT infrastructure to manage their day-to-day operations. As the complexity and scale of these systems grow, so does the need for a reliable method to report and manage technical issues. Without an efficient system in place, tracking problems, assigning tasks, and ensuring timely resolutions can become difficult and disorganized.

The IT Ticketing System addresses this challenge by offering a structured solution for handling IT-related issues. It enables users to raise tickets for any technical problem they face and allows IT staff to manage and resolve these issues systematically. This system not only ensures quick communication between users and the support team but also helps maintain a clear record of all reported issues for future reference and analysis.

By centralizing the issue management process, the IT Ticketing System improves overall operational efficiency, reduces response time, and enhances the performance of IT support services within an organization.

1.2 SCOPE OF THE PROJECT

This project presents a comprehensive web-based IT Ticket Management System designed to streamline the process of reporting, tracking, and resolving technical issues within an organization. Aimed at addressing the common challenges faced by IT departments, the system is ideal for deployment in educational institutions, corporate environments, government offices, and other organizations that require structured IT support services.

The platform supports essential functionalities including user registration and authentication, ticket creation, manual assignment of tickets to IT support staff, real-time progress tracking, and detailed reporting for analysis and performance monitoring. Users can submit issues related to hardware, software, networking, or any department-specific requirement, while IT support teams can manage and resolve them efficiently through a centralized interface.

Designed with scalability in mind, the system can be easily extended to support multiple departments, diverse issue types, and role-based access for employees, IT staff, and administrators.

Though initially developed for internal use within a single organization, the system architecture is flexible enough to be adapted for external client support, enabling service providers to manage customer tickets effectively. This adaptability makes the IT Ticketing System a valuable tool for improving communication, and operational efficiency in IT service management.

1.3 OBJECTIVE

The primary objective of the IT Ticketing System is to develop a centralized platform for managing IT-related issues. The system aims to:

- Simplify the process of raising and tracking technical issues.
- Assign tickets to appropriate personnel.
- Maintain a database of reported issues for future analysis.
- Provide real-time updates on ticket status to both users and admins.
- Enhance the overall responsiveness of IT support teams.

1.4 LITERATURE SURVEY

Title: *Machine Learning Approaches for Helpdesk Ticketing System: A Systematic Literature Review*

Authors: Alvian Shanardi Wijaya & Tanty Oktavia

Year: 2024

Description: This comprehensive review investigates machine learning applications in helpdesk ticketing from 2012 to 2022. It documents the use of classification, and resource allocation through ML models, highlighting emerging trends such as automation of triage and intent detection. The study identifies gaps and future directions in ML-driven helpdesk, providing a valuable roadmap for integrating intelligent systems in IT support environments.

Title: *Enhancing a Ticketing System with AI Intelligence*

Author: Naveen Koka

Year: 2024

Description: This study explores how artificial intelligence, particularly machine learning and NLP, can enhance traditional ticketing platforms. The system automates ticket assessment using sentiment analysis, and intelligent routing to available agents—significantly speeding up response times. By suggesting solutions from historical ticket data, the system empowers agents to resolve issues faster and reduces manual workload.

Title: *TaDaa: Real-Time Ticket Assignment Deep Learning Auto Advisor*

Author: Leon Feng, Jnana Senapati, Bill Liu

Year: 2022

Description: TaDaa utilizes transformer-based deep learning to automate ticket assignment within large-scale organizations. It assigns tickets both at the group and individual staff levels and suggests relevant past tickets to aid in resolution. Tested on a dataset of over 13,000 groups and 10,000 resolvers, it achieved top-3 assignment accuracy of 95.2% and top-5 accuracy of 79.0%, demonstrating high effectiveness in real-world environments.

Title: Design and Implementation of Web-Based Help Desk System

Author: Dr. R. Selvarani, A. Rajkumar

Year: 2021

Description: The paper describes the development of a web-based help desk system to improve customer service efficiency. The proposed system was evaluated in an academic environment, achieving faster response times and more structured issue handling.

Title: Automation of IT Service Management

Author: S. Varghese, K. Mahesh

Year: 2020

Description: Focused on automating IT issue resolution using a ticketing model integrated with AI for predictive analysis. The project highlights the benefits of auto-classification and efficient workload distribution. Various IT service management tools like Jira Service Desk, ServiceNow, and Zoho Desk provide extensive solutions for issue tracking.

However, these tools are often complex and expensive for smaller organizations. Research has shown that custom-built ticketing systems can offer cost-effective and user-friendly alternatives tailored to specific organizational needs. The literature suggests that automation and structured workflows significantly improve response times and issue resolution effectiveness.

1.5 EXISTING SYSTEM:

In many organizations, issue tracking is still done manually using emails, phone calls, or written logs. Some may use basic spreadsheet tracking, which lacks proper record-keeping. These methods are inefficient, prone to human error, and make it difficult to track the status and history of tickets accurately.

1.5.1 EXISTING SYSTEM DISADVANTAGES:

- No centralized database for ticket history.
- Poor tracking of issue resolution timelines.
- Difficult to assign and monitor tasks among support staff.
- Lack of status updates.
- Increased workload on IT teams due to redundant communications.

1.6 PROPOSED SYSTEM

The proposed IT Ticketing System is a custom-developed application designed to overcome the limitations of existing methods. It features user authentication, ticket submission with ticket assignment, and real-time status tracking. The system ensures that every issue is properly logged, assigned, tracked, and resolved within a structured workflow.

1.6.1 PROPOSED SYSTEM ADVANTAGES:

- Centralized and structured ticket management.
- Improved communication between users and IT staff.
- Faster resolution of technical problems.
- Easy access to historical data for reporting and analysis.
- Customizable according to organizational requirements.

CHAPTER 2

PROJECT DESCRIPTION

2.1 GENERAL

This initiative aims to streamline and enhance IT support services by introducing an efficient web-based IT ticketing system. Leveraging modern web development technologies, the project focuses on managing and resolving technical issues within organizations through a structured and automated ticket handling process. Moving beyond traditional manual logging methods, our system enables tracking of tickets. It includes a user-friendly front-end interface built with HTML and Bootstrap. It integrates a Django-based backend for real-time ticket management and updates. Designed with a focus on usability, accuracy, and accessibility, this project offers an affordable and scalable solution for improving IT support workflows and reducing resolution time.

2.2 PROBLEM DEFINITION

In many organizations, handling IT-related issues manually often leads to delays, miscommunication, and unresolved queries, ultimately affecting productivity. The absence of a centralized system to log, track, and manage technical problems creates inefficiencies in addressing support requests. Critical issues may go unnoticed or unresolved for extended periods. This project addresses these challenges by developing a structured IT ticketing system that allows users to submit their issues efficiently, while enabling the IT support team to monitor and resolve tickets in a systematic and timely manner. The system aims to enhance transparency, accountability, and overall effectiveness of IT support operations.

2.3 METHODOLOGIES

2.3.1 MODULES

- User Authentication and Authorization
- Ticket Submission Form
- Ticket Tracking and Status Updates
- Admin Dashboard for Ticket Management
- Database Integration and Storage

2.3.2 MODULES EXPLANATION

1) User Authentication and Authorization:

This module manages secure access to the system by implementing login and registration functionalities. Employees and IT Staff can log in using designated credentials, while admins have their own dedicated authentication and elevated privileges. Authorization ensures that Actors only access features relevant to their roles (e.g., employees can raise tickets, while admins can manage all tickets). This prevents unauthorized access and maintains data integrity.

2) Ticket Submission Form:

This is the core interface for end users to report technical issues. The form typically includes fields such as issue title, description, concerned department (e.g., finance, HR, IT) and reward points. Once submitted, the ticket is stored in the database and made available to the IT team for review.

3) Ticket Tracking and Status Updates:

This module allows employees, IT Staff and admins to track the progress of each ticket. Statuses such as *Open*, *In Progress*, *Resolved*, and *Closed* provide transparency. Employees can view updates or responses from the IT team, while admins can change ticket statuses and add comments to keep the communication flow active.

4) Admin Dashboard for Ticket Management:

The admin dashboard provides an overview of all active, pending, and closed tickets. Admins can view statistics, filter by parameter and manage tickets through the database directly. This centralized control panel improves visibility and enables better auditing capabilities for the administrators.

5) Database Integration and Storage:

All ticket data, identity(and access) information, and system logs are stored securely in a backend database. This module ensures reliable data storage, retrieval, and management. It plays a vital role in ensuring consistency, supporting analytics, and maintaining system performance.

2.3.3 MODULE DIAGRAM

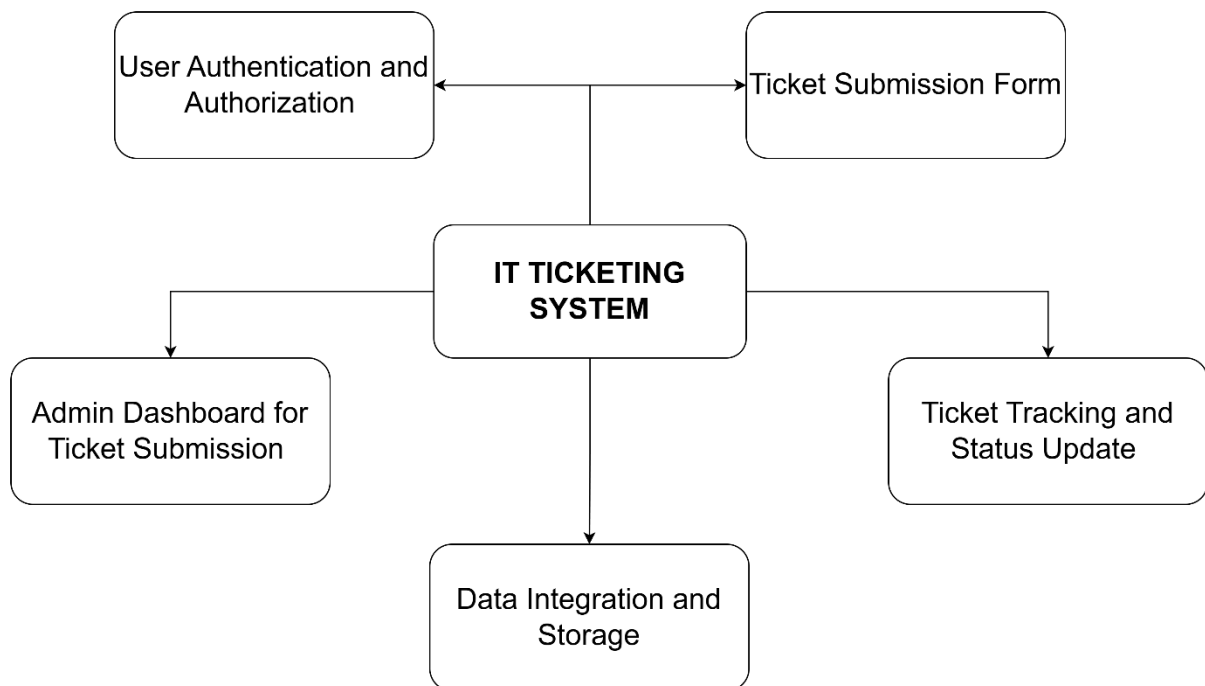


Fig 2.3.3 Module Diagram

2.3.4 GIVEN INPUT AND EXPECTED OUTPUT

S.No.	Given Input	Expected Output
1	Employee logs in with valid credentials.	Redirected to the Employee dashboard or home screen.
2	Employee raises a new ticket with issue details.	Ticket is successfully recorded in the system and a ticket ID is generated.
3	IT staff logs in and views all raised tickets.	IT staff dashboard displays a list of all tickets with status options.
4	IT staff accepts the ticket and updates the status of a ticket to "In Progress" or "Resolved".	Ticket status is updated and reflected in both IT Staff and employee views.
5	Employee attempts to raise a ticket without filling out the required fields.	System prompts an error message requesting all mandatory fields to be filled.
6	Database stores all ticket, employees, IT staff and admin data securely.	All data is safely saved and can be retrieved or modified as required.

The IT ticketing system is designed to handle a variety of user inputs and generate expected outputs in a structured and reliable manner. When an actor enters valid login credentials, the system verifies them and redirects the user to their dedicated dashboard of their concerned role, ensuring a secure and authenticated access experience. Once logged in, employees can raise new tickets by filling out a form with essential details such as the issue title and description (e.g., software, hardware, network),

Upon raising, the system successfully records the ticket in the database, assigns it a unique ticket ID, and makes it available to the IT staff for review and resolution.

IT Staff, upon logging in, are granted access to a dedicated dashboard where they can view a list of all submitted tickets. This dashboard provides filtering and sorting capabilities based on status or date, allowing the IT Staff to efficiently manage tickets.

When an IT Staff updates the status of a ticket to “In Progress,” “Resolved,” or “Closed,” the system processes this change and displays the updated status at all the sites.

Furthermore, all data generated through raised tickets, user profiles, admin actions, and system updates are securely stored in a centralized database. This database ensures that data can be reliably retrieved, updated, or deleted as required, supporting the overall functionality and integrity of the IT ticketing system. In this way, the system efficiently converts user inputs into meaningful actions and responses, ensuring an organized, transparent, and user-friendly support process.

2.4 TECHNIQUE / ALGORITHM

The IT Ticketing System is developed using Python as the core programming language and Django as the backend web framework, which provides a powerful structure for developing secure and scalable web applications. Django's Model-View-Template (MVT) architecture is used to separate concerns and streamline the flow of data between the backend and frontend. One of the key techniques implemented in this project is Role-Based Access Control (RBAC), which governs how users interact with the system based on their assigned roles—Admin, IT Staff or Employee. This technique ensures that the data and functions are accessible only to authorized individuals, thus maintaining the integrity and security of the system.

The authentication system in Django has been utilized to create two distinct login routes: one for Employees and another for admins. Upon logging in, each user is redirected to their respective dashboard. Employees can raise support tickets by submitting a form with details like ticket title and description. Their access is strictly limited to viewing and management of their own tickets. This is enforced through Django's built-in user permissions and middleware, which validate the user's role on every request and restrict access accordingly.

IT Staff are granted access to all the tickets raised so that they can view, accept, resolve and close the tickets accordingly. The accepted tickets are saved in their profile under the My Tickets tab.

Admins, on the other hand, are granted full access to the ticket database. They can view all tickets, database objects, Identity & Access roles. The system dynamically filters views and functionalities based on the user's role. For example, admin dashboards include ticket management panels, charts, and filters, which are hidden from regular users. This separation ensures that data handling capabilities are strictly reserved for authorized personnel.

Additionally, Django's ORM (Object-Relational Mapper) is employed for seamless database interactions. Models are defined for tickets, users, and roles, making it easy to query and manipulate data using Python code without writing raw SQL. This enhances both security and development speed. The use of Django's built-in decorators and class-based views further simplifies access control by allowing restrictions to be applied at the view level.

CHAPTER 3

REQUIREMENTS

3.1 GENERAL

The IT Ticketing System is developed as a web-based platform with a focus on structured support management and role-based access. Leveraging Django's powerful backend capabilities and Python's simplicity, the system offers secure user interactions and efficient ticket handling. Due to the clear separation between user and admin functionalities, the platform ensures data security while simplifying issue resolution workflows. The use of Django's ORM significantly reduces database management complexity, and the built-in authentication system ensures only authorized users can access sensitive operations. Preliminary testing shows that ticket creation, status updates, and role-based views perform reliably across environments. Compared to traditional manual support methods, our system demonstrates a notable improvement in processing speed, access control, and user engagement, making it a competitive solution for IT support.

3.2 HARDWARE REQUIREMENTS

The hardware requirements serve as a foundational guideline for deploying and operating the IT Ticketing System efficiently. These specifications are critical to ensuring that the system performs optimally under expected usage conditions. They act as a reference point for system developers and engineers during the design, testing, and deployment phases. While the implementation approach may vary, the hardware requirements define what the system must support in terms of processing power, memory, and storage, independent of the actual development methodology.

To ensure smooth execution and functionality of the application, the following minimum hardware specifications are recommended:

- **Processor** : INTEL CORE 2 DUO or higher
- **RAM** : 4 GB DDR3 RAM or above
- **Hard Disk** : 250 GB minimum free space

These specifications ensure that the development environment can handle the backend processing, database transactions, and web-based interactions without performance lags. Systems meeting or exceeding these hardware requirements will be capable of running the IT Ticketing System reliably in both development and production settings.

3.3 SOFTWARE REQUIREMENTS

The software requirements define the essential tools, platforms, and technologies needed for the development and execution of the IT Ticketing System. This specification outlines what the system should do rather than how it should do it. The software requirements serve as a foundation for preparing the Software Requirements Specification (SRS) document and are crucial for estimating development cost, allocating team responsibilities, planning tasks, and tracking the overall progress of the project.

These requirements encompass the operating environment, development tools, programming languages, and frameworks necessary to build a robust and maintainable system. The selected technologies must support web-based interactions, backend processing, database integration, and ensure user-friendly interfaces.

The software requirements for this system are as follows:

- **Operating System** : Windows 10 / 11
- **Platform / IDE** : Visual Studio Code
- **Programming Language** : Python
- **Front End Technologies** : HTML and Bootstrap
- **Framework** : Django

These tools collectively facilitate a streamlined development process, providing the environment and features required to implement core functionalities such as user authentication, ticket raising, data handling, and admin management. Python ensures flexibility and integration with other modules. The chosen software stack supports rapid development and deployment, making it suitable for both prototype and production stages.

3.4 FUNCTIONAL REQUIREMENTS

A functional specification outlines the core operations that the IT Ticketing System is expected to perform. These functions represent the core behaviours of the system as triggered by specific user actions or system events. Each functional requirement is described in terms of its input, the behaviour or processing logic applied to that input, and the expected output. These requirements act as a contract between stakeholders and developers, ensuring the system delivers the intended features in a measurable and verifiable way.

