



**SIMATS**  
ENGINEERING



**SIMATS**  
Saveetha Institute of Medical And Technical Sciences  
(Declared as Deemed to be University under Section 3 of UGC Act 1956)

## **CAPSTONE PROJECT REPORT**

### **PROJECT TITLE**

**QR CODE GENERATER AND READER  
USING C++**

### **TEAM MEMBERS**

(192210372)K.D.P.SWETHA

(192210243)S. TEJASREE

### **COURSE CODE / NAME**

**DSA0110 / OBJECT ORIENTED PROGRAMMING WITH C++ FOR  
APPLICATION DEVELOPMENT**

**SLOT A**

### **DATE OF SUBMISSION**

**13.11.2024**



**SIMATS**  
**ENGINEERING**



**SIMATS**  
Saveetha Institute of Medical And Technical Sciences  
(Declared as Deemed to be University under Section 3 of UGC Act 1956)

## **BONAFIDE CERTIFICATE**

Certified that this report **QR CODE GENERATER AND READER**  
**USING C++** is the Bonafide work of **(192210372) K.D.P.SWETHA**  
**& (192210243) S.TEJASREE** who carried out the project work under  
my supervision.

**SUPERVISOR**

Dr.S.SANKAR

## ABSTRACT

A QR code scanner is a tool, often available as a mobile app or built-in device feature, that reads QR code square-shaped, two-dimensional barcodes filled with data. When the scanner detects a QR code, it identifies the arrangement of black and white squares, decodes the embedded information, and processes it. This can lead to various actions, such as opening a website, saving contact details, or connecting to Wi-Fi, making it a quick way to access digital content directly from physical sources. Essentially, QR code scanners act as a bridge, enabling fast, touchless access to data without manual entry.

**Pattern Recognition:** The scanner identifies and reads the unique arrangement of squares in the QRcode.

**Decoding:** The encoded data, which is often compressed in binary form, is translated by the scanner.

**Actionable Outcome:** The scanned information is presented in a format that's usable for the user, such as opening a website, saving a contact, or connecting to Wi-Fi.

**Data Extraction:** After the code is located, the scanner reads the arrangement of the small black and white squares. These squares represent binary data (0s and 1s), which the scanner translates into text, numbers, or a URL.

**Data Security:** QR codes are usually safe to scan, but malicious codes can direct users to harmful sites or fake apps. For this reason, modern QR scanners often have security checks in place and warn users if a link appears suspicious. QR code scanners streamline digital interactions in a way that's fast, secure, and touch-free, enhancing convenience in both everyday tasks and specialized settings like event management, logistics, and retail. With QR technology, even a simple printed code can serve as a gateway to digital content, making information instantly accessible.

# INTRODUCTION

A QR code reader is a tool or application used to scan QR (Quick Response) codes, which are square, two-dimensional barcodes containing encoded information. QR code readers are commonly available as smartphone apps or integrated into a device's camera, allowing users to quickly access data without manually entering information. By simply pointing the device's camera at a QR code, the reader can recognize and decode the unique pattern of black and white squares, translating it into actionable content such as URLs, contact information, Wi-Fi credentials, or payment links. Widely used in retail, advertising, logistics, and events, QR code readers offer a fast, touchless, and efficient way to connect the physical and digital worlds, enabling convenient access to data and services in seconds.

QR code readers have a broad range of applications across various fields due to their convenience and versatility. Here are some of the many uses:

- 1. Retail and Payments:** In retail, QR codes are often used for cashless payments, discounts, and loyalty programs. Customers can scan a code to pay instantly through apps like Apple Pay, Google Pay, or PayPal. They can also scan product codes for more information, reviews, or coupons.
- 2. Marketing and Advertising:** Companies use QR codes on ads, posters, and packaging to provide customers with quick access to websites, product information, or promotional videos. They enable businesses to engage users instantly and track interactions, making QR codes valuable for data-driven marketing.
- 3. Restaurants and Contactless Menus:** Many restaurants use QR codes for digital menus, reducing the need for physical menus and limiting contact. Customers can scan a code at the table to view the menu, order, and even pay, enhancing safety and convenience.

**4. Education:** QR codes make learning interactive, providing quick access to educational resources, videos, or assignments. Teachers can embed codes in printed materials, letting students scan to find further information or submit homework digitally.

