CHAPTER 1

INTRODUCTION

In everyday life, we often see long queues at places like banks, hospitals, colleges, railway counters, and customer service centers. Standing in these lines for a long time can be tiring and frustrating, and it also affects the quality of service. Because of this, managing queues properly has become very important in today's fast-moving world. A Queue Management System (QMS) helps in handling people or requests in an organized and fair manner.

The idea of a queue management system comes from the basic queue data structure, which works on the First Come First Serve (FCFS) rule. This simply means that whoever comes first is served first. In real life, we follow this rule almost everywhere, and this project tries to represent the same concept using a simple program. The main focus is to keep the system easy to understand and close to real-life situations.

Earlier, queues were handled manually, which sometimes caused confusion, crowding, and unfair treatment. There was no proper way to track who came first or who should be served next. With the use of computers, queue handling has become more systematic and reliable. This mini project shows how basic programming concepts can be used to manage a queue in a simple and effective way.

The aim of this project is to design a user-friendly queue management system that can perform basic operations like inserting a new element into the queue, deleting an element from the queue, and displaying the current queue status. The project avoids unnecessary complexity and focuses more on clear logic and correct working.

Overall, this Queue Management System mini project provides a practical learning experience and gives a strong foundation for understanding data structures and their applications.

1.1. Problem Introduction

1.1.1. Motivation

The motivation behind choosing the Queue Management System as a mini project comes from everyday experiences. Almost everyone has faced long waiting lines and confusion while standing in queues. These situations waste time and often create frustration among people. Observing these real-life problems encouraged the development of a simple system that can manage queues in an organized and

fair way. This project is motivated by the idea of solving a common problem using basic programming knowledge and data structure concepts.

1.1.2. Project Objective

The main objective of this project is to design and develop a simple Queue Management System that works on the First Come First Serve principle. The project aims to help users understand how a queue operates by performing basic actions such as inserting elements into the queue, deleting elements from the queue, and displaying the current queue. Another objective is to improve logical thinking and provide hands-on experience with data structures in a practical manner.

1.1.3. Scope of the Project

The scope of this project is limited to implementing a basic queue management system for learning purposes. It focuses only on fundamental queue operations and does not include advanced features such as priority queues, real-time databases, or graphical user interfaces. However, the project provides a strong foundation that can be further expanded in the future for use in real-life applications like hospitals, banks, or service centers.The scope of this project is limited to implementing a basic queue management system for learning purposes. It focuses only on fundamental queue operations and does not include advanced features such as priority queues, real-time databases, or graphical user interfaces. However, the project provides a strong foundation that can be further expanded in the future for use in real-life applications like hospitals, banks, or service centers.

1.2. Related Previous Work

Queue management has been studied and implemented in various fields for a long time. Earlier systems were mostly manual, where queues were managed by people without the help of technology.

(Research Paper-1)

Smart Queueing Management System for Digital Healthcare (2023)

- **Authors**: Mayank Dutta, Fakhra Najm, Dhruv Tomar, Shabana Mehfuz
- **Overview:** Proposes an intelligent digital queue system prioritizing patients dynamically based on urgency and other factors. It aims to reduce physical wait times and congestion while optimizing hospital workflows through real-time data integration and multi-interface support. The system enhances patient experience and operational transparency.
- **Gap:** Limited real-world deployment examples; integration with legacy systems and addressing digital exclusion require more study.

(Research Paper-2)

QUEUING MANAGEMENT SYSTEM IN MANUEL S. ENVERGA UNIVERSITY (2023)

- **Authors:** Ronnel F. Maala, Noelyn B. Sebua, Kim Lester L. Evangelista
- **Overview:** Focuses on digitizing queue management for university administrative services. Enables virtual queueing, real-time updates, and performance tracking, significantly reducing waiting times and operational errors. The system improves user satisfaction through easy access and communication.
- **Gap**: Tailored for a specific institution; broad applicability and data security aspects need further evaluation.

(Research Paper-3)

Queuing Theory-Based Simulation and Optimization of Healthcare Processes (2025)

- **Authors**: M. Bahadori, S. Mohammadnejhad, R. Ravangard, E. Teymourzadeh
- **Overview:** Applies queuing theory models (M/M/1, M/M/c) and simulation tools to optimize patient flow in healthcare settings like emergency departments. Demonstrates how modeling patient arrivals and service times can reduce wait times, alleviate bottlenecks, and improve resource allocation. Incorporates data-driven approaches adapting to dynamic patient loads.
- **Gap:** Model generalization and adoption challenges remain; integration with existing hospital IT systems and emergency scenario handling need further work.

(Research Paper-4)

A Survey on Queue Management Systems (2025)

- **Authors:** Prof. Neha Titarmare, Prof. Ashwini Yerlekar
- **Overview:** Reviews current digital and hybrid queue management systems across sectors. Highlights the role of AI, IoT, and analytics in enhancing customer service and operational efficiency. Discusses system types and application challenges, emphasizing need for scalability and better human-centered design.
- **Gap:** Calls for empirical validation, improved user adaptation studies, and privacy safeguards.

(Research Paper-5)

CLIQUE: A Web-Based Queue Management System with Real-Time Notification (2022)

- **Authors:** Marc Lester Z. Mallari, Joshua S. Guintu, Yoben C. Magalong, and Daisy S. Yap
- **Overview:** Describes a cloud-based web platform providing real-time queue notifications to users. Aims to minimize waiting times through effective communication and queue visibility, suitable for various service domains.
- **Gap**: Mainly technical development focus; lacks comprehensive testing across diverse environments and user behaviors.

1.3. Organization of the Report.

This report is organized into different chapters to clearly explain the project work. The first chapter introduces the project and explains the need for a queue management system. The next chapters describe the motivation, objectives, and scope of the project. Further chapters explain the working of the system, the algorithm used, and implementation details. The final chapter concludes the project and discusses possible future improvements.

CHAPTER 2

LITERATURE SURVEY

Queue management has been an important topic of study in both academic and practical fields due to its wide use in daily life. Many researchers and developers have worked on different ways to manage queues efficiently in service-based systems. The main focus of these studies has been to reduce waiting time, avoid overcrowding, and improve customer satisfaction. Earlier work mainly relied on manual handling, while later studies introduced computerized systems using basic data structures and algorithms. With the advancement of technology, queue management systems have become more structured and reliable, making them suitable for various real-world applications.

2.1 Manual Queue Management System

In manual queue management systems, queues are handled without the use of computers or automated tools. People stand in physical lines and wait for their turn based on verbal instructions or visual observation. This system is simple but often leads to problems such as confusion, line breaking, unfair service, and time wastage. Many studies highlighted that manual systems are inefficient, especially in crowded places like hospitals and railway stations, where managing large numbers of people becomes difficult.

2.2 Computer-Based Queue Management System

Computer-based queue management systems were introduced to overcome the drawbacks of manual systems. These systems use computers to store and manage queue data digitally. Each person or request is given a position in the queue, and services are provided according to that order. Research has shown that computer-based systems improve accuracy and reduce human errors. Such systems are commonly used in banks, offices, and service centers to ensure fair and organized service.

2.3 Queue Data Structure and FCFS Algorithm

The queue data structure plays a major role in queue management systems. Most existing systems are based on the First Come First Serve (FCFS) algorithm, where the first element added to the queue is the first one to be removed. This algorithm is simple and easy to

implement, making it suitable for basic applications. Previous studies have shown that FCFS works well for general service systems where priority is not required. Due to its simplicity, FCFS is widely used in educational and real-life queue applications.

2.4 Automated Queue Management Using Programming Languages

With the growth of programming languages like C, C++, Java, and Python, automated queue management systems have become easier to develop. Researchers have implemented queue systems using arrays and linked lists to perform operations such as insertion, deletion, and display. These studies emphasize that programming-based queue systems help students understand data structures clearly and provide a practical approach to problem-solving.

SUMMARY

In this literature survey, different approaches to queue management were studied. Manual queue systems were found to be simple but inefficient and prone to errors. Computer-based systems improved accuracy and organization by using digital methods. The use of queue data structures and the FCFS algorithm proved to be effective for managing basic queues. Automated queue management systems developed using programming languages offer a simple and practical solution, especially for learning purposes. Overall, the surveyed studies support the idea that a basic queue management system is useful and relevant for real-world as well as academic applications.

CHAPTER 3

SOFTWARE REQUIREMENT SPECIFICATION

3.1 Product Perspective

The Digital Queue Management System (DQMS) is a **web-based application** designed to streamline and automate the process of issuing, managing, and monitoring queues in service-based organizations such as hospitals, banks, government offices, and customer service centers.

The system operates as an **independent, self-contained product**, but can also integrate with other organizational systems such as employee management or notification services. Compared to traditional manual queue methods, DQMS reduces waiting time, improves user experience, and provides real-time updates on queue status.