In the context of the IT Ticketing System, the functional specification includes:

- **User Registration and Login:** The system should allow new employees, IT Staff and admins to register and log in through separate interfaces, based on their role.
- **Raising Ticket:** Authenticated employees must be able to raise new tickets by submitting a form that includes the issue title, description, etc.
- **Ticket Viewing:** Employees can view the list of their own submitted tickets along with their current status and any updates made by the IT Staff.
- **Role-Based Access:** IT Staff should have access to view, update, and manage all tickets in the system, while employees should only have access to their own ticket history. On the other hand, the admin has exclusive and full access to the database for auditing and administration purposes.
- **Ticket Status Updates:** IT Staff can accept and update the ticket status (e.g., Open, In Progress, Resolved).
- **Database Integration:** All data, including user credentials, ticket information, and status updates, should be stored in a secure and consistent manner using Django's ORM with a backend database using SQLite.

3.5 NON-FUNCTIONAL REQUIREMENTS

The major non-functional requirements of the IT Ticketing System are as follows:

- **Usability:**

The system is designed to be user-friendly with an intuitive interface, minimizing the need for extensive training. Streamlined workflows reduce extensive manual intervention, making it easy for employees to submit, track, and resolve IT tickets efficiently.

- **Reliability:**

Built using robust technologies and following best coding practices, the system ensures consistent and reliable performance. It handles concurrent users and ticket updates without data loss or crashes, maintaining system stability.

- **Performance:**

The IT Ticketing System is optimized to respond quickly to user requests, such as raising of tickets, status updates, and report generation. Efficient backend processing ensures minimal delay, even under high load conditions.

- **Supportability:**

The system is designed to be cross-platform and can be deployed on various operating systems and hardware configurations. Its modular architecture makes maintenance and future upgrades straightforward and cost-effective.

- **Implementation:**

The backend logic and data processing are implemented using Python, ensuring ease of maintenance and scalability. The system interface is built using the Django framework, hosted on a Windows 11 Professional environment.

3.6 DOMAIN REQUIREMENTS

Domain requirements refer to the specific rules, standards, constraints, and functionalities that are inherent to the particular domain or industry for which the software system is being developed. These requirements go beyond general system functionalities and are often dictated by domain-specific workflows, legal regulations, business rules, or organizational practices.

Unlike generic software requirements, domain requirements are deeply rooted in the operational context of the system. They are usually derived from subject matter experts (SMEs), industry standards, and the existing environment in which the system will function. Ignoring or misinterpreting domain requirements can lead to serious functional gaps, compliance issues, or system failures after deployment.

1. Ticket Management:

The system must allow employees to raise tickets with details like issue description, concerned department, date, and rewards. Each ticket must have a unique identifier. The system must support multiple ticket statuses: Open, In Progress, Resolved, Closed and Reopened. Employees must be able to track the status of their tickets in real time. Tickets should be editable by authorized personnel before they are closed. The system must allow closing, reopening, or escalating tickets based on permissions and conditions.

2. User Role Management:

The system must define at least three user roles: Employee (Ticket raiser), IT Support Staff (Concerned technicians), Admin (Supervisors or system admins). Role-Based Access Control (RBAC) must restrict or grant permissions based on user roles. Admins must be able to add, edit, or delete users and assign roles.

3. Ticket Assignment:

IT staff can accept ticket they think they can resolve. IT staff must be able to update task progress and close tickets.

4. Audit Trail and Logging:

The system must maintain an audit trail of key actions (ticket status changes, ticket updates, user role changes, etc.). Logs must include who performed the action and when. This information is exclusively accessible by the admin.

5. Security and Data Integrity:

The system must implement user authentication and password hashing. Only authorized users should have access to sensitive data. Input validation must be enforced to prevent SQL injection, XSS, and CSRF attacks.

CHAPTER 4

SYSTEM DESIGN

4.1 GENERAL

System design is a critical phase in the software development life cycle, where the system's functional and non-functional requirements are translated into a detailed blueprint for implementation. It provides a meaningful engineering representation of the software system to be built, ensuring clarity, structure, and alignment with the specified objectives. Design engineering involves the creation of various **UML (Unified Modelling Language)** diagrams to visually model the system architecture, workflows, user interactions, and internal processes. These diagrams help developers, analysts, and stakeholders to better understand the behaviour and structure of the system before the actual development begins.

By leveraging UML diagrams, stakeholders—including project managers, analysts, developers, and end-users—gain a unified and comprehensive understanding of the proposed system. This shared understanding facilitates more informed decision-making, better collaboration, and smoother transitions between the various stages of development.

Overall, the system design phase ensures that the software product is technically feasible, logically consistent, and aligned with user expectations and business goals, forming a solid foundation for successful implementation and deployment.

4.2 SYSTEM ARCHETECTURE

The system architecture of the IT Ticketing System is based on a client – server Model, designed to ensure efficient performance, scalability, and user accessibility. The client side is the user interface that allows users to interact with the system through web pages built using HTML and Bootstrap. This interface is integrated with Django to handle user requests and display responses dynamically. The server side, developed using Django, contains the core business logic. It handles all user requests, manages sessions, processes ticket submissions, updates ticket status, and controls user authentication and authorization based on roles such as user, technician, and admin. This layer acts as a bridge between the user interface and the database. The database is the storage component of the system, which utilizes a relational database like SQLite to store and manage data related to users, tickets, comments, and ticket history. The architecture ensures that each side performs its designated task, making the system modular and easy to maintain. All employee actions—such as creating a ticket, updating status, or assigning tickets—are processed through the server and then stored or retrieved from the database. The system is developed and tested using VS Code, with deployment done on a local server running Windows 11 Professional. This client-server model ensures that the IT Ticketing System functions smoothly and can be expanded or migrated to a cloud-based environment in the future.

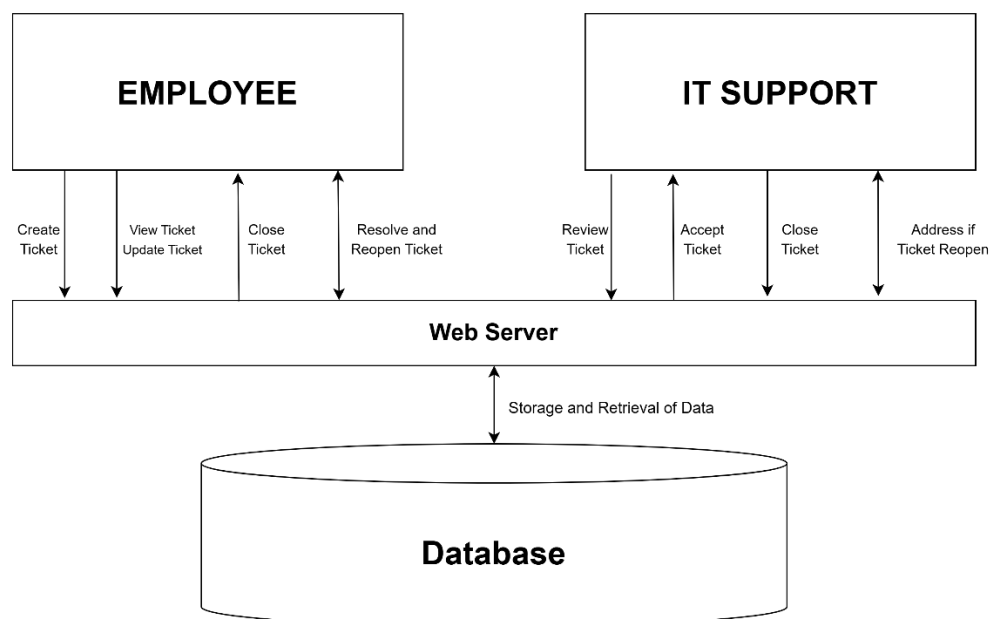


Fig 4.2 System Architecture

4.3 UML DIAGRAMS

4.3.1 USE CASE DIAGRAM

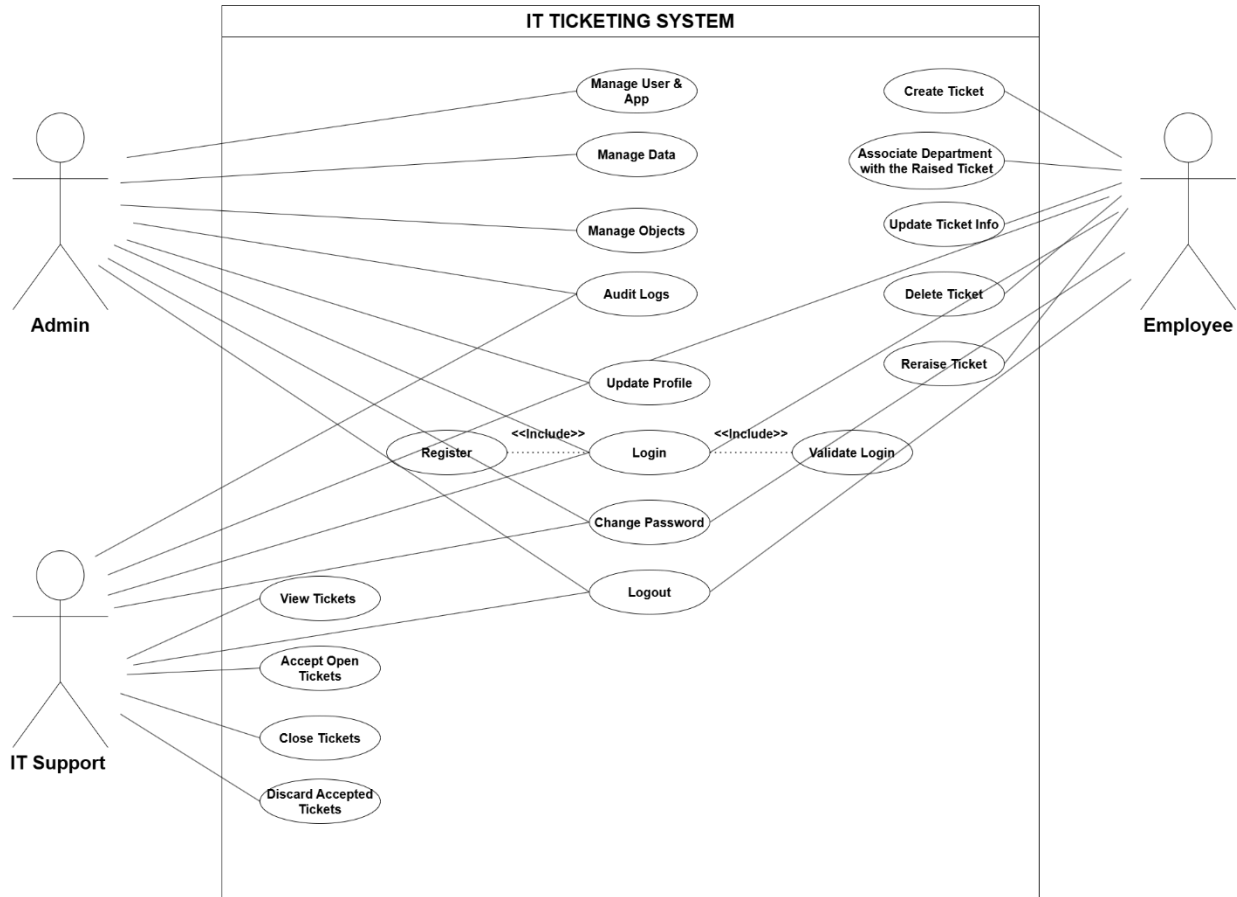


Fig 4.3.1 Use Case Diagram

EXPLANATION:

The Use Case Diagram represents the interaction between different types of actors (Admin, Employee, and IT Support Staff) and the system. It shows the major functionalities such as raising a ticket, updating the ticket status, assigning tickets, viewing ticket history, and managing users. This diagram helps identify the system's functional requirements and how each actor is involved in the process.

4.3.2 CLASS DIAGRAM

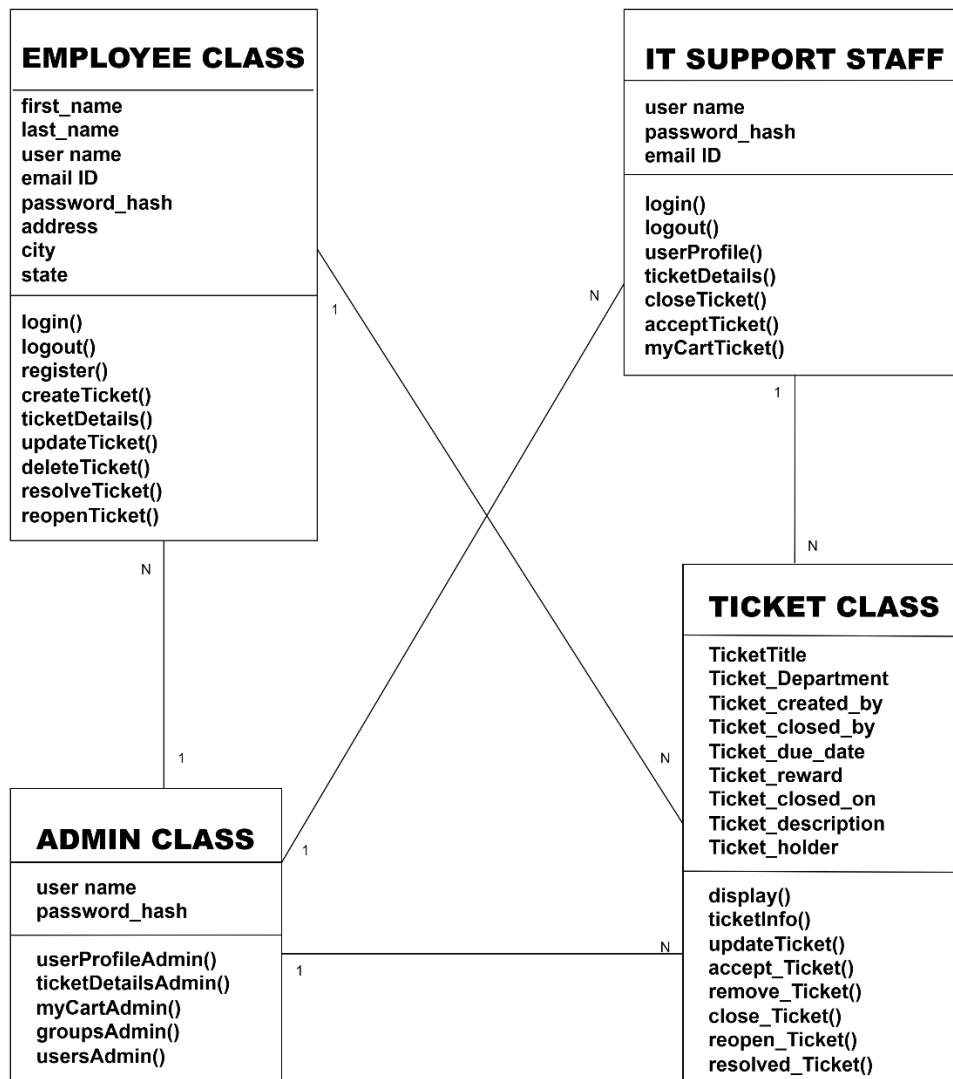


Fig 4.3.2 Class Diagram

EXPLANATION:

The Class Diagram outlines the structure of the system by showing the system's classes, attributes, methods, and relationships. Key classes in the IT ticketing system include User, Admin, Ticket, and IT Support Staff. For example, the Employee class has attributes like email & username and methods like createTicket(), reopenTicket(). Relationships such as inheritance (IT Support class inherit from the employee class) and associations (Employee raises a Ticket) are shown clearly.

4.3.3 OBJECT DIAGRAM

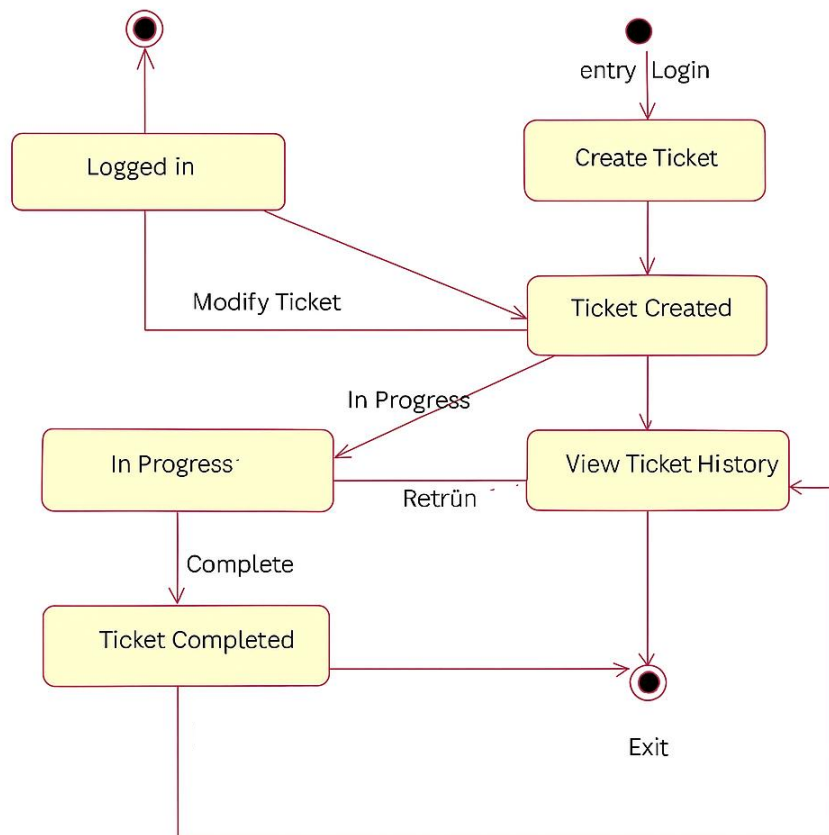


Fig 4.3.3 Object Diagram

EXPLANATION:

The Object Diagram is a snapshot of the system at a particular moment, showing instances of classes and their relationships. It visualizes how specific objects like `ticket1:Ticket`, `admin1:Admin`, and `user1:User` are linked during runtime. This helps understand real-time object interaction and memory usage.