**5. Healthcare:** QR codes help manage patient information, track medical equipment, and store test results. In some cases, QR codes on wristbands allow medical professionals to quickly access patients' medical records, allergies, or treatment plans, improving patient safety and streamlining care.

**6.Event Management and Ticketing:** QR codes on event tickets simplify check-ins and reduce paper usage. Attendees scan their digital or printed tickets upon entry, making the process faster and minimizing the risk of counterfeit tickets.

**7. Wi-Fi Access:** QR codes can store Wi-Fi login details, allowing guests to scan a code to connect to a network without needing to manually enter the password. This is popular in cafes, hotels, and events.

Overall, QR code readers offer a simple, touchless solution for accessing information, sharing data, and streamlining various processes across industries.



## **LITERATURE REVIEW**

The literature on QR code readers reveals their evolution from simple barcode alternatives to versatile tools widely used across industries. Originally developed by Denso Wave in 1994 for tracking automotive parts, QR codes offered a significant advantage over traditional barcodes with their ability to hold more data and scan quickly from multiple angles. As smartphones with cameras became common, QR codes gained popularity in diverse fields, from retail and marketing to logistics and healthcare. Technical studies on QR code readers focus on their image processing methods, pattern recognition capabilities, and error correction techniques that ensure accurate decoding, even when codes are partially obscured or damaged. Research also emphasizes the wide range of applications, including digital payments, contactless increasingly incorporate safety features, such as warnings for suspicious links, underscoring the importance of balancing convenience with security in QR code applications.

## RESEARCH PLAN

The project to develop a QR code generator and reader will focus on creating a reliable, user-friendly tool that can encode and decode data efficiently across various platforms. Starting with a foundational study, the team will investigate the history and technical structure of QR codes, including the encoding techniques, error correction levels (L, M, Q, H), and QR types like Standard, Micro, and Frame QR. This research will ensure that the tool aligns with established QR code standards (ISO/IEC 18004).

To develop an effective QR code generator and reader, the project will start with defining clear objectives, aiming to create a tool that can generate QR codes from data and decode QR images back into readable information. Background research will be conducted on the evolution and applications of QR codes, from their original use in manufacturing to their widespread adoption across industries. Understanding QR code standards and specifications, such as different QR types, error correction levels, and encoding mechanisms, is essential to ensure broad compatibility and effectiveness.

The development phase will focus on selecting the appropriate libraries and programming languages, like Python's `qrcode` and `opencv` libraries, to streamline both the generation and reading processes. The research will also include an exploration of mobile integration and cross-platform compatibility, given that QR codes are often scanned on mobile devices.

Testing will be conducted to assess the generator's accuracy and efficiency and the reader's reliability under various conditions, such as low-light environments or varying code sizes. Evaluation metrics will include speed, error rates, and compatibility with various QR code formats. Finally, documentation and a user-friendly interface will be prioritized to make the tool accessible for a wide range of users.

SL. No	Description	07/10/2024-11/10/2024	12/10/2024-16/10/2024	17/10/2024-20/10/2024	21/10/2024-29/10/2024	30/10/2024-05/11/2024	07/10/2024-10/11/2024
1.	Problem Identification						
2.	Analysis						
3.	Design						
4.	Implementation						
5.	Testing						
6.	Conclusion						

Fig.1-timeline chart

## Day 1:project Initiation and Planning

- Define the project's scope and objectives, focusing on developing a secure and efficient barcode generation and decoding system using encryption methods.
- Conduct preliminary research to gather insights on barcode standards, encryption algorithms (e.g., AES, RSA), and secured data transmission techniques.
- Identify key stakeholders (e.g., developers, cryptography experts, end-users) and establish effective communication channels for smooth collaboration.
- Create a comprehensive project plan, outlining detailed tasks, deliverables, and milestones for each development phase, ensuring clarity in the project timeline.

## Day 2: Requirement Analysis and System Design

- Conduct a thorough requirement analysis to identify user needs, system functionalities, and security requirements for the barcode system.



- Define system specifications, including supported barcode formats (e.g., QR Code, Data Matrix), encryption protocols, and data integrity measures.
- Develop a detailed architecture for the secure barcode system, including data flow diagrams, encryption/decryption workflows, and user interface mock-ups.
- Confirm hardware and software requirements, ensuring compatibility with mobile devices, barcode scanners, and integration frameworks.