A simple contextual block diagram is as follows:

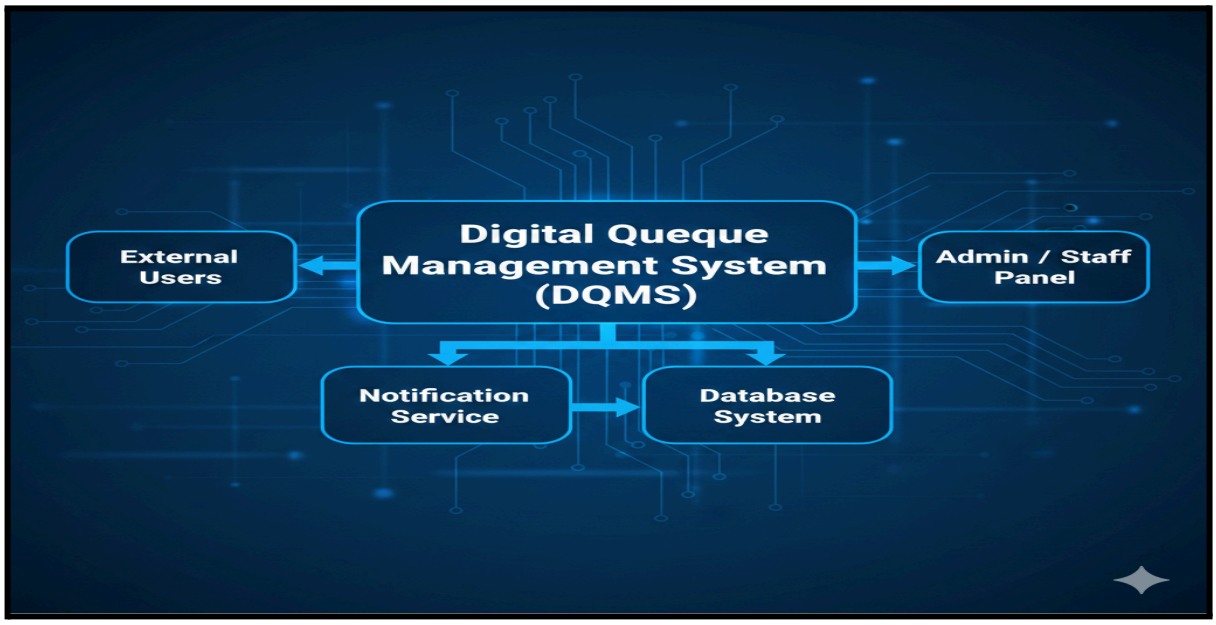


Fig 3.1

The system functions as a black box, interacting with users and staff through web interfaces.

3.1.1 System Interfaces

The DQMS interacts with the following external systems:

1. **SMS/Email Notification Service (Optional)**

    - Used to send queue token alerts and updates.
    - Interface through REST APIs (if enabled).

2. **Organizational Database Server**

    - Stores queue data, user logs, and service counters.

3. **Internal Staff Systems**

    - Admin/staff login panel where queue counters are operated.

3.1.2 User Interfaces

The DQMS provides two types of interfaces:

1. **Web-Based Graphical User Interface (GUI)** for customers

    - View services
    - Generate queue token
    - Check real-time queue status

2. **Web Admin Panel** for staff

    - Call next token
    - Skip/cancel token
    - View queue analytics

The interface is simple, responsive, and optimized for both desktop and mobile devices. Buttons are large, readable, and accessible for users with minimal technical knowledge.

3.1.3 Hardware Interfaces

The system has minimal hardware interface requirements:

- Standard computer or mobile device with a web browser
- Printer (optional) for printing physical queue tokens
- Display screen (optional) for showing the currently called token number

If no external hardware is used, the system still functions completely online.

3.1.4 Software Interfaces

The Digital Queue Management System (DQMS) depends on several software components to operate efficiently. The core backend functionality relies on a relational database system such as **MySQL**, which is used to store queue tokens, service categories, user logs, and system configuration data. The application communicates with the database using standard SQL queries over a secure connection.

The system is hosted on a **web server** such as Apache or Nginx, which manages client requests and executes the application code. The entire system is accessible through a web browser (e.g., Chrome, Firefox, Edge), ensuring cross-platform compatibility without the need for additional installations. All communication between the client and server uses the **HTTP/HTTPS protocol**, ensuring secure data transmission.

If organizations choose to enable notifications, the system can integrate with an **SMS Gateway or Email Service** using REST APIs. These third-party services allow the DQMS to send alerts such as token confirmations, estimated waiting times, or token call notifications. Each of these software interfaces ensures smooth operation and supports the overall functioning of the system.

Software Interfaces

Software	Version	Purpose	Interface
MySQL / SQL Database	5.x or above	Stores queue data, user logs and service counters	SQL Queries / API
Web Server (Apache/Nginx)	Latest	Hosts the web app	HTTP/HTTPS
Optional SMS Gateway	Provided by client	Sends token alerts	REST API
Browser	Chrome/Firefox	User access	HTTP/HTTPS

Fig 3.2

3.1.5 Communications Interfaces

- **HTTP/HTTPS Protocol** for all web communications
- **JSON/REST API** for interaction with optional notification systems
- **LAN/WiFi** for internal communication between display system and admin panel

No custom communication protocol is used.

3.1.6 Memory Constraints

There are no strict memory limitations.

The system is lightweight and runs smoothly on standard web hosting services.

Recommended server RAM: **512 MB – 1 GB** (typical for basic PHP/MySQL hosting).

3.1.7 Operations

Normal and special operations include:

- Customer requests a queue token
- System issues token and updates live queue
- Staff operates the counter and calls next token
- Display screen (optional) reflects real-time updates
- Daily backup of queue data

- Disaster recovery through database restore

3.1.8 Site Adaptation Requirements

- If an organization wants on-premise installation, a local server setup is needed.
- Large display screen installation for showing token numbers (if required).
- Database must be configured before system activation.

No other physical changes are required at the customer site.

3.2 Product Functions

The major functions of the Digital Queue Management System include:

1. **Token Generation**
    - Users can generate a queue token for a specific service.
2. **Real-Time Queue Monitoring**
    - Users can check their current position in the queue.
3. **Counter Management (Admin/Staff)**
    - Call next token
    - Skip token
    - Mark token served
4. **Queue Display Board**
    - Shows current token being served.
5. **Notifications (Optional)**
    - SMS/Email alerts to customers.
6. **Reports and Analytics**

- Daily queue reports
- Counter performance
- Peak hours analysis

3.3 User Characteristics

The system is designed for:

1. **General Public (Customers)**

    - Basic understanding of smartphones or computers
    - No technical knowledge required
    - Needs a simple interface

2. **Staff/Admin Users**

    - Basic computer literacy
    - Ability to operate admin panel
    - Some training may be required

Design is influenced by the need for simplicity and accessibility.

3.4 Constraints

Regulatory Policies

- Must comply with data privacy norms.

Hardware Limitations

- Performance depends on server resources.

Interface Restrictions

- Works only on devices with browsers.

Security Requirements

- Secure login for admin
- Encrypted communication (HTTPS)

Reliability

- The system must work during peak hours.

Parallel Operation

- Multiple counters must operate simultaneously.

3.5 Assumptions and Dependencies

- Users have access to the internet.
- Organization will provide necessary network infrastructure.
- The database server is always operational.
- Optional SMS API is provided and functional.
- Browser compatibility is ensured.

Any change in these assumptions may require modification of system design.

3.6 Apportioning of Requirements.

Features planned for Version 1.0:

- Queue token generation
- Counter operations
- Real-time queue display
- Basic reports

Features delayed for future versions:

- Mobile app integration
- Voice announcement system
- AI-based queue prediction
- Advanced analytics dashboard