4.3.4 COMPONENT DIAGRAM

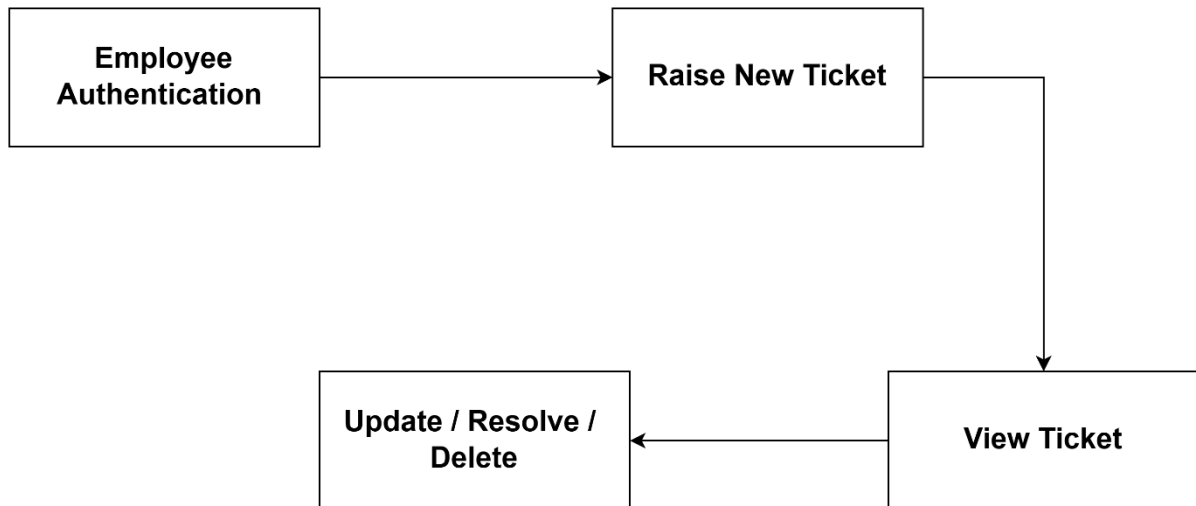


Fig 4.3.4 Component Diagram

EXPLANATION:

The Component Diagram represents the high-level software components and their dependencies. It shows components such as the User Interface, Ticket Management Module, Authentication Module, and Database. Arrows between components indicate dependencies and data flow, helping visualize the modular architecture of the IT Ticketing System.

4.3.5 SEQUENCE DIAGRAM

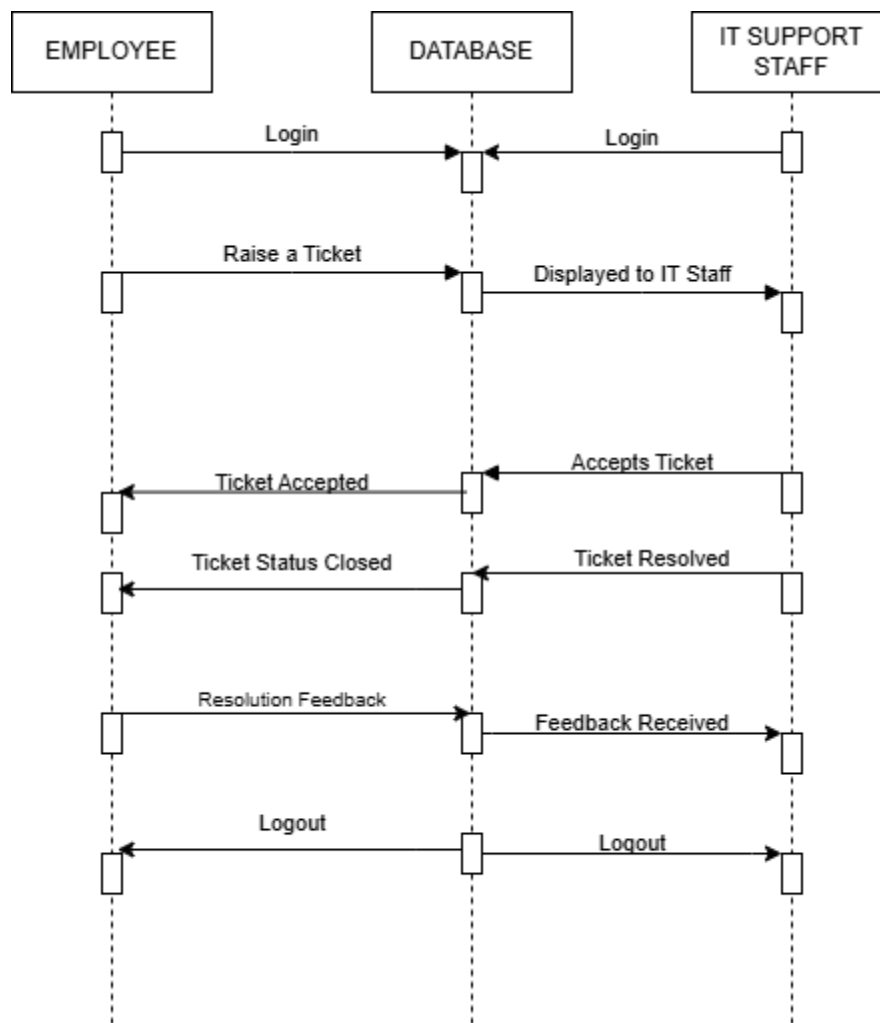


Fig 4.3.5 Sequence Diagram

EXPLANATION:

This sequence diagram illustrates the interaction flow between the Employee, System, and IT Staff in the IT Ticketing System. The process begins when a employee logs in and raises a new ticket through the system interface. The system validates and records the ticket, generating a confirmation for the employee. It then forwards the ticket details to the IT Staff for review. The IT Staff updates the ticket status as they work on resolving the issue. Finally, once the issue is resolved, the system notifies the employee, completing the ticket lifecycle.

4.3.6 COLLABORATION DIAGRAM

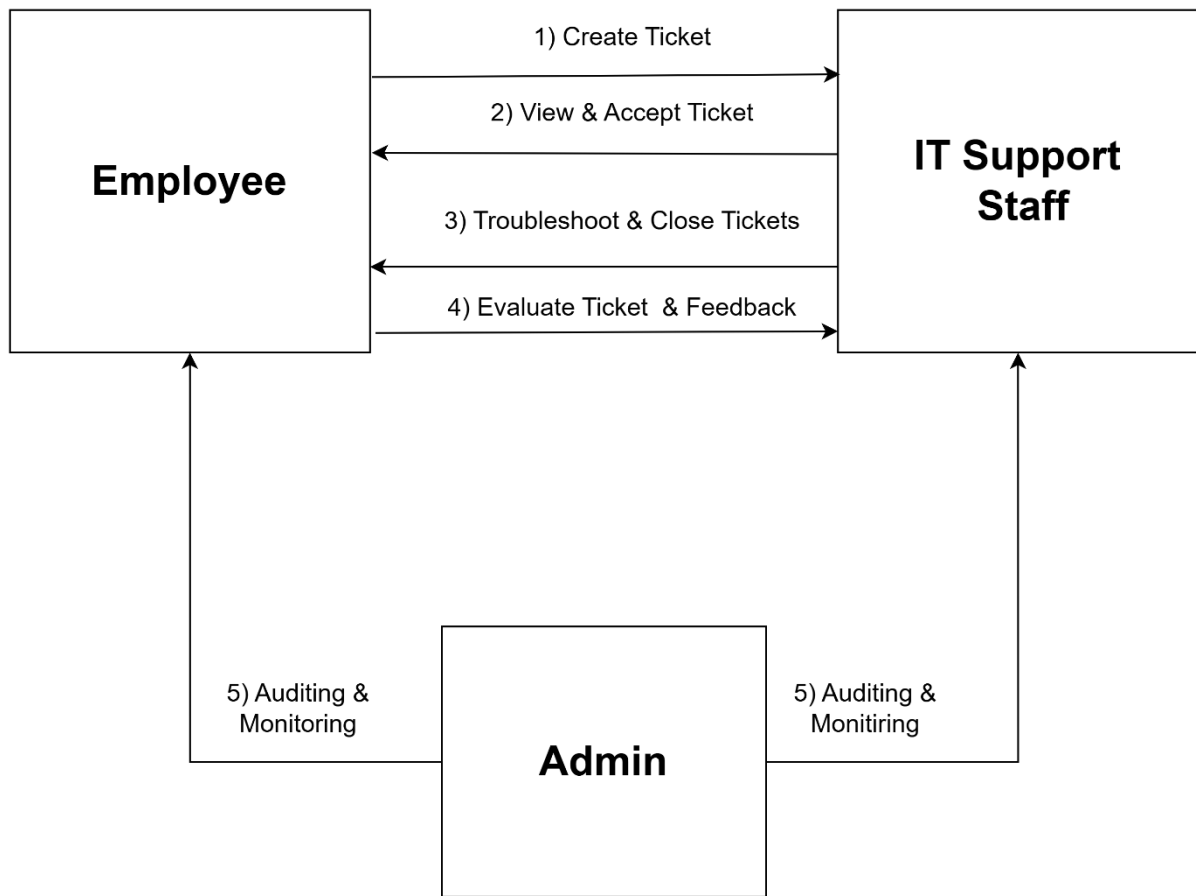


Fig 4.3.6 Collaboration Diagram

EXPLANATION:

This collaboration diagram illustrates the interaction between key roles in the IT ticketing system of the project. The Employee initiates the process by creating a ticket and has the ability to change ticket details and view the status of their tickets. Once a ticket is created, it is directed to the IT Support Staff, who update and resolve the ticket as part of the resolution process. Both the Employee and IT Support Staff collaborate with the Admin for backend operations. The Admin is responsible for managing identity and access, as well as database-related activities. This structured collaboration ensures efficient ticket handling and smooth system functioning.

4.3.7 STATE DIAGRAM

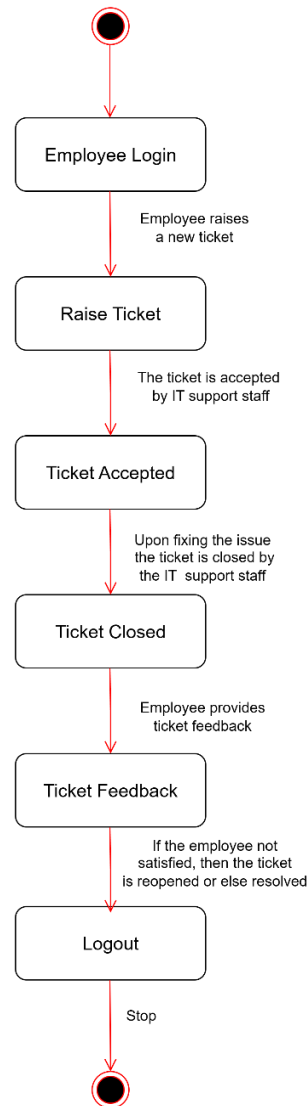


Fig 4.3.7 State Diagram

EXPLANATION:

The State Diagram illustrates the lifecycle of a ticket in the IT ticketing system, starting from Employee Authentication. Once created/Registered, the ticket is raised to IT support staff and moves into the In Progress state after being accepted by an IT Staff member where the issue is actively being worked on. After resolution, the ticket enters the Closed state and waits for confirmation from the employee. Upon employee confirmation, it transitions to the Resolved state, marking the end of the process. If the issue persists, the ticket can be reopened and re-enters the In Progress state upon being accepted by the IT Staff member for further action.

4.3.8 ACTIVITY DIAGRAM

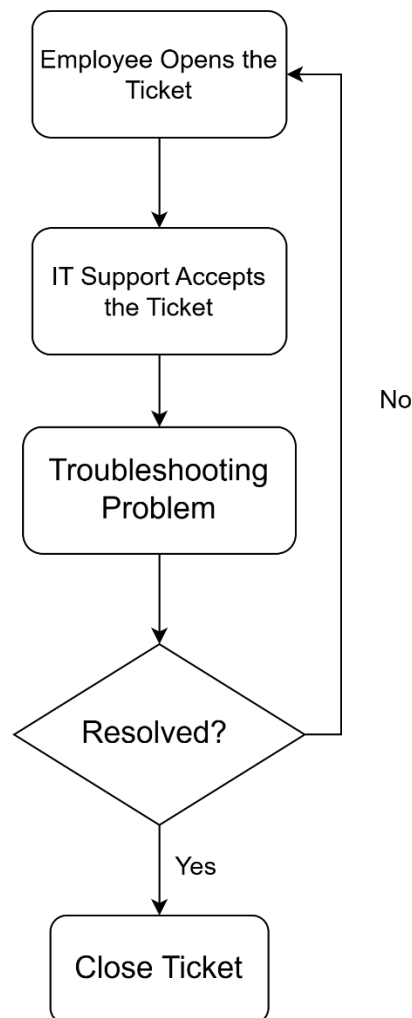


Fig 4.3.8 Activity Diagram

EXPLANATION:

The Activity Diagram shows the workflow of the ticketing process, including decision points and parallel processes. It starts from employee login, moves through ticket raising, IT Staff acceptance towards the final ticket resolution. This diagram helps in understanding the overall process and how different activities are connected.

4.3.9 DEPLOYMENT DIAGRAM

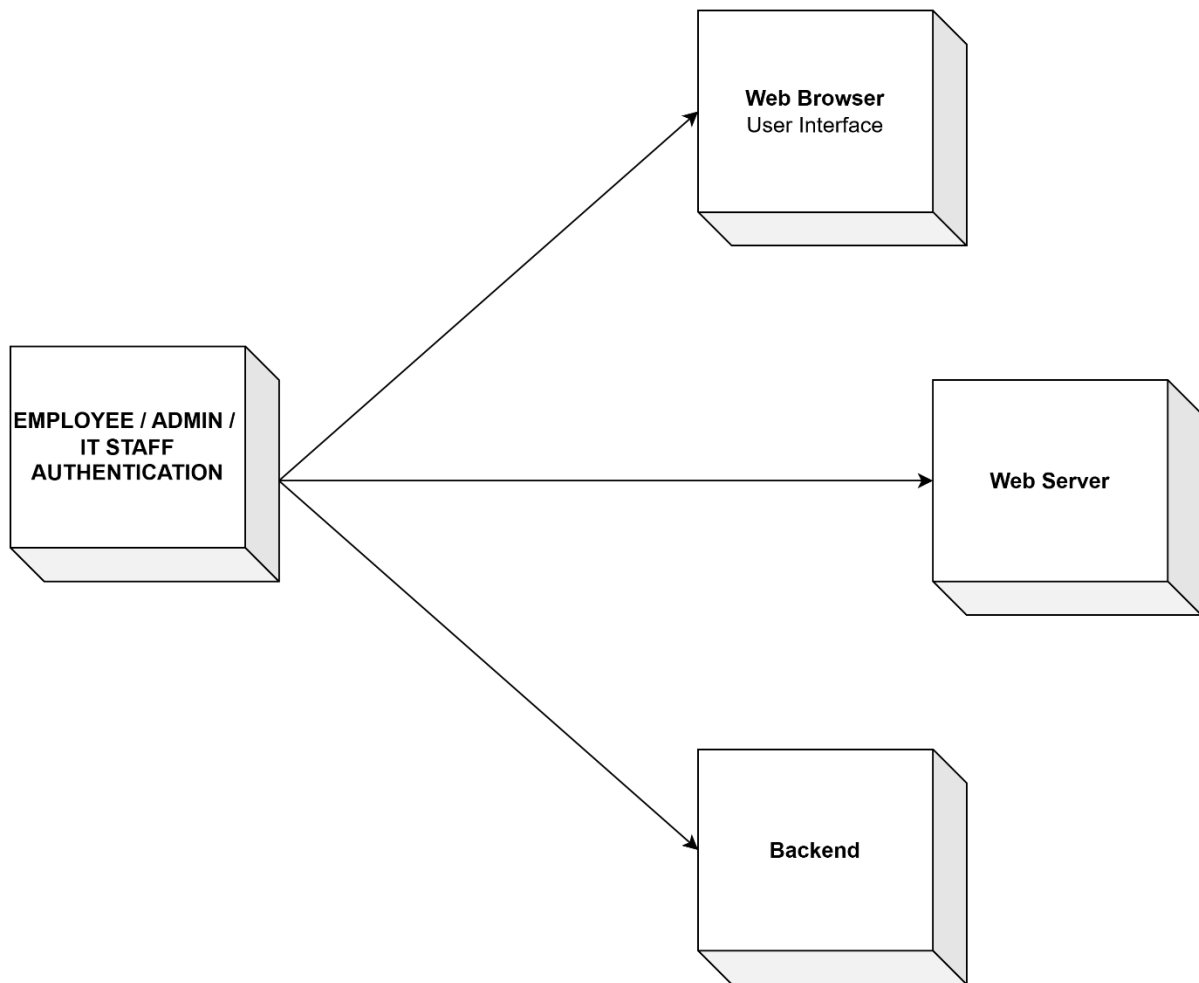


Fig 4.3.9 Deployment Diagram

EXPLANATION:

The Deployment Diagram illustrates the physical deployment of software on hardware. It includes nodes such as a client machine, application server, and database server. Each node hosts components like front-end interfaces, backend logic, and the ticket database. This diagram shows how the system runs on real-world infrastructure. In this particular case, we can see the Employee/Admin/IT Staff Authentication is linked to the backend, web-server and the web browser (User Interface).