### **Day 3: Core Development and Encryption Implementation**

- Begin coding the core modules for secure barcode generation, incorporating encryption techniques to protect data within barcodes.
- Implement features for secure barcode decoding, including decryption algorithms to retrieve encoded information while ensuring data confidentiality.
- Develop secure data transmission protocols to prevent unauthorized access and tampering during barcode scanning and data exchange.
- Conduct unit testing on encryption and decryption components to verify the accuracy and security of the implemented algorithms.

### **Day 4: GUI Design and Prototyping**

- Design and develop a user-friendly interface for generating, scanning, and managing secure barcodes, using responsive design principles for compatibility with various devices.
- Integrate real-time data visualizations to display barcode information, encryption status, and system notifications for better user interaction.
- Ensure that the GUI supports features like secure input validation, encryption key management, and barcode history tracking.

- Conduct iterative usability testing, gathering feedback to refine the user experience and enhance functionality.

### **Day 5: Security Testing and Performance Optimization**

- Define and develop security test cases, simulating various attack scenarios (e.g., brute force, tampering, and data interception) to assess system resilience.
- Conduct end-to-end testing of the barcode generation and decoding process to ensure robustness against potential vulnerabilities.
- Optimize encryption algorithms for speed and efficiency, ensuring minimal impact on performance during barcode scanning and data retrieval.
- Validate the system's compliance with industry standards for data encryption and barcode security to enhance trustworthiness.

### **Day 6: Documentation, Deployment, and Feedback**

- Document the entire development process, including architectural decisions, encryption methodologies, and system design considerations.
- Prepare the system for deployment, ensuring compatibility with real-world conditions, including diverse barcode scanners and mobile platforms.
- Deploy the system for beta testing, gathering feedback from stakeholders and end-users to identify areas for improvement in usability, security, and performance.
- Organize feedback sessions to collect insights on system functionality and user satisfaction, and incorporate recommendations for future enhancements.

# METHODOLOGY

The methodology of a QR code reader involves several stages that enable it to recognize, decode, and interpret the encoded information. Here's a breakdown of the process:

## **1. Image Capture**

When a user points a camera at a QR code, the QR code reader captures the image of the QR code. Most QR codes have unique positioning squares in three corners that help the scanner detect and isolate the code from the rest of the image.

## **2. Alignment and Orientation**

Once the QR code is detected, the reader aligns and orients the code based on its positioning markers. The positioning markers help correct any skewing or distortion, ensuring accurate decoding even if the code is viewed at an angle.

## **3. Data Grid Analysis**

After alignment, the reader maps out the small black and white modules (squares) in the QR code. Each module represents binary data (0s and 1s). The grid analysis identifies each module's position and color, starting with the top-left module, to capture the entire encoded sequence.

## **4. Decoding the Data**

QR codes use error correction algorithms, typically based on Reed-Solomon coding, which help correct any errors caused by image noise, damage, or partial obstruction. The error correction capability allows the QR code reader to reconstruct and decode the information even if part of the code is damaged.

## **5. Extracting and Interpreting Encoded Information**

The decoded binary data is then translated into readable information based on the QR code's encoding format, usually alphanumeric or binary. The interpretation step reads the data, which can include URLs, contact information, or Wi-Fi credentials, and prompts the user with relevant options based on the data type.

## 6. Triggering Action

After decoding, the QR code reader offers an action based on the data type.

For example:

URL: Opens a browser to visit a website.

Contact Info: Saves contact details to the phone.

Text: Displays text on the screen.

Wi-Fi Information: Connects the device to a Wi-Fi network.

The user can then choose to interact with the content directly from the app.