2.7. Use case

2.7.1 Use Case Diagram

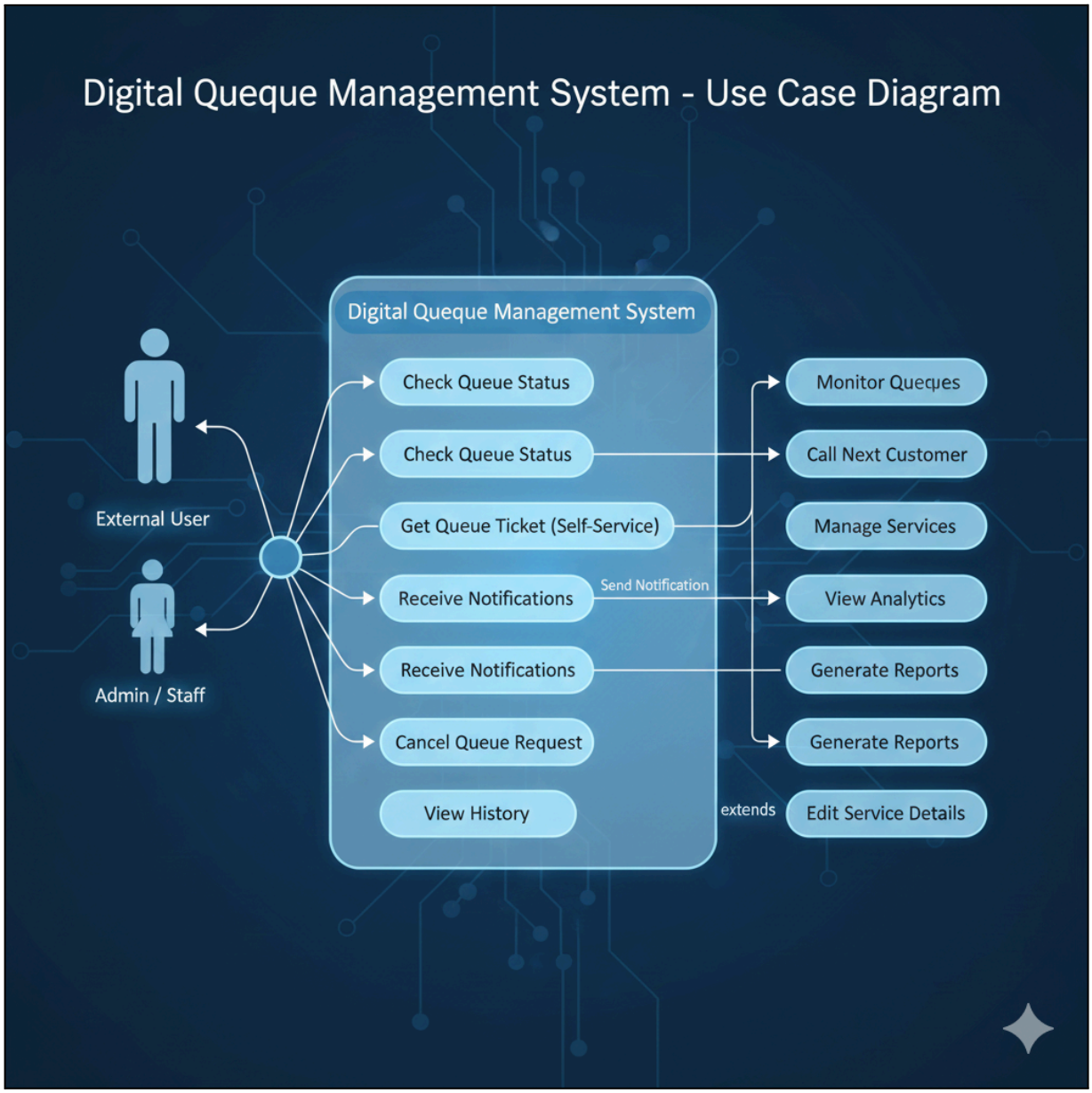


Fig 3.3

CHAPTER 4

SYSTEM DESIGN

System Design should include the following sections (Refer each figure or table in some text). Figure number should be provided below the figure and the table numbering should be provided above the table.