4.4 DATA FLOW DIAGRAM

Level - 0

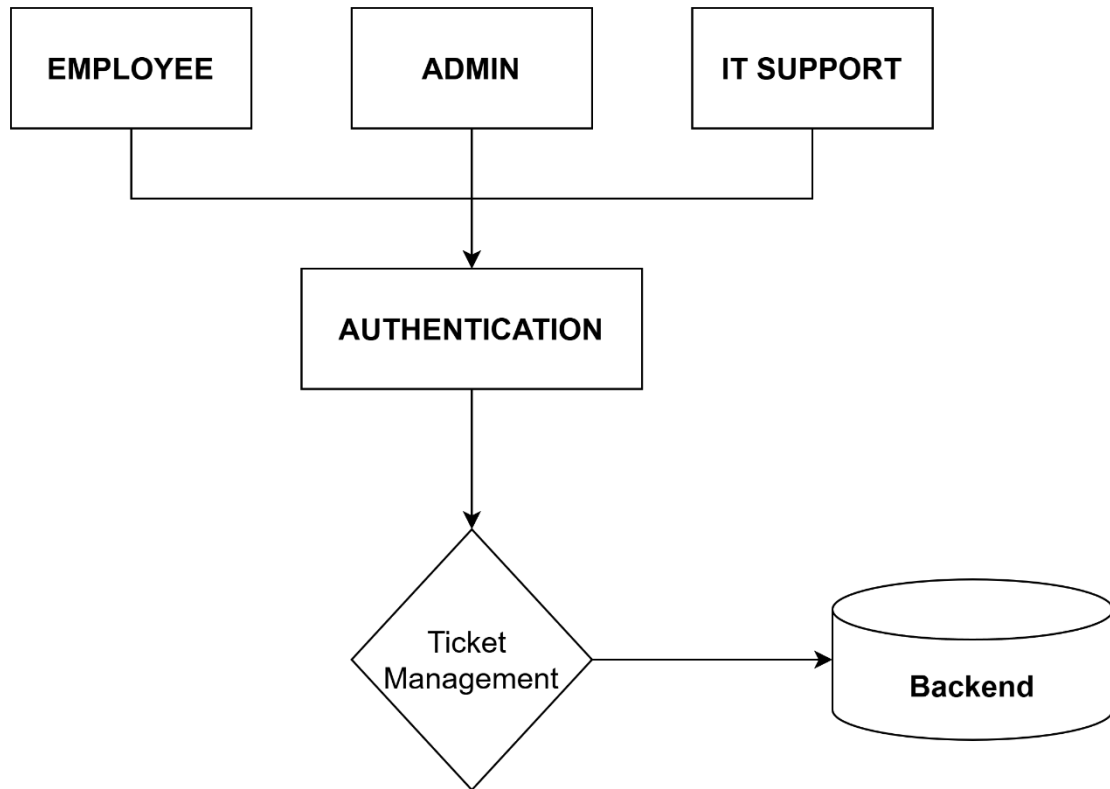


Fig 4.4 Data Flow Diagram (Level 0)

EXPLANATION:

The DFD illustrates how data moves through the system. The Level 0 DFD shows high-level processes such as user interactions, ticket processing, and system responses. The Level 1 DFD breaks these into detailed sub-processes like ticket raising & updating. It includes data stores like the User Database and Ticket Database, and external entities like Employee and Admin. This diagram helps visualize system data flow and interaction points.

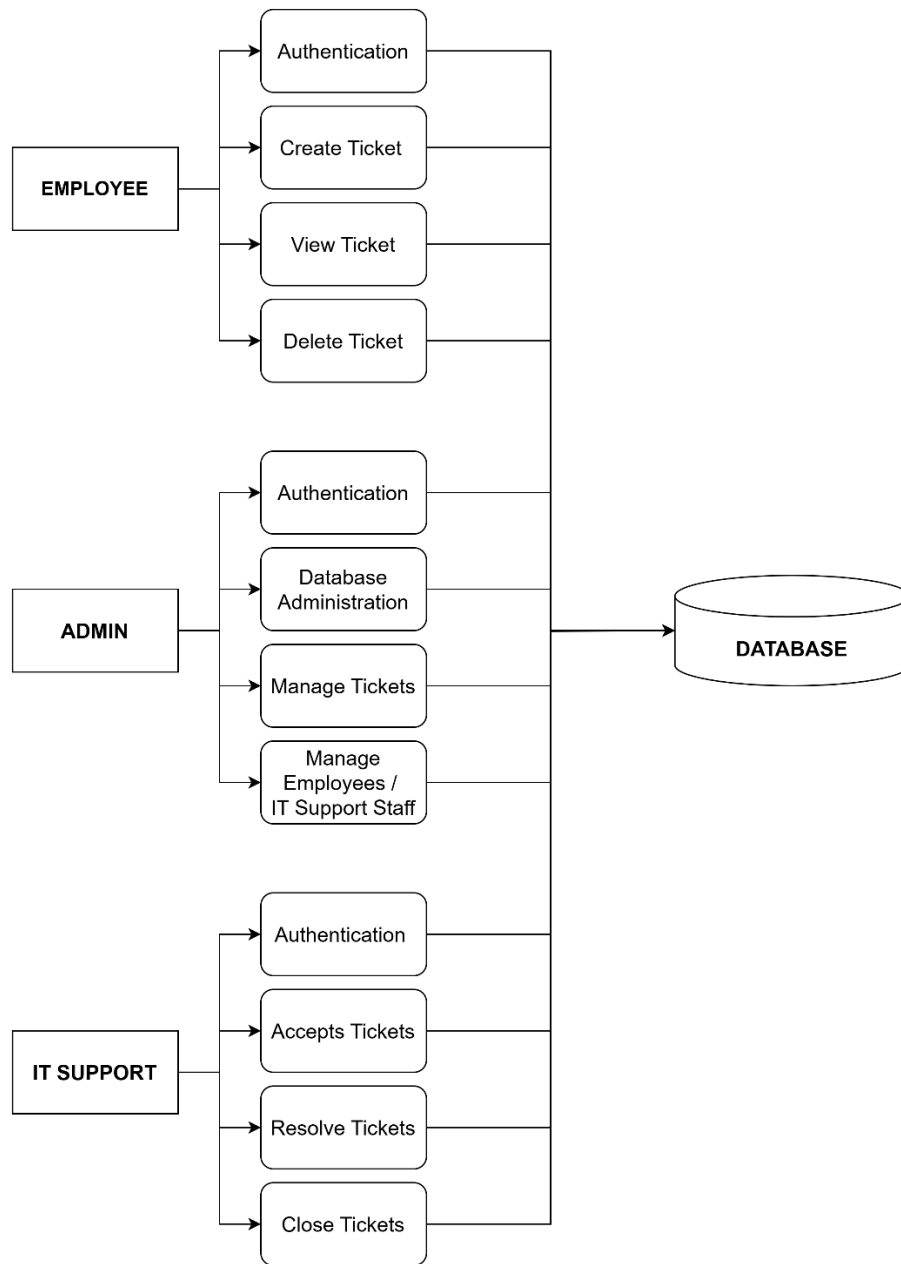
Level - 1

Fig 4.4 Data Flow Diagram (Level 1)

EXPLANATION:

The Level 1 DFD breaks these into detailed sub-processes like ticket raising of employee, accept ticket of IT Support Staff and Database Administration of Admin. This goes a level deeper compared to the previous Level 0 DFD.

4.5 ER DIAGRAM

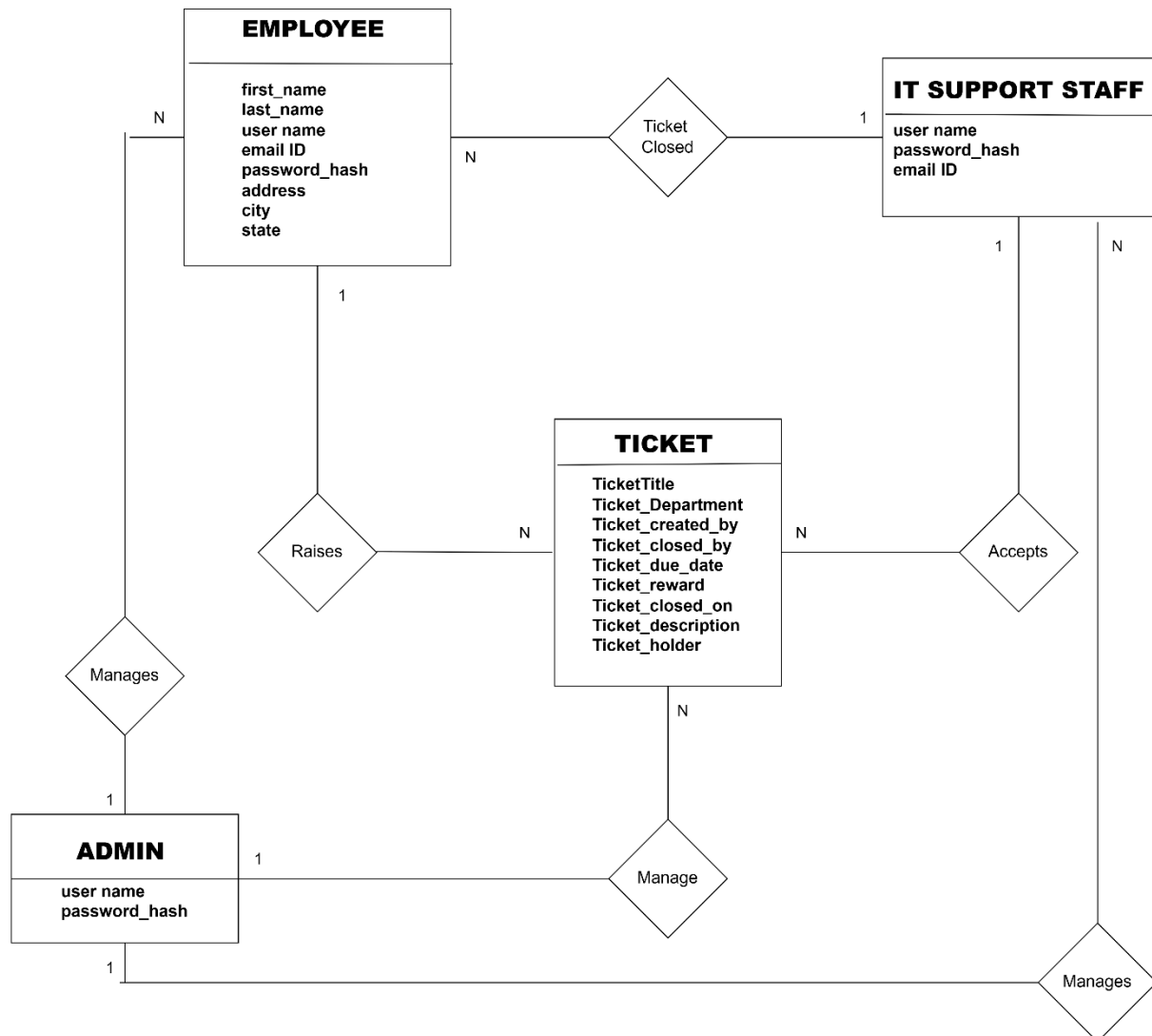


Fig 4.5 ER Diagram

EXPLANATION:

The Entity-Relationship Diagram (ERD) shows the database structure with entities such as Employee, Ticket, Admin, and IT Support Staff, and relationships among them. For example, an employee can raise many Tickets, and an IT Support Staff can manage multiple Tickets. Each entity includes attributes like userID, ticketID, status, etc. The ERD helps design the relational database for the IT Ticketing System.

4.6 GUI DESIGN

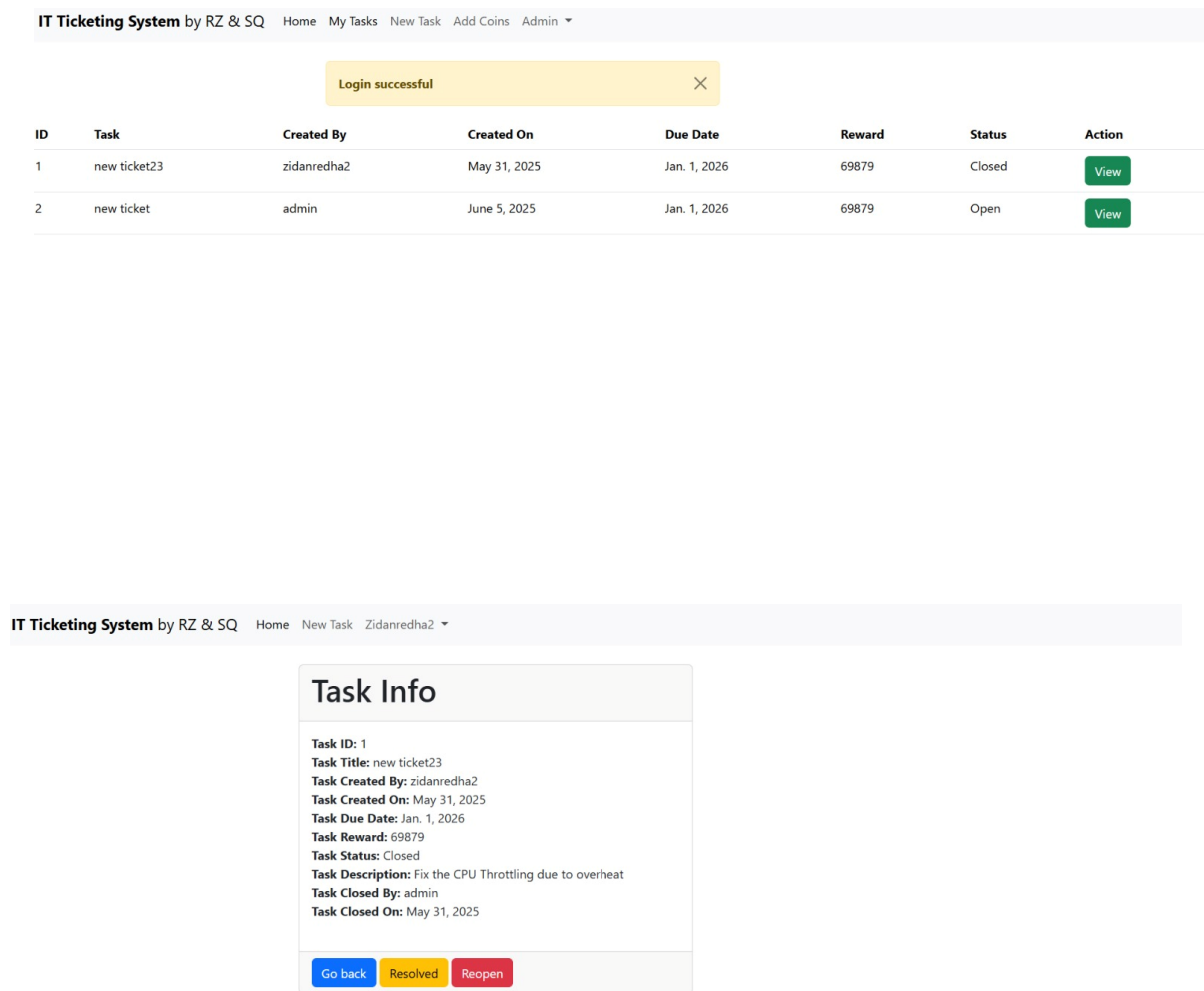


Fig 4.6 GUI Design

Explanation:

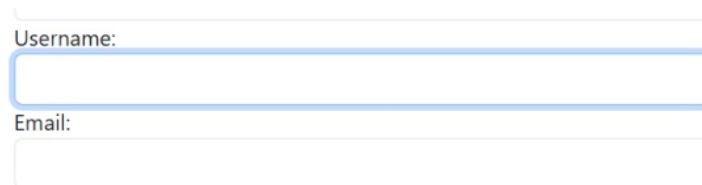
The GUI Design represents the front-end interface for different users. Key screens include the Login Page, Dashboard for Employees and Admins, Ticket Raising Form, Ticket History, and Ticket Management Panel. The interface is designed to be user-friendly, responsive, and intuitive. Buttons like “My Tickets” and “Raise Ticket” provide smooth navigation. Colors and layouts ensure clarity and ease of use.

4.6.1 COMPONENTS OF GUI

Login Page

- **Username / Email Field:**

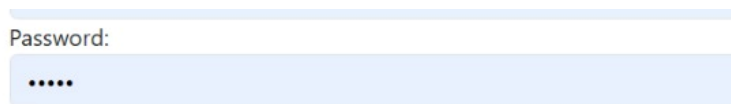
This input field allows users to enter their registered username or email address to log into the system. It is the first step in the authentication process and is essential for identifying the user.



A screenshot of a login form showing two input fields. The first field is labeled 'Username:' and is highlighted with a blue border. The second field is labeled 'Email:' and is empty.

- **Password Field:**

A secured input field that hides the typed characters. Users must enter their correct password corresponding to the email/username. It ensures secure access to the system.



A screenshot of a password input field. The field is labeled 'Password:' and contains six dots, indicating that the password is masked.

- **Login Button:**

When clicked, this button initiates the authentication process. The system checks the entered credentials against the database and grants or denies access based on their validity.



A screenshot of a 'Login' button. The button is dark gray with the word 'Login' in white text.

- **Sign Up / Register Option:**

If enabled, this feature lets new users create an account in the system by entering personal and role-specific details. It helps organizations onboard users without admin intervention.

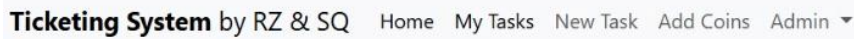


A screenshot of a footer bar. It contains the text 'IT Ticketing System by RZ & SQ' followed by two links: 'Login' and 'Register'.

Dashboard

- **Navigation Sidebar (Home, My Tasks, New Task, User):**

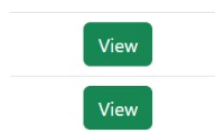
The sidebar provides quick access to the main sections of the application. It remains consistent across pages, helping users easily move between functionalities like viewing own tasks, accepted tasks, task details, etc.



Ticketing System by RZ & SQ Home My Tasks New Task Add Coins Admin ▾

- **Ticket View Button:**

Located prominently on the dashboard, this button redirects users to the ticket list page, where they can view the tickets, they've submitted or are assigned to, depending on their role.