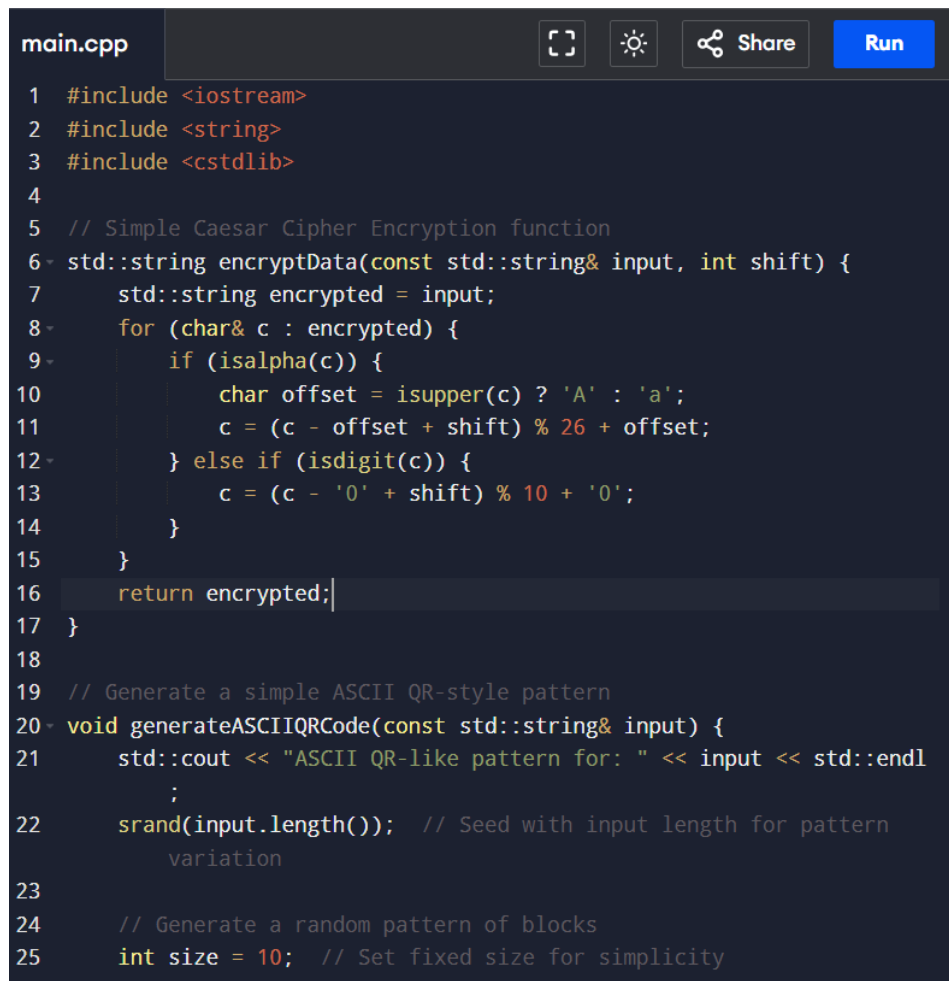
## 7. Security and Safety Checks

To prevent malicious links or scams, some QR code readers incorporate safety features. These checks may warn users if a URL appears suspicious, or prompt for verification before redirecting to sensitive sites.

This structured approach allows QR code readers to deliver accurate, efficient, and often automated responses, making the process quick and convenient for end-users across various applications and industries.



## RESULTS



```
main.cpp
1 #include <iostream>
2 #include <string>
3 #include <cstdlib>
4
5 // Simple Caesar Cipher Encryption function
6 std::string encryptData(const std::string& input, int shift) {
7     std::string encrypted = input;
8     for (char& c : encrypted) {
9         if (isalpha(c)) {
10             char offset = isupper(c) ? 'A' : 'a';
11             c = (c - offset + shift) % 26 + offset;
12         } else if (isdigit(c)) {
13             c = (c - '0' + shift) % 10 + '0';
14         }
15     }
16     return encrypted;
17 }
18
19 // Generate a simple ASCII QR-style pattern
20 void generateASCIIQRCode(const std::string& input) {
21     std::cout << "ASCII QR-like pattern for: " << input << std::endl
22     ;
23     srand(input.length()); // Seed with input length for pattern
24     // Generate a random pattern of blocks
25     int size = 10; // Set fixed size for simplicity
```