4.1 Architecture diagram

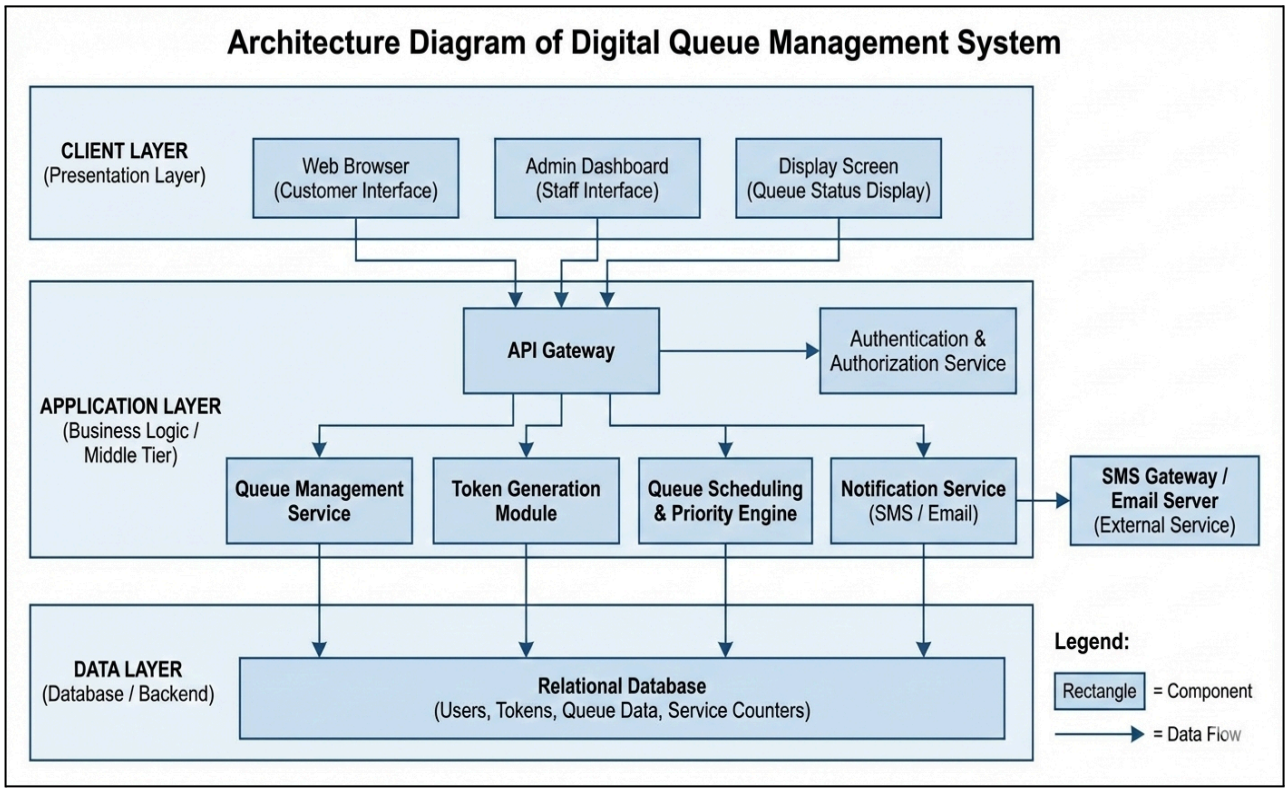


Fig 4.1

4.2 Class diagrams

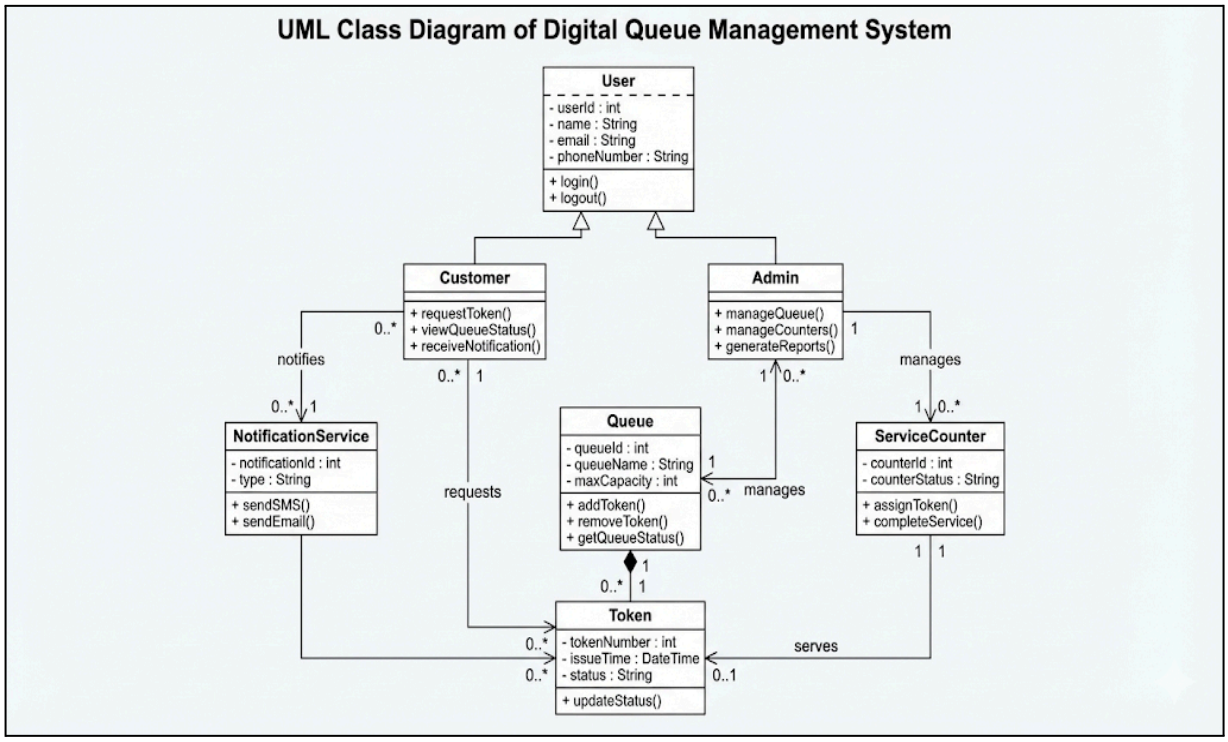


Fig 4.2

4.3 Data Flow Diagram

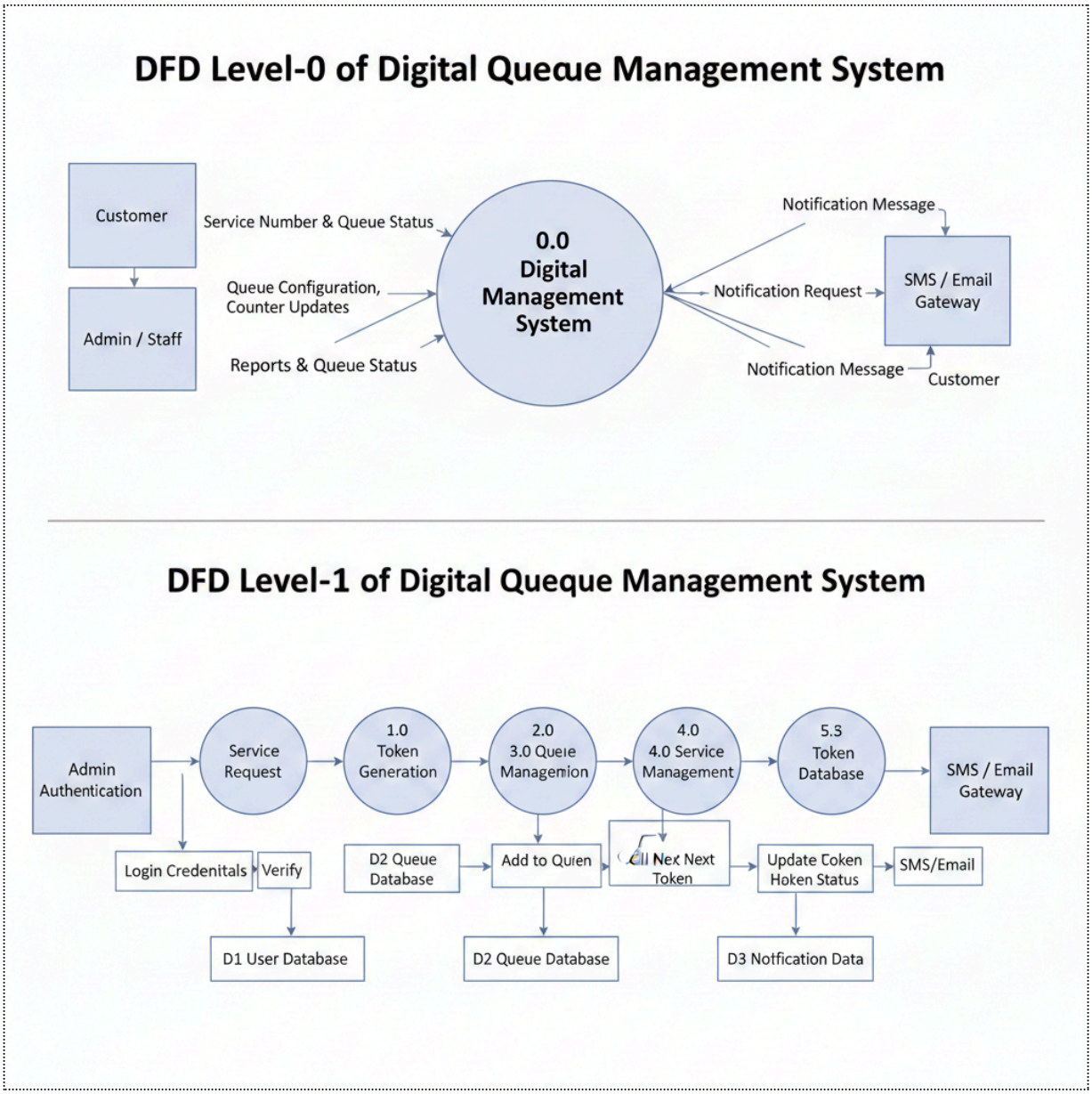


Fig 4.3

4.4 Activity Diagram

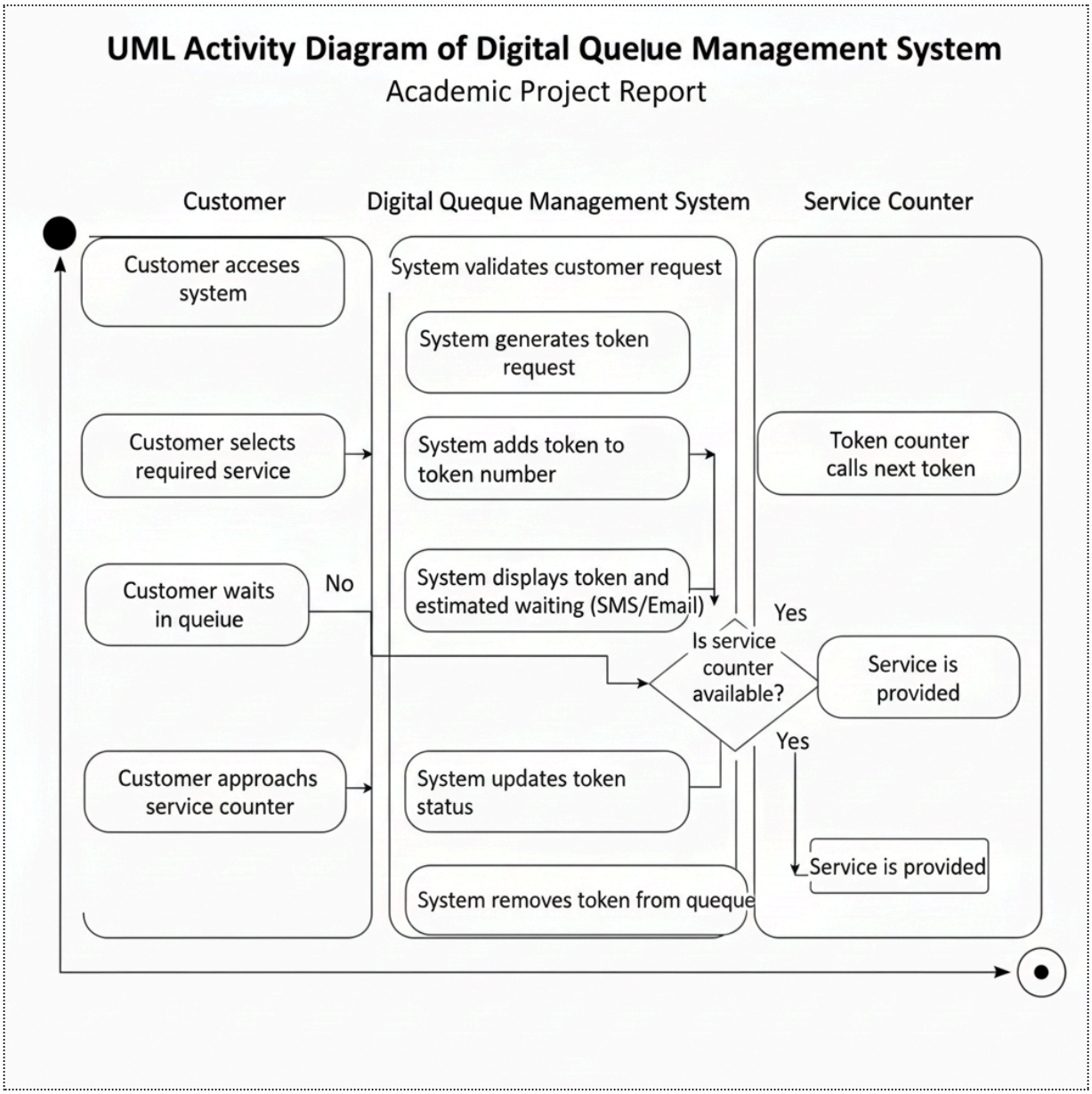


Fig 4.4

4.5 ER Diagrams

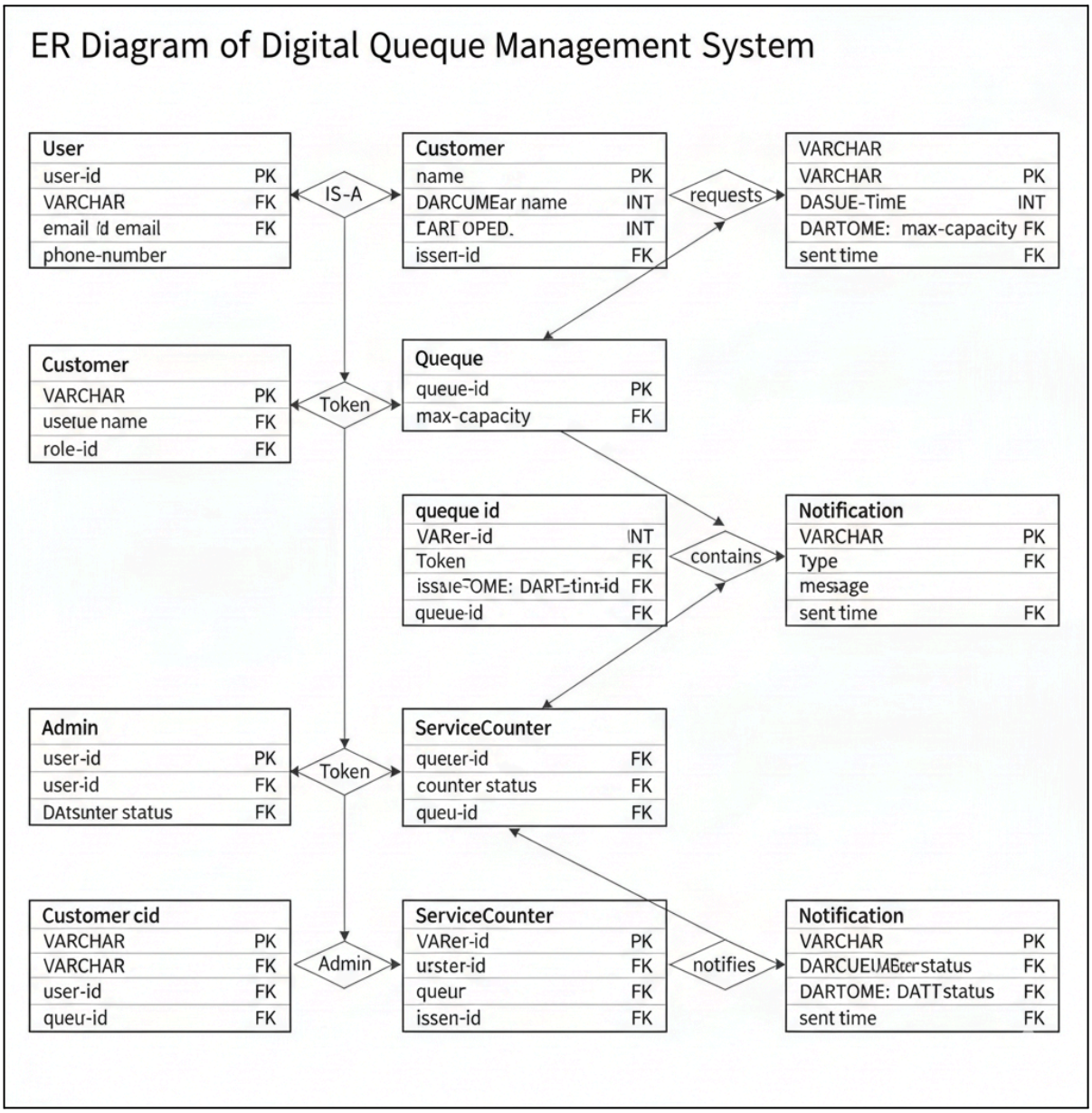


Fig 4.5

4.6 Database schema diagrams

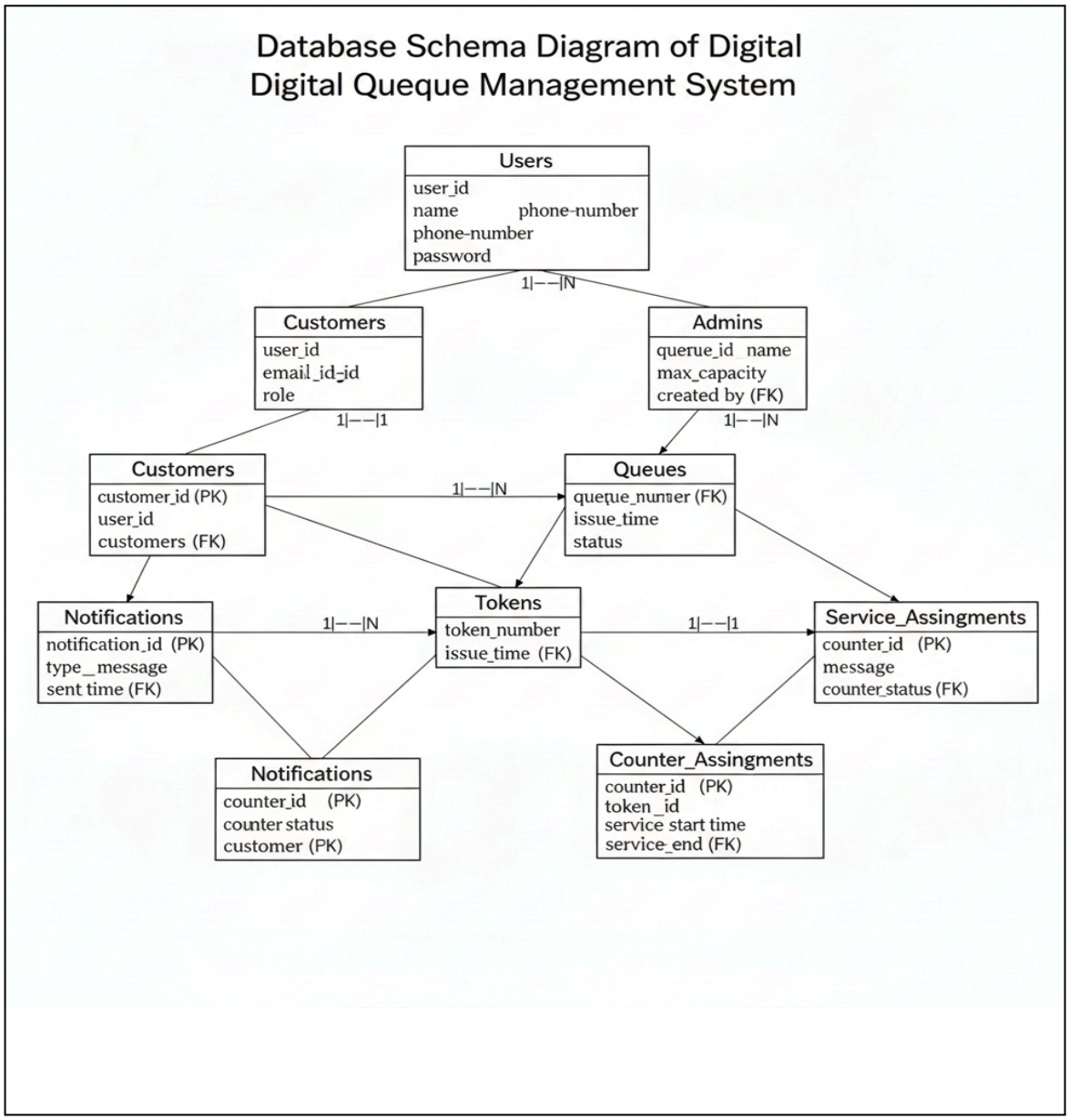


Fig 4.6

CHAPTER 5

IMPLEMENTATION AND RESULTS

5.1 Software and Hardware Requirements

This project does not require any advanced software or hardware and can be implemented on a normal computer system.

Software Requirements:

- Operating System: Windows 10 or above
- Programming Language: C / C++ / Java / Python (any one as used in the project)
- Compiler / Interpreter: GCC / Turbo C / Python IDLE / VS Code
- Text Editor or IDE: VS Code / Code::Blocks / Notepad++

Hardware Requirements:

- Processor: Intel Core i3 or higher
- RAM: Minimum 4 GB
- Hard Disk: 20 GB free space
- Keyboard and Mouse

5.2 Assumptions and dependencies

The following assumptions are made while developing this project:

- Users will enter valid input values.
- The system follows the First Come First Serve (FCFS) rule.
- The project is console-based and does not include a graphical user interface.
- The queue size is limited depending on the implementation.

5.3 Constraints

- The system uses static, rule-based logic instead of machine learning, so it cannot automatically adapt or improve based on user behavior over time.

- The project uses SQLite database, which works well for small to medium usage but may face performance issues when many users access the system at the same time.

- User authentication is not implemented, so there is no login system for Admins or Users in the current version.