Raise Ticket Form

- **Ticket Title:**

A short and precise text field where the user inputs a summary of the issue. This helps categorize and identify the ticket quickly.

Task Title:

- **Description Text Area:**

A larger field where employees can explain the issue in detail. It may include information such as error messages, affected systems, or steps already taken to resolve the issue. The more detailed this input, the easier it is for the IT team to understand the problem.

Task Description:

- **Submit Button:**

Once all required fields are filled, clicking this button sends the ticket data to the server. The system saves the ticket, assigns it a unique ID, and may notify the assigned technician or admin.



Ticket List View

- **Ticket Table:**

A structured table layout that displays all tickets raised by the employees or visible to the admin/staff, depending on their access rights. It helps with quick ticket monitoring.



Task Info

Task ID: 1
 Task Title: new ticket23
 Task Created By: zidanredha2
 Task Created On: May 31, 2025
 Task Due Date: Jan. 1, 2026
 Task Reward: 69879
 Task Status: Closed
 Task Description: Fix the CPU Throttling due to overheat
 Task Closed By: admin
 Task Closed On: May 31, 2025

Go back Resolved Reopen

- **Ticket ID:**

A system-generated unique identifier for each ticket. It is used for referencing and tracking the ticket throughout its lifecycle.

Task ID: 1

- **Title:**

Shows the short issue summary provided during ticket creation.

Task Title: new ticket23

- **Status (Open, In Progress, Closed):**

Reflects the current state of the ticket. "Open" indicates a new issue, "In Progress" means it's being worked on, and "Closed" shows the issue has been resolved.

Task Status: Closed

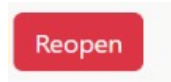
- **Date Created:**

The date and time when the ticket was raised. It helps track response and resolution times.

Task Created On: May 31, 2025

- **View / Edit Button:**

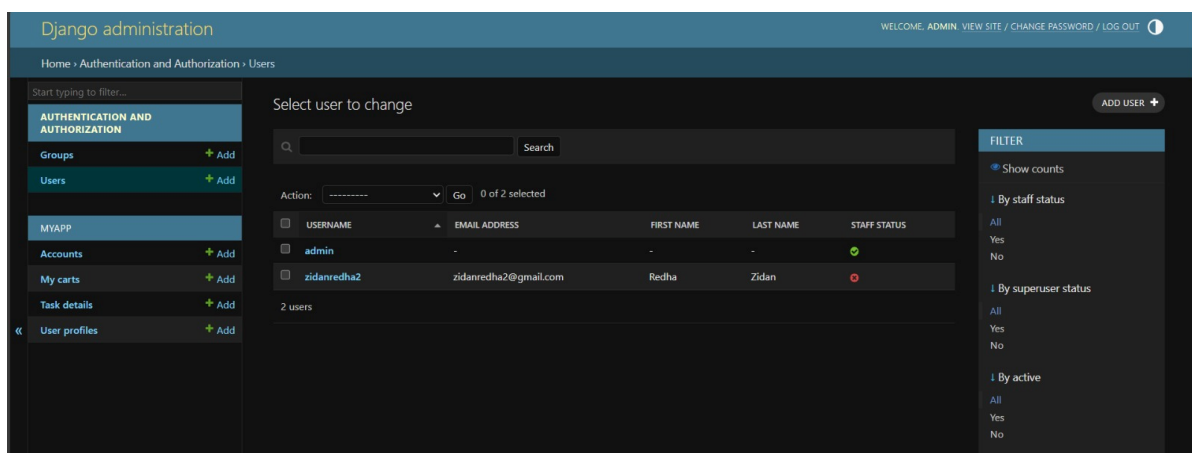
Allows users to open detailed views of the ticket. Admins or authorized users may also edit ticket information, update descriptions, or change the status.



Admin Panel

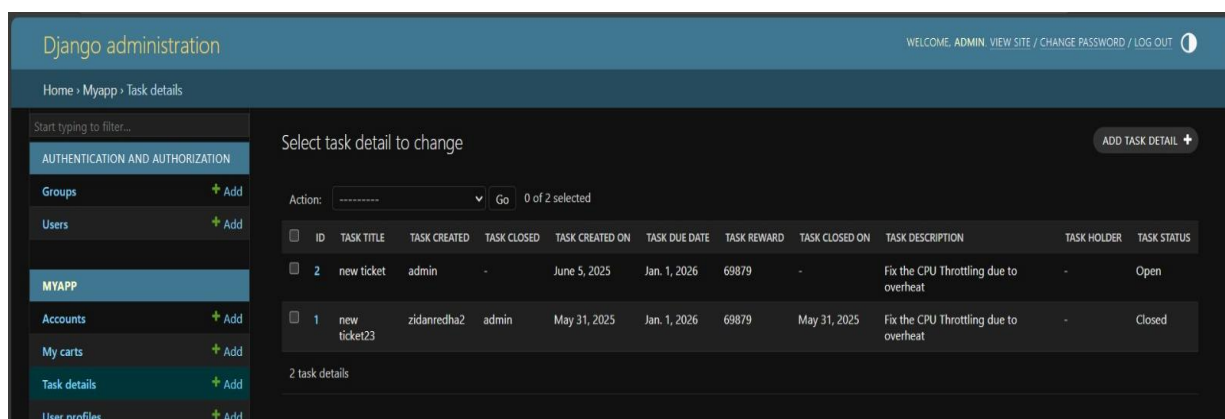
- **User Management:**

Allows administrators to add new users, assign or modify roles (e.g., User, IT Support Staff, Admin), reset passwords, or deactivate accounts. This feature ensures secure and organized user control within the system.



- **Ticket Management:**

Enables users to create, update, and monitor support tickets efficiently. IT staff can assign, prioritize, and resolve issues while tracking progress through status updates.



4.6.2 FEATURES OF GUI

1. User-Friendly Interface

The system is designed with a clean and intuitive user interface that ensures ease of use for all types of users, including employees, IT staff, and administrators. The layout is organized with a sidebar or menu that enables quick access to major sections such as Dashboard, Raise Ticket, and View Tickets. Additionally, the responsive design ensures that the application functions smoothly across various devices including desktops, tablets, and mobile phones, making it accessible anytime and anywhere.

2. Login & Authentication

A secure login mechanism is implemented to protect user accounts and system data. The login form includes input validation to ensure proper data format, while role-based access controls determine the user's permissions upon login—be it Admin, IT Support Staff or Employee. The system also includes a "Forgot Password" feature that allows users to reset their credentials securely, ensuring account recovery without compromising security.

3. Dashboard Overview

The dashboard offers a real-time visual summary of the overall ticketing status, displaying metrics such as the number of tickets that are Open, In Progress, and Closed. It also features shortcut links that allow users to quickly raise new tickets or access existing ones. Additionally, a recent activity log is displayed, showing the latest ticket actions, assignments, and comments—providing users with immediate insight into ongoing activities.

4. Ticket Management

Employees can raise detailed support tickets by entering a title and description of the issue. Once created, tickets can be updated to reflect new information or changes in status. IT staff have the ability to edit tickets assigned to them, add progress notes, or escalate issues as needed. All tickets are stored and organized for easy tracking and reference, forming the core workflow of the system.

5. Real-Time Status Tracking

Each ticket's status is updated in real-time as it progresses through different stages such as Open, In Progress, and Closed. Employees can log in at any time to check the current status of their tickets without needing to contact IT staff. This enhances transparency, improves communication, and reduces the need for follow-up emails or messages.

6. Comments and Updates

The system includes a comment feature within each ticket, allowing employees and IT staff to communicate directly within the platform. Comments are time-stamped and tagged with the name of the commenter, ensuring a clear history of all discussions and updates related to the issue. This functionality helps document the resolution process and supports better collaboration between employees and IT Support teams.

7. Admin Controls

The Admin Panel provides full control over system settings and user management. Admins can create, edit, or deactivate user accounts, assign roles such as Admin, IT Support, or Employee, and manage tickets to keep the system organized. This centralized control panel helps maintain security, structure, and accountability throughout the system.

CHAPTER 5

IMPLEMENTATION

5.1 GENERAL

The implementation phase is the process of translating the design of the IT Ticketing System into a working software solution. After thorough analysis and design using UML diagrams, the system was developed using appropriate front-end and back-end technologies. This phase also involves the integration of modules, database setup, and coding of core functionalities like ticket creation, management, and user access. Proper testing and debugging were performed to ensure that the application meets its functional and non-functional requirements.

5.2 IMPLEMENTATION

The IT Ticketing System is implemented as a web-based application using front-end technologies like HTML and Bootstrap and a back-end using Django with an SQLite database.

HTML

HTML (HyperText Markup Language) is used as the fundamental building block for designing the structure and layout of the web pages in the IT Ticketing System. All essential components such as the ticket raising form, employee dashboard, admin interface, and IT support staff panels were structured using HTML tags. It provided the backbone for placing elements like input fields, tables, buttons, and navigation links. By organizing the content meaningfully, HTML enabled seamless integration with Bootstrap for interactive behavior, forming the base for a functional and user-friendly interface.

BOOTSTRAP

Bootstrap is utilized in the project to design a responsive and consistent front-end layout for the IT Ticketing System. Its powerful grid system and pre-designed components such as cards, alerts, buttons, modals, and navigation bars helped streamline the UI development process. Bootstrap ensured that the system was mobile-friendly and adaptable to various screen sizes, enhancing the usability for employees, IT staff, and administrators. By reducing the need to write custom CSS from scratch, Bootstrap significantly accelerated the design process and maintained a professional appearance across all pages.

DJANGO

Django, a high-level Python web framework, served as the back-end technology for developing the IT Ticketing System. It is used to handle all major functionalities including user authentication, ticket creation, status updates, and role-based access control. Django's Model-View-Template (MVT) architecture enabled a clean separation of data, logic, and presentation layers, improving maintainability and scalability. The built-in admin panel provided a powerful interface for managing tickets and users, while Django's ORM (Object Relational Mapper) simplified database interactions. Overall, Django ensured the system was secure, efficient, and robust.

CODE

Views.py – Main logic

```
from django.shortcuts import render, redirect
from django.utils import timezone
from .forms import LoginForm, RegisterForm, UserProfileForm, TaskDetailForm,
AccountForm
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login, logout
from django.contrib import messages
from django.contrib.auth.forms import PasswordChangeForm
from django.contrib.auth import update_session_auth_hash
from .models import UserProfile, TaskDetail, MyCart, Account
# Create your views here.
def Basepage(request):
    if request.user.is_authenticated:
        Taskdatas= TaskDetail.objects.all()
        return render(request, 'Base.html', {'Taskdatas': Taskdatas})
    else:
        messages.success(request, 'Please login to view your profile')
        return redirect('login')
    return render(request, 'Base.html')
#LoginView
def LoginView(request):
```

```

if request.method == 'POST':
    form=LoginForm(request.POST)
    if form.is_valid():
        username = form.cleaned_data['username']
        password = form.cleaned_data['password']
        user= authenticate(request, username=username, password=password)
        if user is not None:
            login(request,user)
            messages.success(request, 'Login successful')
            return redirect('base')
        else:
            messages.error(request, 'Invalid username or password')
            return render(request, 'Login.html', {'form': form})
    else:
        form = LoginForm()
        return render(request, 'Login.html', {'form': form})

#LogoutView
def LogoutView(request):
    logout(request)
    messages.success(request, 'Logout successful')
    return redirect('login')

#RegisterView
def RegisterView(request):
    if request.method == 'POST':
        Register_form = RegisterForm(request.POST)
        UserProfile_form = UserProfileForm(request.POST)
        if Register_form.is_valid() and UserProfile_form.is_valid():
            Registerform= Register_form.save()
            UserProfileform= UserProfile_form.save(commit=False)
            Address = UserProfile_form.cleaned_data['Address']
            City = UserProfile_form.cleaned_data['City']
            State = UserProfile_form.cleaned_data['State']
            profile = UserProfile(user=Registerform, Address=Address, City=City, State=State)
            profile.save()

```



```

        messages.success(request, 'Registration successful')
        return redirect('login')
    else:
        messages.error(request, 'Registration failed. Please try again.')
        return redirect('register')
    else:
        Register_form = RegisterForm()
        UserProfile_form = UserProfileForm()
        return render(request, 'Register.html', {'Register_form': Register_form,
        'UserProfile_form': UserProfile_form})
#Change_Password
def Change_Password(request):
    if request.method == 'POST':
        fm = PasswordChangeForm(user=request.user, data=request.POST)
        if fm.is_valid():
            fm.save()
            update_session_auth_hash(request, fm.user)
            messages.success(request, 'Password changed successfully')
            return redirect('login')
        else:
            fm = PasswordChangeForm(user=request.user)
            return render(request, 'Change_Password.html', {'fm': fm})

#User_Profile
def User_Profile(request):
    if request.user.is_authenticated:
        user= request.user
        ProfileDatas = UserProfile.objects.filter(user=user)
        AccountDatas = Account.objects.filter(ACCOUNT_HOLDER=user)
        return render(request, 'Userprofile.html', {'AccountDatas': AccountDatas, 'ProfileDatas':
        ProfileDatas})
    else:
        messages.success(request, 'Please login to view your profile')
        return redirect('login')

```

```
# Update_Profile
```

```
def Update_Profile(request, pk):
    if request.user.is_authenticated:
        ProfileDatas= UserProfile.objects.get(id=pk)
        form = UserProfileForm(request.POST or None, instance=ProfileDatas)
        if form.is_valid():
            form.save()
            messages.success(request, 'Profile updated successfully')
            return redirect('profile')
        return render(request, 'Update_Profile.html', {'form': form})
    else:
        messages.success(request, 'Please login to view your profile')
        return redirect('login')
```

```
#Task Creation Function
```

```
def TaskDetails(request):
    if request.user.is_authenticated:
        if request.method=='POST':
            form = TaskDetailForm(request.POST)
            if form.is_valid():
                Task=form.save(commit=False)
                Task.TASK_CREATED = request.user
                Task.save()
                messages.success(request, 'Task created successfully')
                return redirect('base')
            else:
                form = TaskDetailForm()
                return render(request, 'TaskDetail.html', {'form': form})
        else:
            messages.success(request, 'Please login to create a task')
            return redirect('login')
```

```
#TaskInfo
```

```
def TaskInfo(request, pk):
    if request.user.is_authenticated:
```

```

    taskinfos= TaskDetail.objects.get(id=pk)
    return render(request, 'TaskInfo.html', {'taskinfos': taskinfos})
else:
    messages.success(request, 'Please login to view task details')
    return redirect('login')

#UpdateTask
def Update_Task(request, pk):
    if request.user.is_authenticated:
        taskinfos=TaskDetail.objects.get(id=pk)
        form = TaskDetailForm(request.POST or None, instance=taskinfos)
        if form.is_valid():
            form.save()
            messages.success(request, 'Task updated successfully')
            return redirect('base')
        return render(request, 'Update_Task.html', {'form': form})
    else:
        messages.success(request, 'Please login to view tasks.')
        return redirect('login')

#DeleteTask
def Delete_Task(request, pk):
    if request.user.is_authenticated:
        taskinfos= TaskDetail.objects.get(id=pk)
        taskinfos.delete()
        messages.success(request, 'Task deleted successfully')
        return redirect('base')
    else:
        messages.success(request, 'Please login to delete tasks.')
        return redirect('login')

#AcceptTask
def Accept_Task(request, pk):
    if request.user.is_authenticated:
        currentuser = request.user
        taskinfos= TaskDetail.objects.get(id=pk)
        taskinfos.TASK_STATUS = 'In-process'

```

```

        taskinfos.save()
    MyCart(user=currentuser, task=taskinfos).save()
    messages.success(request, 'Task accepted successfully')
    return redirect('mycart')
else:
    messages.success(request, 'Please login to accept tasks.')
    return redirect('login')
#MyCart
def MyCarts(request):
    if request.user.is_authenticated:
        currentuser = request.user
        Carts= MyCart.objects.filter(user=currentuser)
        return render(request, 'MyCart.html', {'Carts': Carts})
    else:
        messages.success(request, 'Please login to View your tasks.')
        return redirect('login')
# Remove Task
def RemoveTask(request, pk):
    if request.user.is_authenticated:
        Taskdatas= TaskDetail.objects.get(id=pk)
        Taskdatas.TASK_STATUS= 'Open'
        Taskdatas.save()
        mycart = MyCart.objects.filter(task=pk)
        mycart.delete()
        messages.success(request, 'Task removed successfully')
        return redirect('mycart')
    else:
        messages.success(request, 'Please login to remove tasks.')
        return redirect('login')
# Closed Task
def CloseTask(request, pk):
    if request.user.is_authenticated:
        Taskdatas= TaskDetail.objects.get(id=pk)
        Taskdatas.TASK_STATUS= 'Closed'

```

```

Taskdatas.TASK_CLOSED = request.user
Taskdatas.TASK_CLOSED_ON = timezone.now()
Taskdatas.save()
mycart = MyCart.objects.filter(task=pk)
mycart.delete()
messages.success(request, 'Task closed successfully')
return redirect('mycart')
else:
    messages.success(request, 'Please login to close tasks.')
    return redirect('login')
#ReopenTask
def Reopen_Task(request, pk):
    if request.user.is_authenticated:
        currentuser = request.user
        taskinfos= TaskDetail.objects.get(id=pk)
        taskinfos.TASK_STATUS = 'Reopened'
        holder = User.objects.get(username=taskinfos.TASK_CLOSED)
        taskinfos.save()
        MyCart(user=holder, task=taskinfos).save()
        messages.success(request, 'Task opened successfully')
        return redirect('base')
    else:
        messages.success(request, 'Please login to open tasks.')
        return redirect('login')
#ResolvedTask
def Resolved_Task(request, pk):
    if request.user.is_authenticated:
        taskinfos= TaskDetail.objects.get(id=pk)
        taskinfos.TASK_STATUS = 'Resolved'
        #holder = User.objects.get(username=taskinfos.TASK_CLOSED)
        taskinfos.save()
        messages.success(request, 'Task resolved successfully')
        return redirect('base')
    else:

```

```

        messages.success(request, 'Please login to resolve tasks.')
        return redirect('login')
#Account Function
def AccountDetails(request):
    if request.user.is_authenticated:
        if request.method=='POST':
            form = AccountForm(request.POST)
            if form.is_valid():
                Task=form.save(commit=False)
                Task.TASK_CREATED = request.user
                Task.save()
                messages.success(request, 'Coins added successfully!')
                return redirect('base')
            else:
                form = AccountForm()
                return render(request, 'Account.html', {'form': form})
        else:
            messages.success(request, 'Please login to add coins.')
            return redirect('login')

```

urls.py – Template URL mapping

```

from django.contrib import admin
from django.urls import path, include
from . import views
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('base/', views.Basepage, name='base'),
    path('login/', views.LoginView, name='login'),
    path('logout/', views.LogoutView, name='logout'),
    path('register/', views.RegisterView, name='register'),
    path('password_reset/',
auth_views.PasswordResetView.as_view(template_name='password_reset_form.html'),
name='password_reset'),

```

```

    path('password_reset/done/', auth_views.PasswordResetDoneView.as_view(template_name
= 'password_reset_done.html'), name='password_reset_done'),
    path('password_reset_confirm/<uidb64>/<token>/',
auth_views.PasswordResetConfirmView.as_view(template_name
= 'password_reset_confirm.html'), name='password_reset_confirm'),
    path('password_reset_complete/',
auth_views.PasswordResetCompleteView.as_view(template_name
= 'password_reset_complete.html'), name='password_reset_complete'),
    path('ChangePassword/', views.Change_Password, name='ChangePassword'),
    path('profile/', views.User_Profile, name='profile'),
    path('update_profile/<int:pk>', views.Update_Profile, name='update_profile'),
    path('taskdetails/', views.TaskDetails, name='taskdetails'),
    path('taskinfo/<int:pk>', views.TaskInfo, name='taskinfo'),
    path('update_task/<int:pk>', views.Update_Task, name='task_update'),
    path('delete_task/<int:pk>', views.Delete_Task, name='task_delete'),
    path('accept_task/<int:pk>', views.Accept_Task, name='task_accept'),
    path('remove_task/<int:pk>', views.RemoveTask, name='task_remove'),
    path('close_task/<int:pk>', views.CloseTask, name='task_close'),
    path('reopen_task/<int:pk>', views.Reopen_Task, name='task_reopen'),
    path('resolved_task/<int:pk>', views.Resolved_Task, name='task_resolved'),
    path('account', views.AccountDetails, name='account'),
    path('mycart', views.MyCarts, name='mycart'),
]

```

Forms.py - User form pages

```

from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
from .models import UserProfile, TaskDetail, Account
#LoginForm
class LoginForm(forms.Form):
    username = forms.CharField(label='Username', max_length=100,
widget=forms.TextInput(attrs={'class':'form-control'}))
    password = forms.CharField(label='Password', max_length=100,
widget=forms.PasswordInput(attrs={'class':'form-control'}))

```

RegisterForm

```
class RegisterForm(UserCreationForm):
```

```
    first_name      =      forms.CharField(label='First      Name',      max_length=100,
widget=forms.TextInput(attrs={'class':'form-control'}))
```

```
    last_name       =      forms.CharField(label='Last      Name',      max_length=100,
widget=forms.TextInput(attrs={'class':'form-control'}))
```

```
    username        =      forms.CharField(label='Username',      max_length=100,
widget=forms.TextInput(attrs={'class':'form-control'}))
```

```
    email           =      forms.EmailField(label='Email',      max_length=100,
widget=forms.EmailInput(attrs={'class':'form-control'}))
```

```
    password1       =      forms.CharField(label='Password',      max_length=100,
widget=forms.PasswordInput(attrs={'class':'form-control'}))
```

```
    password2       =      forms.CharField(label='Confirm      Password',      max_length=100,
widget=forms.PasswordInput(attrs={'class':'form-control'}))
```

```
    class Meta:
```

```
        model = User
```

```
        fields = ('first_name','last_name','username', 'email', 'password1', 'password2')
```

UserProfileForm

```
class UserProfileForm(forms.ModelForm):
```

```
    Address         =      forms.CharField(label='Address',      max_length=100,
widget=forms.TextInput(attrs={'class':'form-control'}))
```

```
    City            =      forms.CharField(label='City',      max_length=100,
widget=forms.TextInput(attrs={'class':'form-control'}))
```

```
    State          =      forms.CharField(label='State',      max_length=100,
widget=forms.TextInput(attrs={'class':'form-control'}))
```

```
    class Meta:
```

```
        model = UserProfile
```

```
        fields = ( 'Address','City', 'State')
```

#TaskDetail Form

```
class TaskDetailForm(forms.ModelForm):
```

```
    TASK_TITLE      =      forms.CharField(label='Task      Title',      max_length=100,
widget=forms.TextInput(attrs={'class':'form-control'}))
```



```

TASK_DEPARTMENT = forms.ModelChoiceField(label='Concerned Department',
queryset=Group.objects.all(), widget=forms.Select(attrs={'class':'form-control'}))

TASK_DUE_DATE = forms.DateField(label='Task Due Date',
widget=forms.TextInput(attrs={'class':'form-control', 'placeholder':'YYYY-MM-DD'}))

TASK_REWARD = forms.IntegerField(label='Task Reward')

TASK_DESCRIPTION = forms.CharField(label='Task Description', max_length=300,
widget=forms.TextInput(attrs={'class':'form-control'}))

class Meta:
    model= TaskDetail
    fields = ('TASK_TITLE', 'TASK_DEPARTMENT', 'TASK_DUE_DATE',
'TASK_REWARD', 'TASK_DESCRIPTION',)
# AccountForm
class AccountForm(forms.ModelForm):
    ACCOUNT_HOLDER = forms.ModelChoiceField(label='Holder',
queryset=User.objects.all())
    ACCOUNT_BALANCE = forms.IntegerField(label='Balance',
widget=forms.TextInput(attrs={'class':'form-control'}))
    class Meta:
        model = Account
        fields = ('ACCOUNT_HOLDER', 'ACCOUNT_BALANCE')

```

Models.py – Represents Django Models

```

from django.db import models
from django.forms import forms
from django.contrib.auth.models import User, Group
# Create your models here.
Group.objects.get_or_create(name='IT')
Group.objects.get_or_create(name='HR')
Group.objects.get_or_create(name='Finance')
#UserProfile model
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, null= True)
    Address= models.CharField(max_length=100)

```

```
City = models.CharField(max_length=100)
```

```
State= models.CharField(max_length=100)
```

```
#TaskDetails Model
```

```
class TaskDetail(models.Model):
```

```
    TASK_TITLE = models.CharField(max_length=100)
```

```
    TASK_DEPARTMENT = models.ForeignKey(Group, on_delete=models.CASCADE,
null=True)
```

```
    TASK_CREATED = models.ForeignKey(User, related_name='CREATED_BY',
on_delete=models.CASCADE, null=True)
```

```
    TASK_CLOSED = models.ForeignKey(User, related_name='CLOSED_BY',on_delete=models.CASCADE, null=True)
```

```
    TASK_CREATED_ON = models.DateField(auto_now_add=True,null=True)
```

```
    TASK_DUE_DATE = models.DateField()
```

```
    TASK_REWARD = models.IntegerField()
```

```
    TASK_CLOSED_ON = models.DateField(null=True)
```

```
    TASK_DESCRIPTION = models.CharField(max_length=300)
```

```
    TASK HOLDER = models.CharField(max_length=100, null=True)
```

```
    choice=[('Open', 'Open'),('In-process', 'In-process'), ('Reopened', 'Reopened'), ('Closed',
'Closed'), ('Expired', 'Expired'), ('Resolved', 'Resolved')]
```

```
    TASK_STATUS = models.CharField(max_length=100,choices=choice, default='Open')
```

```
#MyCart Model
```

```
class MyCart(models.Model):
```

```
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True)
```

```
    task= models.ForeignKey(TaskDetail, on_delete=models.CASCADE, null=True)
```

```
    task_count= models.IntegerField(default=1)
```

```
#Account Model
```

```
class Account(models.Model):
```

```
    ACCOUNT HOLDER = models.OneToOneField(User, on_delete=models.CASCADE,
null=True)
```

```
    ACCOUNT_BALANCE= models.IntegerField()
```

Admin.py – Represents Admin Dashboard

```

from django.contrib import admin
from .models import UserProfile, TaskDetail, MyCart, Account
# Register your models here.

#UserProfile Admin
class UserProfileAdmin(admin.ModelAdmin):
    list_display = ['Address', 'City', 'State']
admin.site.register(UserProfile, UserProfileAdmin)

#TaskDetails Admin
class TaskDetailsAdmin(admin.ModelAdmin):
    list_display = ['id', 'TASK_TITLE', 'TASK_DEPARTMENT', 'TASK_CREATED',
'TASK_CLOSED', 'TASK_CREATED_ON', 'TASK_DUE_DATE', 'TASK_REWARD',
'TASK_CLOSED_ON', 'TASK_DESCRIPTION', 'TASK HOLDER', 'TASK_STATUS']
admin.site.register(TaskDetail, TaskDetailsAdmin)

#MyCart Admin
class MyCartAdmin(admin.ModelAdmin):
    list_display = ['id', 'user', 'task', 'task_count']
admin.site.register(MyCart, MyCartAdmin)

#Account Admin
class AccountAdmin(admin.ModelAdmin):
    list_display = ['id', 'ACCOUNT HOLDER', 'ACCOUNT_BALANCE']
admin.site.register(Account, AccountAdmin)

```

Templates – HTML pages:**base.html:**

```

<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>IT Ticketing System</title>
    <link          href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.5/dist/css/bootstrap.min.css"
rel="stylesheet"                                integrity="sha384-

```

```

SgOJa3DmI69IUzQ2PVdRZhWQ+dy64/BUtbMJw1MZ8t5HZApC4W0kG879m7"
crossorigin="anonymous">
</head>
<body>
    {% include "Navbar.html" %}
    <div class="col-4 offset-3">
        {% if messages %}
        <br>
        {% for message in messages %}
        <div class="alert alert-warning alert-dismissible fade show" role="alert">
            <strong>{{ message }}</strong>
            <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
label="Close"></button>
        </div>
        {% endfor %}
        {% endif %}
    </div>
    {% block content %}
    <table class="table table-hover">
        <thead>
            <tr>
                <th scope="col">ID</th>
                <th scope="col">Ticket</th>
                <th scope="col">Department</th>
                <th scope="col">Created By</th>
                <th scope="col">Created On</th>
                <th scope="col">Due Date</th>
                <th scope="col">Reward</th>
                <th scope="col">Status</th>
                <th scope="col">Action</th>
            </tr>
        </thead>

        {% if request.user.is_superuser %}

```

```

<tbody>
  {% for taskdata in Taskdatas %}
    <tr>
      <td>{{taskdata.id}}</td>
      <td>{{taskdata.TASK_TITLE}}</td>
      <td>{{taskdata.TASK_DEPARTMENT}}</td>
      <td>{{taskdata.TASK_CREATED}}</td>
      <td>{{taskdata.TASK_CREATED_ON}}</td>
      <td>{{taskdata.TASK_DUE_DATE}}</td>
      <td>{{taskdata.TASK_REWARD}}</td>
      <td>{{taskdata.TASK_STATUS}}</td>
      <td><a href="{% url 'taskinfo' taskdata.id %}" class="btn btn-success">View</a></td>
    </tr>
  {% endfor %}
</tbody>

{% elif request.user.is_authenticated %}
<tbody>
  {% for taskdata in Taskdatas %}
    {% if taskdata.TASK_CREATED == request.user %}
      <tr>
        <td>{{taskdata.id}}</td>
        <td>{{taskdata.TASK_TITLE}}</td>
        <td>{{taskdata.TASK_DEPARTMENT}}</td>
        <td>{{taskdata.TASK_CREATED}}</td>
        <td>{{taskdata.TASK_CREATED_ON}}</td>
        <td>{{taskdata.TASK_DUE_DATE}}</td>
        <td>{{taskdata.TASK_REWARD}}</td>
        <td>{{taskdata.TASK_STATUS}}</td>
        <td><a href="{% url 'taskinfo' taskdata.id %}" class="btn btn-success">View</a></td>
      </tr>
    {% endif %}
  {% endfor %}
</tbody>
{% endif %}

```

```

</table>

{%endblock content%}

<script    src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.5/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
k6d4wzSlapyDyv1kpU366/PK5hCdSbCRGRCMv+epIQJWyd1fbcAu9OCUj5zNLiq"
crossorigin="anonymous"></script>

</body>
</html>

```

Account.html:

```

{%extends 'Base.html' %}

{%block content%}

<div class="col-4 offset-3">

<h1>Add Coins</h1>

<form action="" method="post">

    {% csrf_token %}

    {{form.as_p}}

    <br>

    <button type="submit" class="btn btn-secondary">Submit</button>

</form>

</div>

{%endblock content%}

```

Change_password.html:

```

{% extends 'base.html' %}

{% block content %}

<h3>Change Password</h3>

<form method='post'>

    {% csrf_token %}

    {{ fm.as_p }}

    <button type="submit">Submit</button>

</form>

{% endblock %}

```

Login.html:

```
{%extends 'Base.html' %}
{%block content%}
<div class="col-4 offset-3">
<h1>Login</h1>
<form action="" method="post">
    {% csrf_token %}
    {{form}}
    <!-- <a href="{% url 'password_reset' %}">Forgot password?</a> -->
    <br>
    <button type="submit" class="btn btn-secondary">Login</button>
</form>
</div>
{%endblock content%}
```

Mycart.html:

```
{% extends 'Base.html' %}
{% block content %}
<div class="col-4 offset-3">
    <br>
    <h1 class="text-center">My Tickets</h1>
    <hr>
    {% if Carts %}
    {% for cart in Carts %}
    <div class="card text">
        <div class="card-body">
            <strong>Ticket ID: </strong>{{cart.id}}<br>
            <strong>Ticket Title: </strong>{{cart.task.TASK_TITLE}}<br>
            <strong>Ticket Title: </strong>{{cart.task.TASK_DEPARTMENT}}<br>
            <strong>Ticket Created By: </strong>{{cart.task.TASK_CREATED}}<br>
            <strong>Ticket Created On: </strong>{{cart.task.TASK_CREATED_ON}}<br>
            <strong>Ticket Due Date: </strong>{{cart.task.TASK_DUE_DATE}}<br>
            <strong>Ticket Reward: </strong>{{cart.task.TASK_REWARD}}<br>
            <strong>Ticket Status: </strong>{{cart.task.TASK_STATUS}}<br>
```

```

        <strong>Ticket Assigned To: </strong>{{request.user}}<br>
        <strong>Ticket Description: </strong>{{cart.task.TASK_DESCRIPTION}}<br>
    <br>
</div>

<div class="card-footer text-body-secondary">
    <a href="{% url 'base' %}" class="btn btn-primary">Go back</a>
    <a href="{% url 'task_close' cart.task.id %}" class="btn btn-warning">Close
Ticket</a>
    <a href="{% url 'task_remove' cart.task.id %}" class="btn btn-danger">Remove
Ticket</a>
</div>
</div>
<br>
{% endfor %}
{% else %}
<div class="alert alert-warning alert-dismissible fade show" role="alert">
    <strong>No tasks assigned to you!</strong>
    <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
label="Close"></button>
</div>
{% endif %}
</div>
{% endblock content %}

```

Navbar.html:

```

<nav class="navbar navbar-expand-lg bg-body-tertiary">
    <div class="container-fluid">
        <a class="navbar-brand" href="{% url 'base' %}"><b>IT Ticketing System</b> by RZ &
SQ</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-
expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
    </div>
</nav>

```



```

<div class="collapse navbar-collapse" id="navbarSupportedContent">
  <ul class="navbar-nav me-auto mb-2 mb-lg-0">
    {% if request.user.is_authenticated and request.user.is_superuser %}
    <li class="nav-item">
      <a class="nav-link active" aria-current="page" href="{% url 'base' %}">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link active" aria-current="page" href="{% url 'mycart' %}">My
Tickets</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="{% url 'taskdetails' %}">Raise Ticket</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="{% url 'account' %}">Add Coins</a>
    </li>
    <li class="nav-item dropdown">
      <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-
toggle="dropdown" aria-expanded="false">
        {{ request.user|capfirst }}
      </a>
      <ul class="dropdown-menu">
        <li><a class="dropdown-item" href="{% url 'profile' %}">Show Profile</a></li>
        <li><a class="dropdown-item" href="{% url 'ChangePassword' %}">Change
Password</a></li>
        <li><a class="dropdown-item" href="{% url 'logout' %}">Logout</a></li>
      </ul>
    </li>
    {% elif request.user.is_authenticated %}
    <li class="nav-item">
      <a class="nav-link active" aria-current="page" href="{% url 'base' %}">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="{% url 'taskdetails' %}">Raise Ticket</a>
    </li>
  </ul>

```

```

<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-
toggle="dropdown" aria-expanded="false">
        {{ request.user|capfirst }}
    </a>
    <ul class="dropdown-menu">
        <li><a class="dropdown-item" href="{% url 'profile' %}">Show Profile</a></li>
        <li><a class="dropdown-item" href="{% url 'ChangePassword' %}">Change
Password</a></li>
        <li><a class="dropdown-item" href="{% url 'logout' %}">Logout</a></li>
    </ul>
    {% else %}
    <li class="nav-item">
        <a class="nav-link" href="{% url 'login' %}">Login</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="{% url 'register' %}">Register</a>
    </li>
    </li>
    {% endif %}
</ul>
</div>
</div>
</nav>

```

Register.html:

```

{%extends 'Base.html' %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register</title>
</head>