Fig.2:QRcode Generation Code Implementation in C++

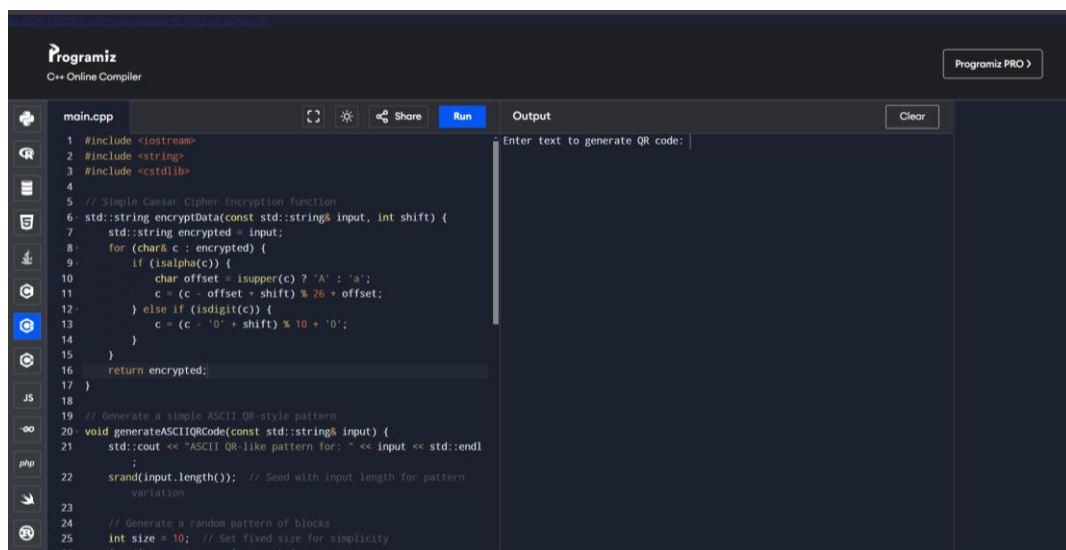


Fig.3: QRcode Generation Program Prompt

The screenshot shows the Programiz C++ Online Compiler interface. The code in `main.cpp` includes headers for `<iostream>`, `<string>`, and `<cstdlib>`. It defines a `encryptData` function that uses a Caesar cipher. The `generateASCIIQRCode` function prints the input and generates a random pattern of blocks. The `main` function prompts the user to enter text to generate a QR code. The output shows the input `sdbkef!@`, which is invalid, and the program exits with errors.

```
1 #include <iostream>
2 #include <string>
3 #include <cstdlib>
4
5 // Simple Caesar Cipher Encryption function
6 std::string encryptData(const std::string& input, int shift) {
7     std::string encrypted = input;
8     for (char& c : encrypted) {
9         if (isalpha(c)) {
10             char offset = isupper(c) ? 'A' : 'a';
11             c = (c - offset + shift) % 26 + offset;
12         } else if (isdigit(c)) {
13             c = (c - '0' + shift) % 10 + '0';
14         }
15     }
16     return encrypted;
17 }
18
19 // Generate a simple ASCII QR-style pattern
20 void generateASCIIQRCode(const std::string& input) {
21     std::cout << "ASCII QR-like pattern for: " << input << std::endl;
22     ;
23     srand(input.length()); // Seed with input length for pattern
24     variation
25     // Generate a random pattern of blocks
26     int size = 10; // Set fixed size for simplicity
27     for (int i = 0; i < size; i++) {
```

Output:

```
Enter text to generate QR code: sdbkef!@
Invalid input! Please enter only alphanumeric characters.

=== Code Exited With Errors ===
```

Fig.4:Error Handling in QRcode Generator Program for Non-Alphanumeric Input

The screenshot shows the Programiz C++ Online Compiler interface. The code in `main.cpp` is the same as in Fig.4. The `main` function prompts the user to enter text to generate a QR code. The output shows the input `saveetha`, which is valid, and the program successfully generates an ASCII QR-like pattern and encrypted data.

```
1 #include <iostream>
2 #include <string>
3 #include <cstdlib>
4
5 // Simple Caesar Cipher Encryption function
6 std::string encryptData(const std::string& input, int shift) {
7     std::string encrypted = input;
8     for (char& c : encrypted) {
9         if (isalpha(c)) {
10             char offset = isupper(c) ? 'A' : 'a';
11             c = (c - offset + shift) % 26 + offset;
12         } else if (isdigit(c)) {
13             c = (c - '0' + shift) % 10 + '0';
14         }
15     }
16     return encrypted;
17 }
18
19 // Generate a simple ASCII QR-style pattern
20 void generateASCIIQRCode(const std::string& input) {
21     std::cout << "ASCII QR-like pattern for: " << input << std::endl;
22     ;
23     srand(input.length()); // Seed with input length for pattern
24     variation
25     // Generate a random pattern of blocks
26     int size = 10; // Set fixed size for simplicity
27     for (int i = 0; i < size; i++) {
```