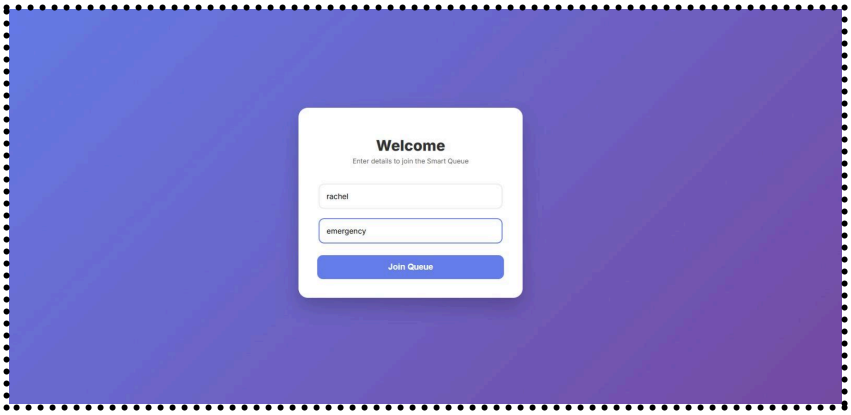
- The system depends on a stable internet connection to function properly and does not support offline usage.

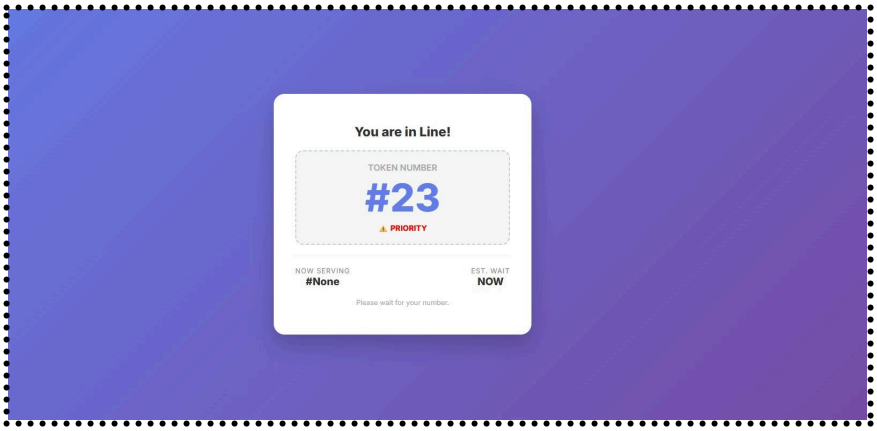
- If the browser tab is closed, the user may lose live queue updates, although the token information remains stored in the database.

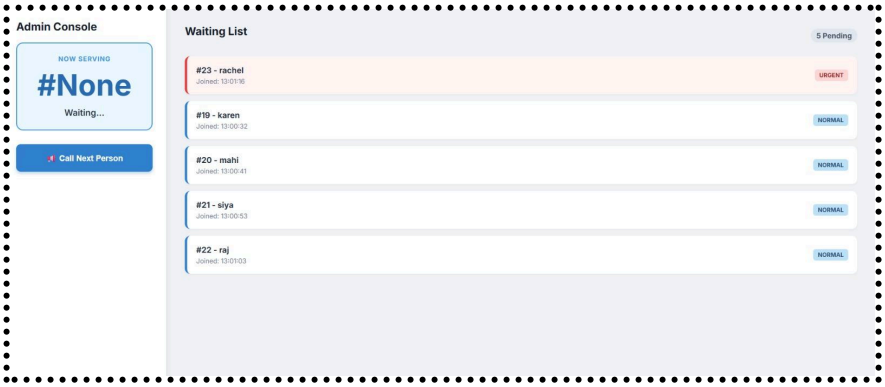
5.4 Implementation Details

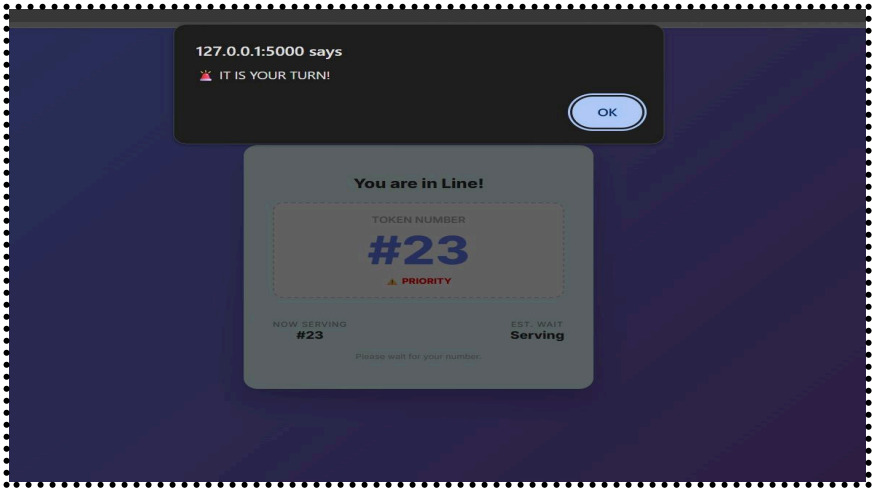
The Queue Management System is implemented using basic programming concepts and the queue data structure. The system allows users to perform operations such as generating a token, adding it to the queue, serving the next token, and displaying the current queue status. A menu-driven approach is used so that users can easily select the required operation. The First Come First Serve principle is followed throughout the execution of the program.

5.4.1 Snapshots Of Interfaces




You are in Line!
TOKEN NUMBER
#23
PRIORITY
NOW SERVING
#None
EST. WAIT
NOW
Please wait for your number.


Admin Console
NOW SERVING
#None
Waiting...
Call Next Person
Waiting List
5 Pending
#23 - rachel
Joined: 13:01:16
URGENT
#19 - karen
Joined: 13:00:32
NORMAL
#20 - mahi
Joined: 13:00:41
NORMAL
#21 - siya
Joined: 13:00:53
NORMAL
#22 - raj
Joined: 13:01:03
NORMAL


127.0.0.1:5000 says
IT IS YOUR TURN!
OK
You are in Line!
TOKEN NUMBER
#23
PRIORITY
NOW SERVING
#23
EST. WAIT
Serving
Please wait for your number.

5.4.2 Test Cases

The following test cases were used to test the system:

Test Case Report: Digital Queue Management System

Test Case No.	Input	Expected Output	Result
1	Add token	Token added successfully	Pass
2	Serve token	Token served and removed	Pass
3	Display queue	Current queue shown	Pass
4	Serve empty queue	Error message shown	Pass

5.4.3 Results

The Queue Management System worked as expected for all test cases. Tokens were generated in sequential order and served according to the First Come First Serve principle. The system successfully handled queue insertion, deletion, and display operations. The results confirm that the implemented system is simple, correct, and suitable for managing queues in basic service scenarios.

CHAPTER 6

CONCLUSION

6.1 Performance Evaluation

The performance of the Queue Management System was evaluated based on the following points:

- Correct execution of queue operations such as enqueue, dequeue, and display
- Proper generation and serving of tokens
- Effective handling of priority-based requests
- Dynamic queue size without overflow issues
- Smooth working for a single service counter

The system was tested using multiple test cases, and all operations produced the expected results. The overall performance is satisfactory for a mini project and suitable for small-scale applications.

6.2 Comparison with existing State-of-the-Art Technologies

The comparison with existing advanced queue management systems is summarized below:

- Modern systems use mobile applications and web-based interfaces
- Advanced systems include cloud databases and real-time monitoring
- Some systems use artificial intelligence to predict waiting time
- The proposed system is simpler and console-based
- The proposed system focuses on learning basic concepts rather than large-scale deployment

Although it does not match the features of state-of-the-art systems, it effectively demonstrates the fundamental working of queue management.

6.3 Future Directions

The following future enhancements can be considered to extend the work of this project:

- Support for multiple service counters to allow parallel servicing
- Development of a graphical user interface (GUI) for better user interaction
- Integration with a web or mobile application for remote token generation
- Use of artificial intelligence to predict waiting time and manage crowd flow
- Performance optimization for handling a larger number of users

Practical Implications:

This Queue Management System can be practically used in small service centers such as clinics, offices, banks, help desks, and college administrative departments. It helps reduce confusion, saves time, and ensures fair service by following an organized approach. With future improvements, the system can be scaled and adapted for real-world environments where efficient queue handling is required.