```

```

<body>
    {%block content%}
    <div class="col-4 offset-3">
        <h1>Register</h1>
        <form action="" method="post">
            {% csrf_token %}
            {{Register_form}}
            {{UserProfile_form}}
            <br>
            <button type="submit" class="btn btn-secondary">Register</button>
        </form>
    </div>
    {%endblock content%}
</body>
</html>

```

TaskDetail.html:

```

{%extends 'Base.html' %}
{%block content%}
<div class="col-4 offset-3">
<h1>Raise Ticket</h1>
<form action="" method="post">
    {% csrf_token %}
    {{form.as_p}}
    <br>
    <button type="submit" class="btn btn-secondary">Raise</button>
</form>
</div>
{%endblock content%}

```

Taskinfo.html:

```

{%extends 'Base.html' %}
<head>
    <meta charset="UTF-8">

```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Ticket Info</title>
</head>
<body>
  {%block content%}
  <div class="col-4 offset-3">
    <br>
    <div class="card">
      <div class="card-header">
        <h1>Ticket Info</h1>
      </div>
      <div class="card-body">
        {% if taskinfos.TASK_STATUS == 'Open' %}
        <strong>Ticket ID: </strong>{{taskinfos.id}}<br>
        <strong>Ticket Title: </strong>{{taskinfos.TASK_TITLE}}<br>
        <strong>Concerned Department: </strong>{{taskinfos.TASK_DEPARTMENT}}<br>
        <strong>Ticket Created By: </strong>{{taskinfos.TASK_CREATED}}<br>
        <strong>Ticket Created On: </strong>{{taskinfos.TASK_CREATED_ON}}<br>
        <strong>Ticket Due Date: </strong>{{taskinfos.TASK_DUE_DATE}}<br>
        <strong>Ticket Reward: </strong>{{taskinfos.TASK_REWARD}}<br>
        <strong>Ticket Status: </strong>{{taskinfos.TASK_STATUS}}<br>
        <strong>Ticket Description: </strong>{{taskinfos.TASK_DESCRIPTION}}<br>
        <br>
        {% elif taskinfos.TASK_STATUS == 'Closed' %}
        <strong>Ticket ID: </strong>{{taskinfos.id}}<br>
        <strong>Ticket Title: </strong>{{taskinfos.TASK_TITLE}}<br>
        <strong>Concerned Department: </strong>{{taskinfos.TASK_DEPARTMENT}}<br>
        <strong>Ticket Created By: </strong>{{taskinfos.TASK_CREATED}}<br>
        <strong>Ticket Created On: </strong>{{taskinfos.TASK_CREATED_ON}}<br>
        <strong>Ticket Due Date: </strong>{{taskinfos.TASK_DUE_DATE}}<br>
        <strong>Ticket Reward: </strong>{{taskinfos.TASK_REWARD}}<br>
        <strong>Ticket Status: </strong>{{taskinfos.TASK_STATUS}}<br>
        <strong>Ticket Description: </strong>{{taskinfos.TASK_DESCRIPTION}}<br>
        <strong>Ticket Closed By: </strong>{{taskinfos.TASK_CLOSED}}<br>

```

```

<strong> Ticket Closed On: </strong>{{taskinfos.TASK_CLOSED_ON}}<br>
<br>
{% elif taskinfos.TASK_STATUS == 'In-process' %}
<strong> Ticket ID: </strong>{{taskinfos.id}}<br>
<strong> Ticket Title: </strong>{{taskinfos.TASK_TITLE}}<br>
<strong>Concerned Department: </strong>{{taskinfos.TASK_DEPARTMENT}}<br>
<strong> Ticket Created By: </strong>{{taskinfos.TASK_CREATED}}<br>
<strong> Ticket Created On: </strong>{{taskinfos.TASK_CREATED_ON}}<br>
<strong> Ticket Due Date: </strong>{{taskinfos.TASK_DUE_DATE}}<br>
<strong> Ticket Reward: </strong>{{taskinfos.TASK_REWARD}}<br>
<strong> Ticket Status: </strong>{{taskinfos.TASK_STATUS}}<br>
<strong> Ticket Description: </strong>{{taskinfos.TASK_DESCRIPTION}}<br>
<strong> Ticket Holder: </strong>{{taskinfos.TASK_HOLDER}}<br>
<br>
{% elif taskinfos.TASK_STATUS == 'Reopened' %}
<strong> Ticket ID: </strong>{{taskinfos.id}}<br>
<strong> Ticket Title: </strong>{{taskinfos.TASK_TITLE}}<br>
<strong>Concerned Department: </strong>{{taskinfos.TASK_DEPARTMENT}}<br>
<strong> Ticket Created By: </strong>{{taskinfos.TASK_CREATED}}<br>
<strong> Ticket Created On: </strong>{{taskinfos.TASK_CREATED_ON}}<br>
<strong> Ticket Due Date: </strong>{{taskinfos.TASK_DUE_DATE}}<br>
<strong> Ticket Reward: </strong>{{taskinfos.TASK_REWARD}}<br>
<strong> Ticket Status: </strong>{{taskinfos.TASK_STATUS}}<br>
<strong> Ticket Description: </strong>{{taskinfos.TASK_DESCRIPTION}}<br>
<br>
{% elif taskinfos.TASK_STATUS == 'Resolved' %}
<strong> Ticket ID: </strong>{{taskinfos.id}}<br>
<strong> Ticket Title: </strong>{{taskinfos.TASK_TITLE}}<br>
<strong>Concerned Department: </strong>{{taskinfos.TASK_DEPARTMENT}}<br>
<strong> Ticket Created By: </strong>{{taskinfos.TASK_CREATED}}<br>
<strong> Ticket Created On: </strong>{{taskinfos.TASK_CREATED_ON}}<br>
<strong> Ticket Due Date: </strong>{{taskinfos.TASK_DUE_DATE}}<br>
<strong> Ticket Reward: </strong>{{taskinfos.TASK_REWARD}}<br>
<strong> Ticket Status: </strong>{{taskinfos.TASK_STATUS}}<br>

```

```

<strong> Ticket Description: </strong>{{taskinfos.TASK_DESCRIPTION}}<br>
<br>
{% endif %}
</div>

<div class="card-footer text-body-secondary">
    {% if taskinfos.TASK_STATUS == 'Open' and taskinfos.TASK_CREATED ==
request.user %}
        <a href="{% url 'base' %}" class="btn btn-primary">Go back</a>
        <a href="{% url 'task_update' taskinfos.id%}" class="btn btn-warning">Edit Info</a>
        <a href="{% url 'task_delete' taskinfos.id%}" class="btn btn-danger">Delete</a>
        {% elif taskinfos.TASK_STATUS == 'In-process' and taskinfos.TASK_CREATED ==
request.user %}
            <a href="{% url 'base' %}" class="btn btn-primary">Go back</a>
            <a href="{% url 'task_update' taskinfos.id%}" class="btn btn-warning">Edit Info</a>
            <a href="{% url 'task_delete' taskinfos.id%}" class="btn btn-danger">Delete</a>
            {% elif taskinfos.TASK_STATUS == 'In-process' and taskinfos.TASK_CREATED !=
request.user %}
                <a href="{% url 'base' %}" class="btn btn-primary">Go back</a>
                {% elif taskinfos.TASK_STATUS == 'Closed' and taskinfos.TASK_CREATED !=
request.user %}
                    <a href="{% url 'base' %}" class="btn btn-primary">Go back</a>
                    {% elif taskinfos.TASK_STATUS == 'Closed' and taskinfos.TASK_CREATED ==
request.user %}
                        <a href="{% url 'base' %}" class="btn btn-primary">Go back</a>
                        <a href="{% url 'task_resolved' taskinfos.id%}" class="btn btn-
warning">Resolved</a>
                        <a href="{% url 'task_reopen' taskinfos.id%}" class="btn btn-danger">Reopen</a>
                        {%elif taskinfos.TASK_STATUS == 'Reopened' and taskinfos.TASK_CREATED ==
request.user %}
                            <a href="{% url 'base' %}" class="btn btn-primary">Go back</a>
                            <a href="{% url 'task_update' taskinfos.id%}" class="btn btn-warning">Edit Info</a>
                            <a href="{% url 'task_delete' taskinfos.id%}" class="btn btn-danger">Delete</a>
                            {%elif taskinfos.TASK_STATUS == 'Resolved' and taskinfos.TASK_CREATED ==
request.user %}

```

```

        <a href="{% url 'base' %}" class="btn btn-primary">Go back</a>
        <a href="{% url 'task_update' taskinfos.id%}" class="btn btn-warning">Edit Info</a>
        <a href="{% url 'task_delete' taskinfos.id%}" class="btn btn-danger">Delete</a>
        {%elif taskinfos.TASK_STATUS == 'Resolved' and taskinfos.TASK_CREATED !=
request.user %}
        <a href="{% url 'base' %}" class="btn btn-primary">Go back</a>
        {% else %}
        <a href="{% url 'base' %}" class="btn btn-primary">Go back</a>
        <a href="{% url 'task_accept' taskinfos.id%}" class="btn btn-success">Accept</a>
        {% endif %}
    </div>
</div>
{%endblock content%}
</body>
</html>

```

Update_Profile.html:

```

{%extends 'Base.html' %}
<body>
    {%block content%}
    <div class="col-4 offset-3 "><br>
    <h1 class="text-center">User Info</h1>
    <hr>
    <form action="" method="post">
        {% csrf_token %}
        {{form}}      <br>
        <button type="submit" class="btn btn-primary">Save Info</button>
    </form>
    </div>
    {%endblock content%}
</body>
</html>

```

Update_Task.html:

```

{%extends 'Base.html' %}
<body>

```

```

    {%block content%}
    <div class="col-4 offset-3 "><br>
    <h1 class="text-center">Ticket Info</h1>
    <hr>
    <form action="" method="post">
        {% csrf_token %}
        {{form}}      <br>
        <button type="submit" class="btn btn-primary">Save Info</button>
    </form>
    </div>
    {%endblock content%}
</body>
</html>

```

Userprofile.html:

```

{%extends 'Base.html' %}
<body>
    {%block content%}
    <div class="col-4 offset-3">
        <br>
        <div class="card text">
            <div class="card-header">
                <h5>User Profile</h5>
            </div>
            <div class="card-body">
                {% for ProfileData in ProfileDatas %}
                <strong>First Name: </strong>{{request.user.first_name}}<br>
                <strong>Last Name: </strong>{{request.user.last_name}}<br>
                <strong>Email: </strong>{{request.user.email}}<br>
                <strong>Address: </strong>{{ProfileData.Address}}<br>
                <strong>City: </strong>{{ProfileData.City}}<br>
                <strong>State: </strong>{{ProfileData.State}}<br>
                <strong>User ID: </strong>{{ProfileData.id}}<br>
                {% for AccountData in AccountDatas %}

```



```

        <strong>Account Name: </strong>{{AccountData.ACCOUNT_HOLDER}}<br>
        <strong>Account Balance: </strong>{{AccountData.ACCOUNT_BALANCE}}<br>
        {% endfor %}
    <br>
    <a href="{% url 'update_profile' ProfileData.id %}" class="btn btn-primary">Edit
Info</a>
    <a href="{% url 'update_profile' ProfileData.id %}" class="btn btn-info">Show
Transactions</a>
</div>
<div class="card-footer text-body-secondary">
</div>
</div>
{% endfor %}
</div>
{%endblock content%}
</body>
</html>

```

password_reset_complete.html:

```

{% extends 'base.html' %}
{% block content %}
<p>
    Your password has been set. You may go ahead and <a href="{% url 'login' %}">log in</a>
now.
</p>
{% endblock %}

```

password_reset_confirm.html:

```

{%extends 'base.html'%}
{% block content %}
    {% if validlink %}
    <h3>Change password</h3>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
    
```

```

    <button type="submit">Change password</button>
</form>
{% else %}

<p>
    The password reset link was invalid, possibly because it has already been used.
    Please request a new password reset.
</p>
{% endif %}
{% endblock %}

```

password_reset_done.html:

```

{% extends 'base.html' %}
{% block content %}

<p>
    We've emailed you instructions for setting your password. If your account exists, You
    should receive the email shortly.
</p>
<p>
    If you don't receive an email, please make sure you entered the address you registered with,
    and check your spam folder.
</p>
{% endblock %}

```

password_reset_form.html:

```

{% extends 'base.html' %}
{% block content %}

<p>
    We've emailed you instructions for setting your password. If your account exists, You
    should receive the email shortly.
</p>
<p>
    If you don't receive an email, please make sure you entered the address you registered with,
    and check your spam folder.
</p>
{% endblock %}

```

CHAPTER 6

SNAPSHOTS

6.1 GENERAL

This chapter presents the visual representation of the working IT Ticketing System in the form of output screenshots. These snapshots are taken during the implementation and testing phase to demonstrate the system's functionalities, user interfaces, and flow. They help understand how the actual application appears to users (Admin, IT Staff, and Employees) and what actions can be performed at various stages of the ticketing process.

Screenshots serve as visual evidence of:

- System interfaces
- User interactions
- Core functionalities like login, ticket creation, assignment, status updates, etc.
- The output produced after successful execution of each function

6.2 OUTPUT SNAPSHOTS

1. Login Page

- Snapshot showing the login interface for Admin, IT Staff, and User.
- Fields for username, password, and login button.
- Role-based redirection after login.

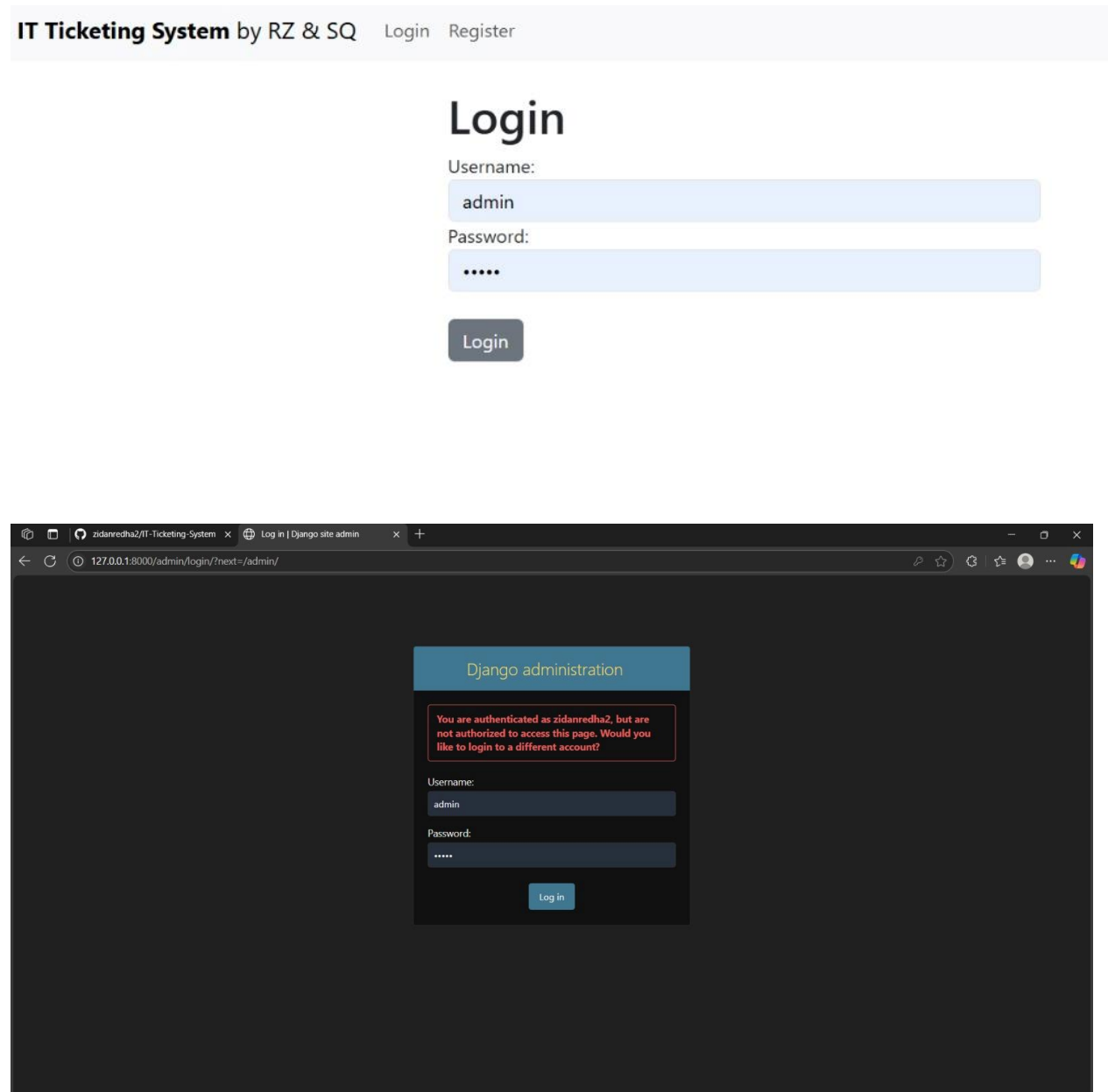


Fig 6.2.1 Login Page

2. User Dashboard

- Interface after a user logs in.
- Options to raise a ticket, view previous tickets, and check status.

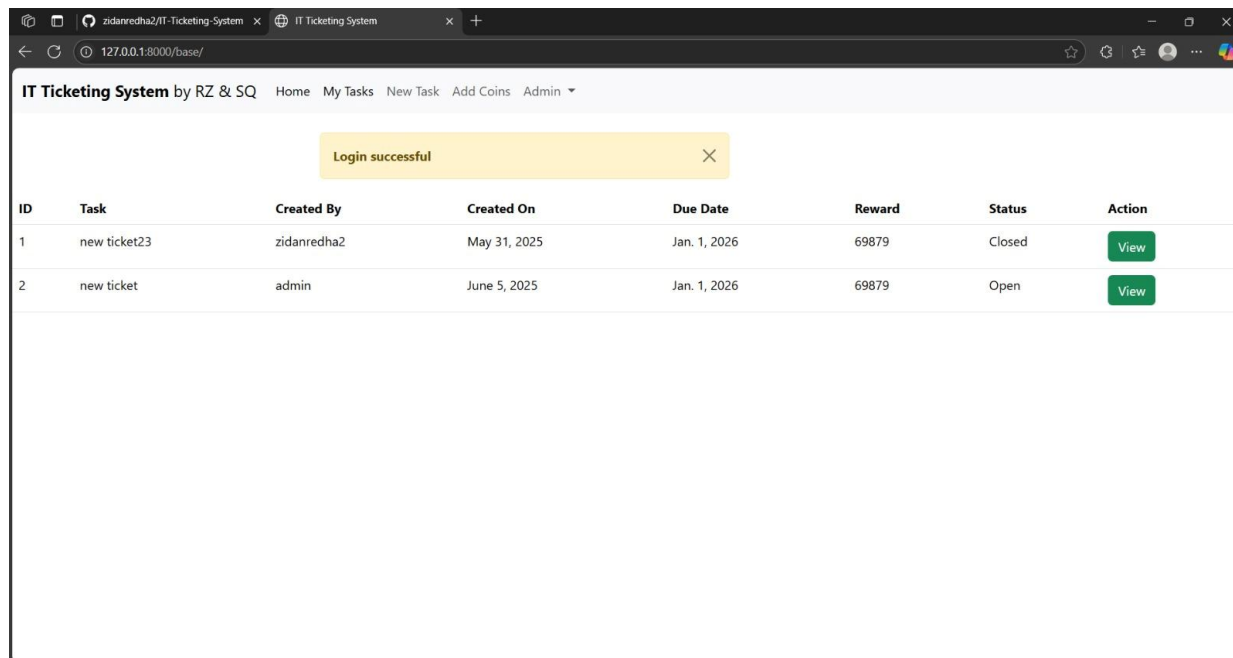


Fig 6.2.2 User Dashboard

3. Raise a New Ticket

- Form for employees to raise a new ticket.
- Fields include subject, description.

Task Assignment

Task Title:

Concerned Department:

Task Due Date:

Task Reward:

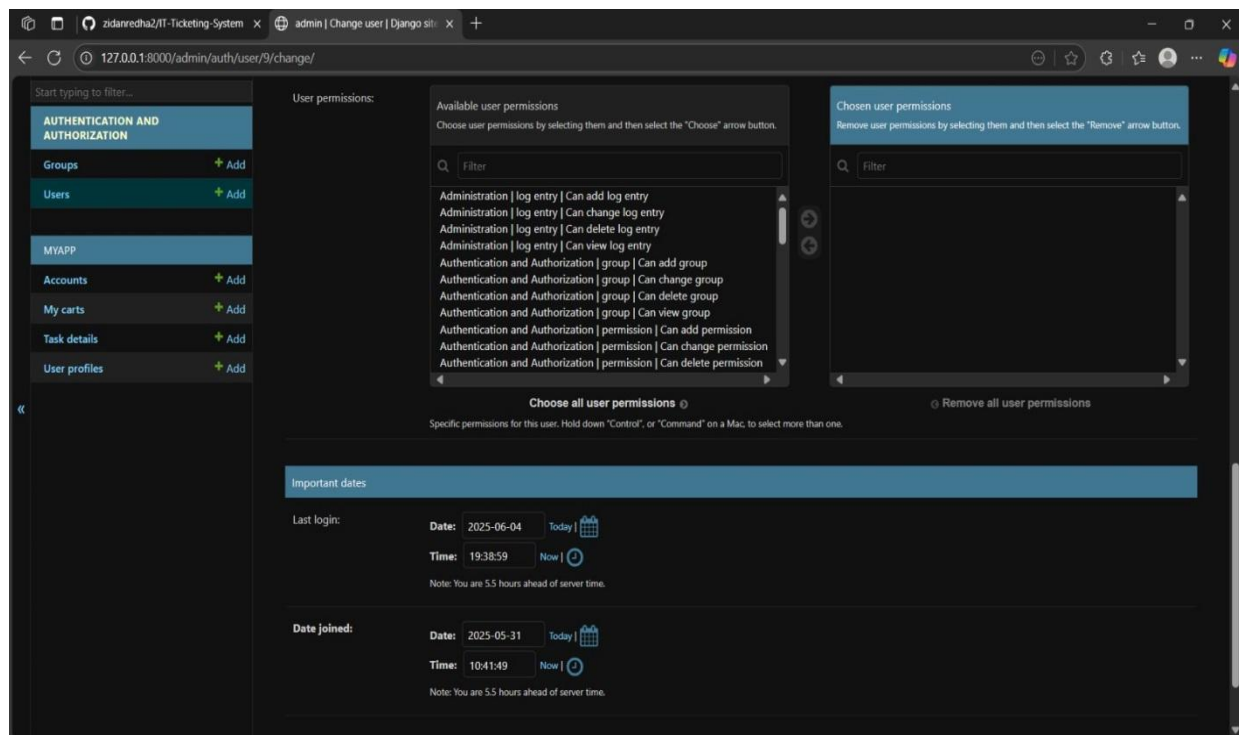
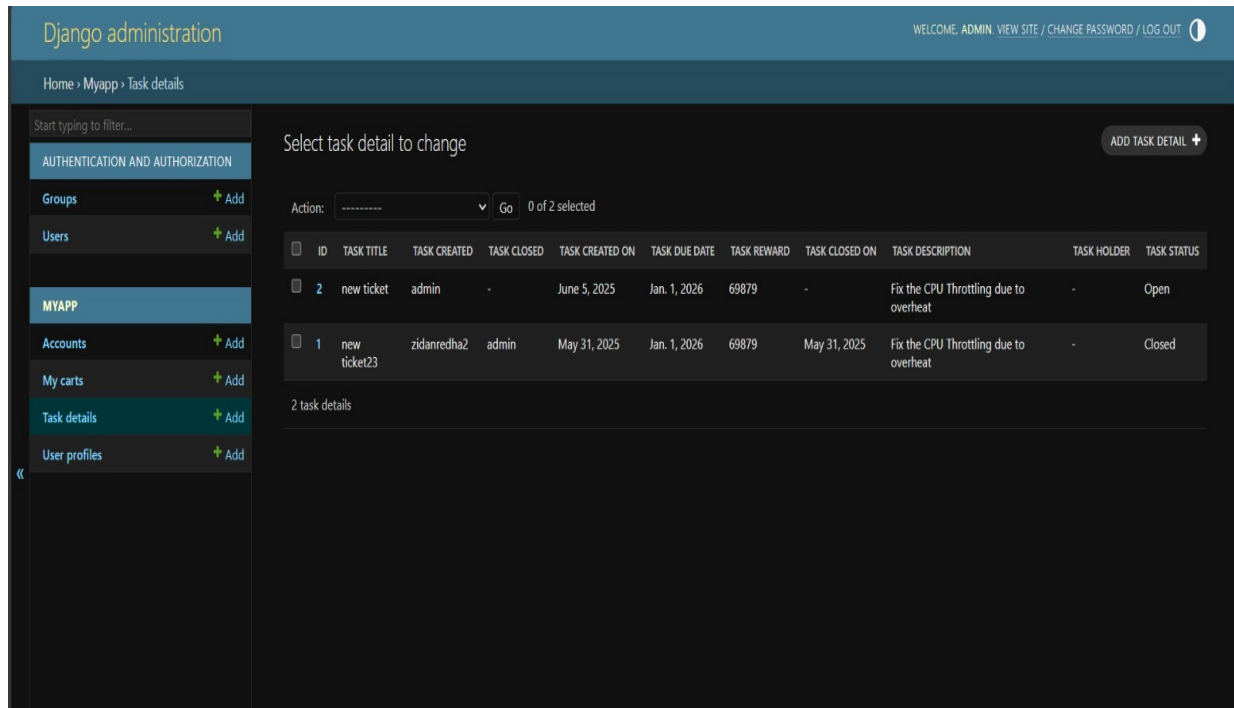
Task Description:

[Assign](#)

Fig 6.2.3 Raise a new Ticket

4. Admin Dashboard

- View for admin showing all submitted tickets.
- Buttons for managing tickets, managing users, and tracking ticket history.



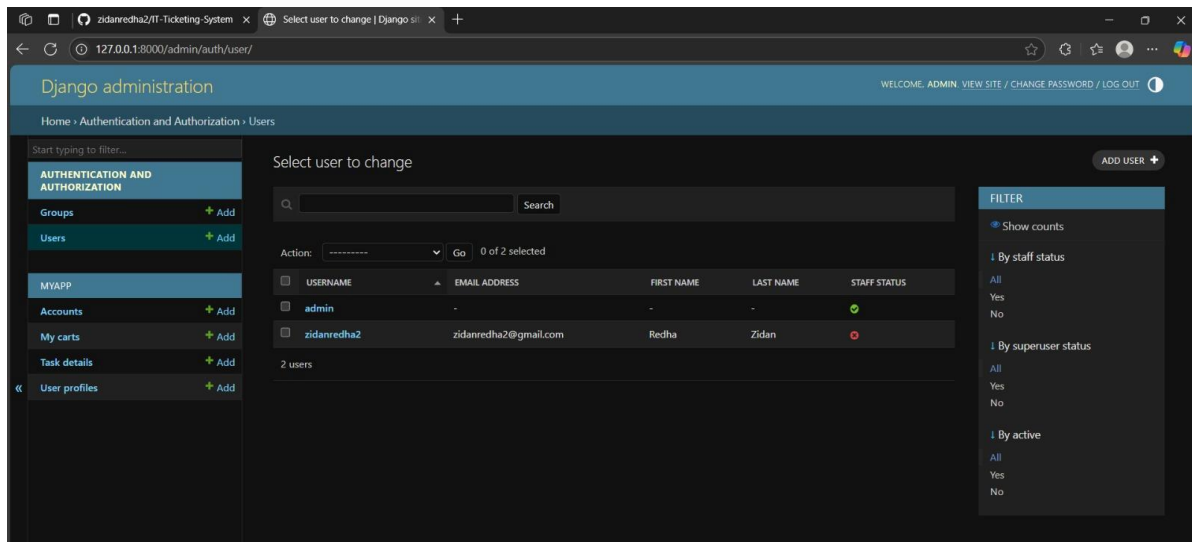


Fig 6.2.4 Admin Dashboard

5. Update Ticket Status

- Interface for employee to edit ticket information or delete it altogether if it was raised by an accident.

Task Info

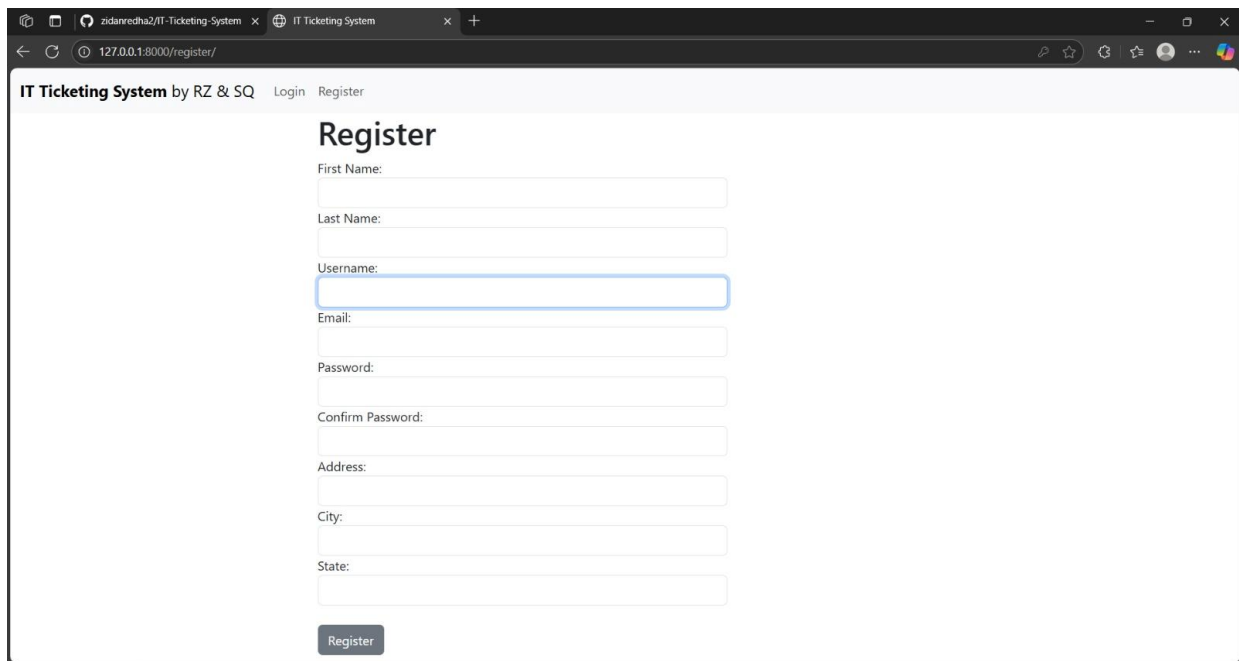
Task ID: 8
Task Title: Ticket
Concerned Department: Finance
Task Created By: zidanredha2
Task Created On: June 23, 2025
Task Due Date: Jan. 1, 2026
Task Reward: 12288
Task Status: Open
Task Description: Fix the CPU issue

Go back
Edit Info
Delete

Fig 6.2.5 Update Ticket Status

6. Register Page

- Registration form that allows us to register as new employees.



The screenshot shows a web browser window with the URL `127.0.0.1:8000/register/`. The page title is "IT Ticketing System by RZ & SQ" and the navigation bar includes "Login" and "Register" links. The main heading is "Register". The form contains the following fields: First Name, Last Name, Username (highlighted with a blue border), Email, Password, Confirm Password, Address, City, and State. A "Register" button is located at the bottom of the form.

Fig 6.2.6 Register Page

CHAPTER 7

SOFTWARE TESTING

7.1 GENERAL

Software testing is a critical phase in the development of the IT Ticketing System. It ensures that the system functions correctly, meets user requirements, and is free from bugs or issues that could affect usability or performance. Testing validates both functional and non-functional aspects of the application. It helps identify and fix errors before deployment, ensuring reliability, accuracy, and a smooth user experience. A well-tested application also builds trust and enhances system stability.

7.2 DEVELOPING METHODOLOGIES

For the IT Ticketing System, the following testing methodologies were followed:

1. White Box Testing:

White box testing focuses on the internal logic, structure, and code of the application. Developers tested individual functions such as ticket creation, status updates, and login verification by checking for logical correctness and proper flow of control.

2. Black Box Testing:

Black box testing was used to validate the system against its functional requirements without looking into the internal code. Testers focused on user input and system output, such as:

- Ticket successfully submitted after form input.
- Proper error messages displayed for invalid login credentials.
- Status updates reflect correctly across user and admin dashboards.

3. Top-Down Integration Testing:

This method was used to test high-level modules first (like the dashboard or main menus), followed by sub-modules (like ticket update or assign functions). This helped catch issues in early integrations between modules.

4. Incremental Testing:

Each module was tested independently as it was developed and then integrated with the overall system. This approach helped isolate bugs early and reduce debugging complexity during final integration.

7.3 TYPES OF TESTING

Several types of testing were carried out during the software testing phase:

1. Unit Testing:

- Each function/module (e.g., login validation, raise ticket form submission) was tested individually.
- Ensures that each component behaves as expected.

2. Integration Testing:

- After successful unit testing, integration testing was done to check data flow between modules.
- Example: Ensuring that the ticket ID generated in the user module is accurately reflected in the admin assignment module.

3. System Testing:

- The complete system was tested in a simulated real-world environment.
- All functionalities like login, raise ticket, accept, update, and view history were tested collectively.

4. Functional Testing:

- All features were validated against the functional requirements.
- Example: Test cases were written to ensure that the "Raise Ticket" feature does not allow submission without mandatory fields.

5. Usability Testing:

- The system was tested for ease of use, interface clarity, and navigation.
- Feedback was collected from potential users to improve the UI/UX.

6. Security Testing:

- Login module tested for incorrect login attempts.
- Role-based access verified to ensure that users cannot access admin/staff functionalities.

CHAPTER 8

FUTURE ENHANCEMENTS

8.1 FUTURE ENHANCEMENTS

The current version of the IT Ticketing System successfully addresses the basic needs of managing technical issues, ticket tracking, and staff assignment. However, several enhancements can be made in future versions to improve performance, usability, and scalability. Some potential future enhancements include:

1. Email & SMS Notifications:

- Automatically notify users and staff when tickets are created, updated, or resolved.

2. Live Chat Support:

- Integrate real-time chat between users and support staff for quicker resolutions.

3. AI-based Ticket Categorization:

- Use machine learning to automatically categorize and assign tickets based on keywords or past data.

4. Mobile Application:

- Develop Android and iOS apps for easier ticket management on mobile devices.

5. Ticket Analytics Dashboard:

- Add visual reports and statistics (e.g., number of tickets, average resolution time) for better decision-making.

6. Multi-language Support:

- Allow users to access the system in different regional languages for better accessibility.

7. Role-based Workflow Customization:

- Admins can define new roles and assign custom permissions dynamically.

8.2 CONCLUSION

The IT Ticketing System provides an efficient platform to manage technical issues in an organized and systematic manner. It simplifies the process of raising, assigning, tracking, and resolving tickets for both employees and IT Support staff. The system reduces manual tracking efforts, ensures transparency, and enhances productivity.

Through careful design, development, and testing, the system has been implemented to meet functional and non-functional requirements. It is scalable, user-friendly, and adaptable to different organizational environments. With proper future enhancements, it can be extended into a more advanced support management platform for enterprises.

8.3 REFERENCES:

Books & Academic Sources:

- Roger S. Pressman, *Software Engineering – A Practitioner's Approach*, McGraw-Hill
- Ian Sommerville, *Software Engineering*, Pearson Education

Websites:

- www.geeksforgeeks.org
- www.w3schools.com
- www.stackoverflow.com
- developer.mozilla.org