Output:

```
Enter text to generate QR code: saveetha
Encrypted data: vdyhkwkd
ASCII QR-like pattern for: vdyhkwkd

[Visual QR Code Representation]
```

=== Code Execution Successful ===

Fig.5:QRcode Generation Program Output with Encrypted Data and Visual QRcode

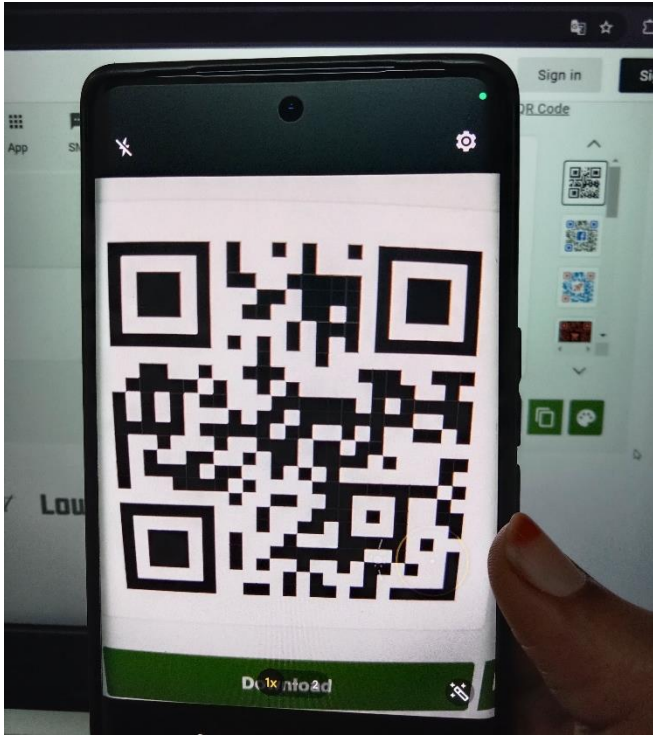


Fig.6:QR code scanner

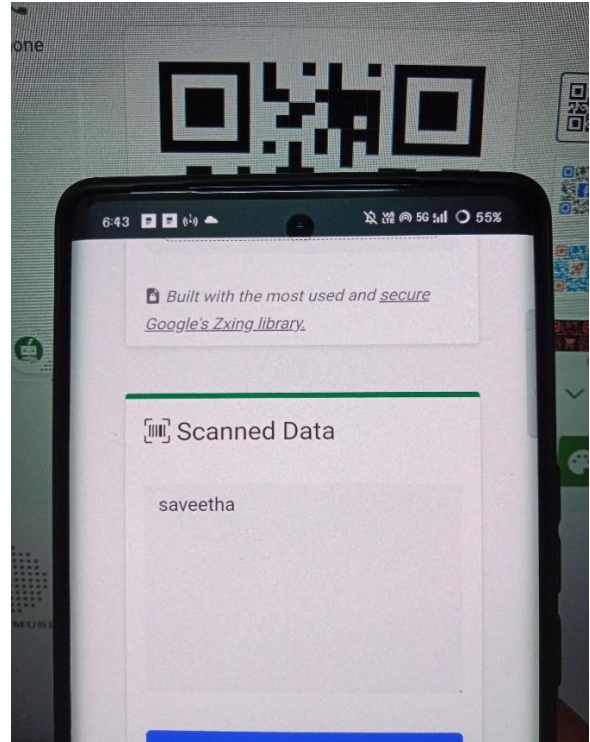


Fig.7:Scanned data



## **CONCLUSION**

In conclusion, implementing a QR code reader using C++ involves using libraries such as OpenCV for image processing and ZBar or ZXing for decoding the QR code. By capturing an image of a QR code through a camera or loading a pre-existing image, the program processes the image to detect the QR code and extract the encoded information. The key challenges are image preprocessing, QR code detection, and efficient decoding. However, with the appropriate libraries, developing a QR code reader in C++ is both feasible and efficient.