Appendix

Appendix A: Source Code

This appendix includes the complete source code developed for the Queue Management System mini project.

```
# importing necessary libraries

from flask import Flask, render_template, request, jsonify

from flask_socketio import SocketIO, emit

from flask_sqlalchemy import SQLAlchemy  # <--- NEW: Database Library

from datetime import datetime

import os

#initialise app and socket io

app = Flask(__name__)

app.config['SECRET_KEY'] = 'secret!'


# --- DATABASE CONFIGURATION ---

# a file 'queue.db' which acts as your permanent storage

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///queue.db'

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

socketio = SocketIO(app)

# --- DEFINE THE DATA MODEL (SCHEMA) ---

# In Full Stack, you define what your data looks like in the DB

class Token(db.Model):

    id = db.Column(db.Integer, primary_key=True)
```

```
    name = db.Column(db.String(100), nullable=False)
    reason = db.Column(db.String(200), nullable=False)
    join_time = db.Column(db.String(20), nullable=False)
    is_urgent = db.Column(db.Boolean, default=False)
    status = db.Column(db.String(20), default="waiting")  # waiting, serving, done
#creating  the database file if it doesn't exist
with app.app_context():
    db.create_all()
# --- AI CORE  ---
class AICore:
    def check_priority(self, reason_text):
        urgent_keywords = ['emergency', 'urgent', 'critical', 'severe']
        if any(word in reason_text.lower() for word in urgent_keywords):
            return True
        return False
        # NLP LOGIC: Scans the input for keywords to decide priority.If it finds 'emergency' or 'urgent', it returns True.
        # Convert text to lowercase to match 'Emergency' or 'emergency'
    def calculate_wait_time(self, is_priority, queue_count):
        if is_priority:
            return 0
        return (queue_count * 5) + 5
    # if priority is true and one of the words from urgent_keywords has been used than wait time becomes 0 making the system treat them as a priority
    # Normal Logic: the queue length and 5 minutes per person
```

```
    """
    REGRESSION LOGIC: Calculates time based on queue length.
    If priority is True, wait time is 0.
    """
ai_core = AICore()
# --- ROUTES ---
@app.route('/')
def user_view():
    return render_template('user.html')
@app.route('/admin')
def admin_view():
    return render_template('admin.html')
# --- API ENDPOINTS (Updated for Database) ---
@app.route('/get_token', methods=['POST'])
def get_token():
    name = request.form.get('name')
    visit_reason = request.form.get('reason')
    # receives data from user end...name and reason and checks the urgency condition
    # AI Logic:  ai asks for urgnet keywords and how long is it suppose to wait
    is_urgent = ai_core.check_priority(visit_reason)
    # Count how many are waiting in DB
    # if its urgent we apply function check_priority from class aicore
    # if not urgent than we use function calculate_wait_time from class ai_core
    queue_count = Token.query.filter_by(status='waiting').count()
```

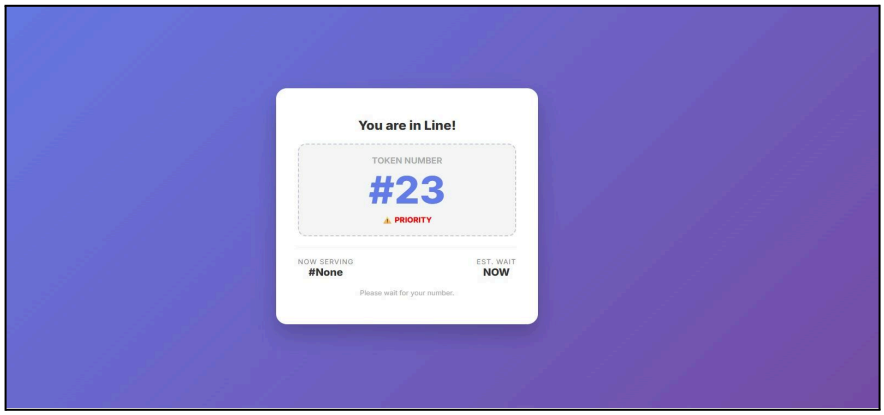
```
    wait_time = ai_core.calculate_wait_time(is_urgent, queue_count)
    # Save to Database (Persist Data)
    new_token = Token(
        name=name,
        reason=visit_reason,
        join_time=datetime.now().strftime("%H:%M:%S"),
        is_urgent=is_urgent,
        status="waiting"
    )
    db.session.add(new_token)
    db.session.commit()  # Commits change to queue.db file
    # Data for User
    response_data = {
        'id': new_token.id,
        'estimated_wait': wait_time
    }
    update_all_clients()
    return jsonify(response_data)
@app.route('/next_token', methods=['POST'])
def next_token():
    # Logic: Find someone "serving" and mark them "done"
    current_serving = Token.query.filter_by(status='serving').first()
    if current_serving:
        current_serving.status = 'done'
```

```
        db.session.commit()
    # Find next person
    # If priority exists, get them first. If not, get by ID (FIFO)
    urgent_next = Token.query.filter_by(status='waiting', is_urgent=True).order_by(Token.id).first()
    if urgent_next:
        next_person = urgent_next
        # assigning immediately token 0
    else:
        # assign according to priority list 5 mins from the last token
        next_person = Token.query.filter_by(status='waiting').order_by(Token.id).first()
    if next_person:
        next_person.status = 'serving'
        db.session.commit()
        # Convert DB object to Dictionary for JSON response
        person_dict = {
            'id': next_person.id,
            'name': next_person.name,
            'reason': next_person.reaso
}
        update_all_clients()
        return jsonify({'status': 'success', 'serving': person_dict})
    else:
        update_all_clients()
        return jsonify({'status': 'empty'})
```

```
# REAL-TIME UPDATES public announcement system
def update_all_clients():
    # Fetch fresh data from DB
    queue_count = Token.query.filter_by(status='waiting').count()
    serving_person = Token.query.filter_by(status='serving').first()
    # Get list of waiting people (Limit to top 10 for display)
    waiting_list_objs = Token.query.filter_by(status='waiting').order_by(Token.is_urgent.desc(),
Token.id).all()
    # Convert DB objects to simple list
    waiting_list = []
    for t in waiting_list_objs:
        waiting_list.append({'id': t.id, 'name': t.name, 'join_time': t.join_time, 'urgent': t.is_urgent})
    data = {
        'queue_length': queue_count,
        'current_serving': serving_person.id if serving_person else "None",
        'queue_list': waiting_list
    }
    socketio.emit('queue_update', data)
# data is the packet information that is been sent and emit means to broadcast
if __name__ == '__main__':
    socketio.run(app, debug=True, allow_unsafe_werkzeug=True)
```

Appendix B: Test Data and Test Cases

This appendix contains the test cases and sample input data used to verify the correctness of the system. Different scenarios such as adding tokens, serving tokens, displaying the queue, and handling an empty queue were tested to ensure proper functionality.



Test Cases →

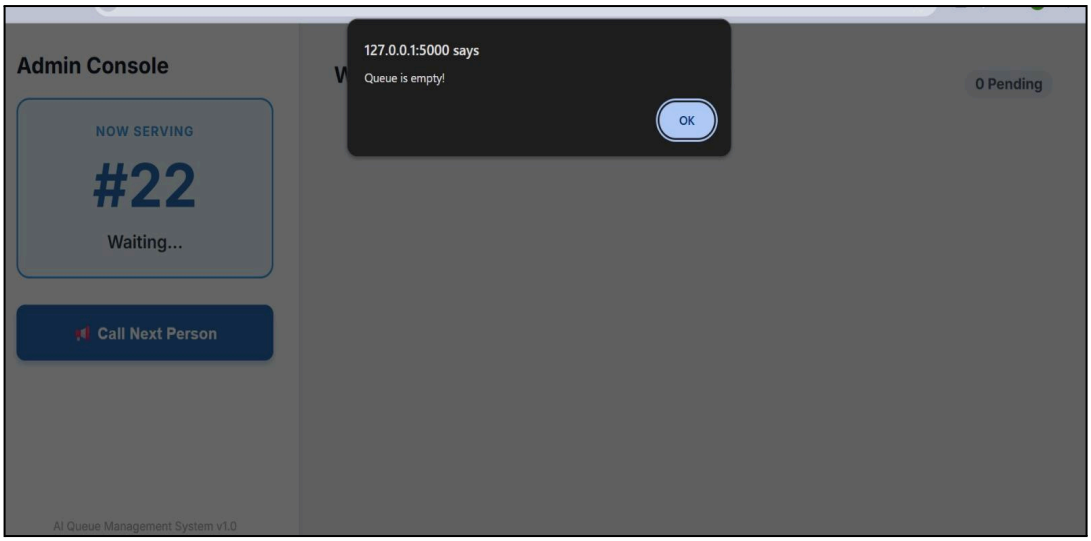
Test Case	Input Action	Queue State Before	Expected Output	Waiting Status	Result
1	Generate Token	Empty Queue	Token T1 Generated	0 Users Waiting	Pass
2	Generate Token	T1	Token T2 Generated	1 User Waiting	Pass
3	Serve Token	T1, T2	Token T1 Served	T2 Waiting	Pass
4	Display Queue	T2	Current Queue Displayed	Correct Status	Pass
5	Serve Empty Queue	Empty Queue	Error Message Displayed	No Waiting	Pass
5	Serve Empty Queue	Empty Queue	Error Message Displayed	No Waiting	Pass