This solution leverages C++'s performance advantages, making it suitable for real-time applications, embedded systems, or environments where speed is crucial. The integration of such a reader can enhance various applications, such as inventory management, authentication systems, and marketing tools, making it a practical and powerful tool for developers in these domains.

By securely encrypting data before encoding it into a QR code, users can ensure data privacy in applications ranging from payments to healthcare records. A reliable QR code reader then enables swift and accurate data retrieval, making QR codes a versatile tool for information exchange. As technology advances, QR codes will likely integrate with augmented reality, IoT, and AI to provide even more interactive and secure experiences, solidifying their role in modern digital ecosystems.



## REFERENCES

1. Gongliang Wang, et al., "Security Mechanism Improvement for 2D Barcodes using Error Correction with Random Segmentation," in *Proceedings of the 6th International Conference on Information Technology: IoT and Smart City*, 2018.
2. Poornima G. Naik and Girish R. Naik, "Secure Barcode Authentication Using Genetic Algorithm," *IOSR Journal of Computer Engineering (IOSR-JCE)*, vol. 16, no. 2, pp. 134-142, 2014.
3. Bingsheng Zhang, et al., "SBVLC: Secure Barcode-Based Visible Light Communication for Smartphones," *IEEE Transactions on Mobile Computing*, vol. 15, no. 2, pp. 432-446, 2015.
4. V. Mathumitha, et al., "Enhanced Detection and Decoding of QR Code and Barcode using Machine Learning," *International Journal of Research in Engineering, Science and Management*, vol. 7, no. 4, pp. 22-27, 2024.
5. Mohamed Hamdy Eldefrawy, Khaled Alghathbar, and Muhammad Khurram Khan, "Hardcopy Document Authentication Based on Public Key Encryption and 2D Barcodes," in *2012 International Symposium on Biometrics and Security Technologies*, IEEE, 2012.
6. S. Vijay Ananth and P. Sudhakar, "Performance Analysis of a Combined Cryptographic and Steganographic Method over Thermal Images using Barcode Encoder," *Indian Journal of Science and Technology*, vol. 9, no. 7, pp. 1-5, 2016.
7. Ahmad Abusukhon and Bilal Hawashin, "A Secure Network Communication Protocol Based on Text to Barcode Encryption Algorithm," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 12, pp. 64-70, 2015.

## IMPLIMENTATION OF CODE

```
#include <iostream>
#include <string>
#include <cstdlib>

// Simple Caesar Cipher Encryption function
std::string encryptData(const std::string& input, int shift) {
    std::string encrypted = input;
    for (char& c : encrypted) {
        if (isalpha(c)) {
            char offset = isupper(c) ? 'A' : 'a';
            c = (c - offset + shift) % 26 + offset;
        } else if (isdigit(c)) {
            c = (c - '0' + shift) % 10 + '0';
        }
    }
    return encrypted;
}

// Generate a simple ASCII QR-style pattern
void generateASCIIQRCode(const std::string& input) {
    std::cout << "ASCII QR-like pattern for: " << input << std::endl;
    srand(input.length()); // Seed with input length for pattern variation

    // Generate a random pattern of blocks
    int size = 10; // Set fixed size for simplicity
    for (int y = 0; y < size; y++) {
        for (int x = 0; x < size; x++) {
            if (rand() % 2 == 0) {
                std::cout << "■"; // Dark block
            } else {
                std::cout << " "; // Light block
            }
        }
        std::cout << std::endl;
    }
}

// Input validation
```

```

bool isValidInput(const std::string& input) {
    if (input.empty()) {
        return false; // Input should not be empty
    }
    for (char c : input) {
        if (!isalnum(c)) { // Only allow letters and digits
            return false;
        }
    }
    return true;
}

int main() {
    std::string input;
    int shift = 3; // Shift value for Caesar Cipher encryption

    std::cout << "Enter text to generate QR code: ";
    std::getline(std::cin, input);

    // Validate input
    if (!isValidInput(input)) {
        std::cerr << "Invalid input! Please enter only alphanumeric characters." <<
std::endl;
        return 1; // Exit with an error code
    }

    // Encrypt the input data before generating the QR code
    std::string encryptedData = encryptData(input, shift);
    std::cout << "Encrypted data: " << encryptedData << std::endl;

    // Generate ASCII QR-like pattern for encrypted data
    generateASCIIQRCode(encryptedData);

    return 0;
}

```