Table B.1 Test Cases for Queue Management System

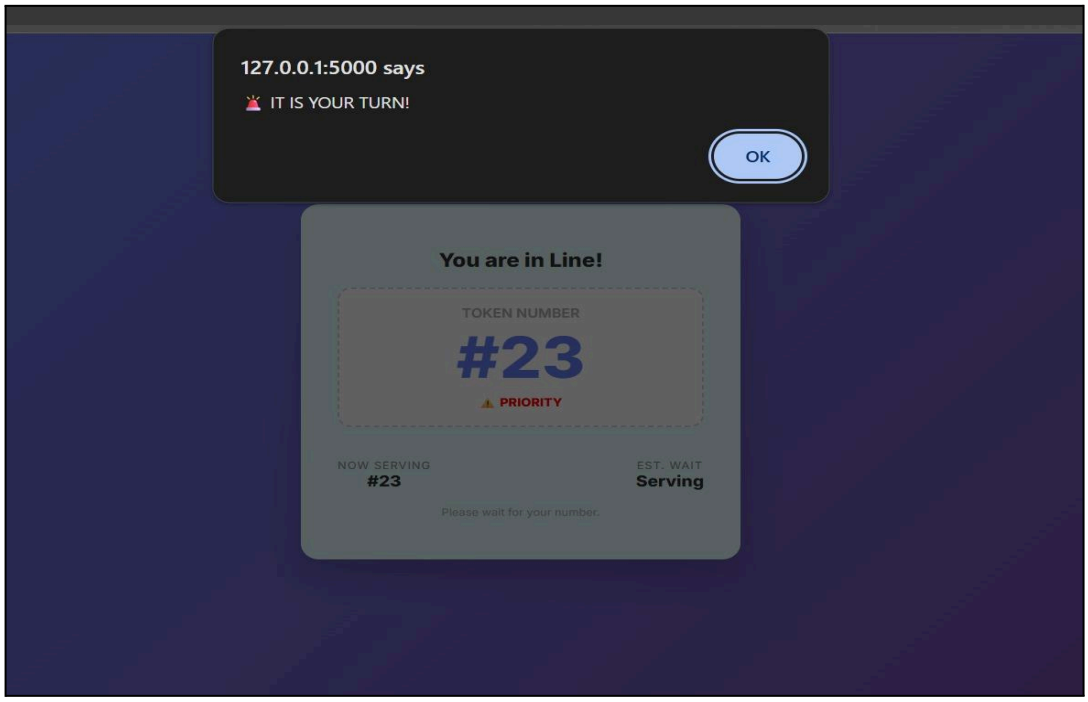
Appendix C: Output Screenshots

This appendix includes screenshots of the program output showing:

Error in token generation (Empty) →



Successful token generation →



Appendix D: Flowchart of Queue Management System

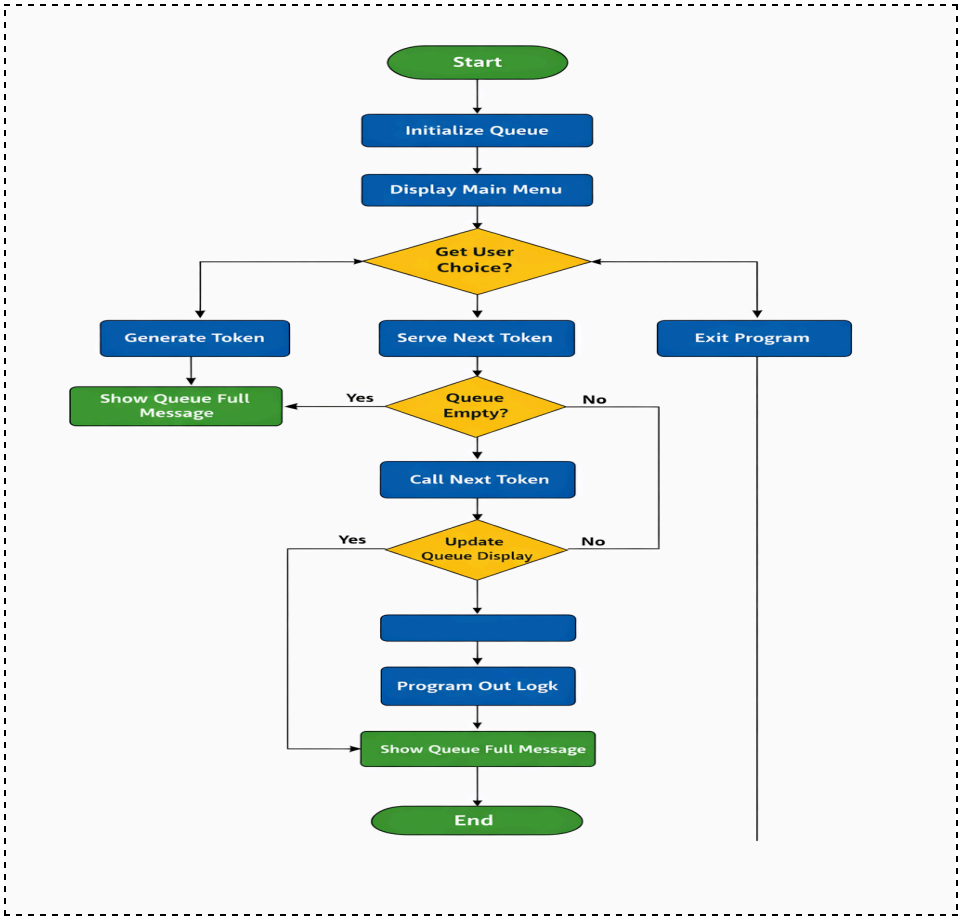


Figure D.1 Flowchart of Queue Management System

References

1. Silberschatz, A., Korth, H. F., and Sudarshan, S., *Database System Concepts*, 6th Edition, New York: McGraw-Hill, 2011.

2. Pressman, R. S., *Software Engineering: A Practitioner's Approach*, 7th Edition, New York: McGraw-Hill, 2010.

3. Sommerville, I., *Software Engineering*, 10th Edition, Boston: Pearson Education, 2016.

4. Fowler, M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd Edition, Boston: Addison-Wesley, 2004.

5. Booch, G., Rumbaugh, J., and Jacobson, I., *The Unified Modeling Language User Guide*, 2nd Edition, Boston: Addison-Wesley, 2005.

6. IEEE Computer Society, *IEEE Standard for Software Requirements Specifications*, IEEE Std 830-1998, IEEE Press, 1998.

7. Jain, R., *The Art of Computer Systems Performance Analysis*, New York: John Wiley & Sons, 1991.

8. Dennis, A., Wixom, B. H., and Tegarden, D., *Systems Analysis and Design with UML*, 5th Edition, Hoboken, NJ: John Wiley & Sons, 2012.

9. Kumar, A., and Sharma, R., "Design and Implementation of Digital Queue Management System," *International Journal of Computer Applications*, Vol. 178, No. 7, pp. 25–30, 2019.

10. Patil, S., and Deshmukh, P., "Web-Based Queue Management System," *Proceedings of the International Conference on Emerging Trends in Engineering and Technology*, pp. 112–116, Pune, India, 2018.

11. Oracle Corporation, *MySQL 8.0 Reference Manual*, Oracle Documentation, 2022.

12. Google Developers, "SMS and Notification Services Using APIs," https://developers.google.com, accessed March 2024.

13. W3C, “HTML5 Specification,” World Wide Web Consortium, https://www.w3.org/TR/html5/, accessed March 2024.

14. Mozilla Developer Network, “Web Application Architecture,” https://developer.mozilla.org, accessed March 2024.

15. Kendall, K. E., and Kendall, J. E., *Systems Analysis and Design*, 9th Edition, Upper Saddle River, NJ: Pearson Education, 2014.
16. Larson, R. C., “Perspectives on Queues: Social Justice and the Psychology of Queueing,” *Operations Research*, Vol. 35, No. 6, pp. 895–905, 1987.

17. Gross, D., Shortle, J. F., Thompson, J. M., and Harris, C. M., “Queueing Theory and Modeling,” *Queueing Systems*, Vol. 44, No. 1, pp. 1–20, 2003.

18. Alotaibi, Y., and Alzahrani, A., “Design and Implementation of an Intelligent Queue Management System,” *International Journal of Advanced Computer Science and Applications*, Vol. 8, No. 9, pp. 234–240, 2017.

19. Rahman, M., and Hossain, M., “Web-Based Automated Queue Management System,” *International Journal of Scientific and Engineering Research*, Vol. 9, No. 5, pp. 1021–1026, 2018.

20. Kumar, N., and Verma, S., “Smart Queue Management System Using Web Technologies,” *International Journal of Engineering Research and Technology (IJERT)*, Vol. 7, No. 6, pp. 415–420, 2018.