**Student Cafeteria Attendance System Documentation**

**Version:** 1.0.0 **Date:** [Insert Date] **Author:** [Your Name] **Maintainer:** [Your Name or Team]

**Table of Contents**

1. Introduction
2. Project Overview  2.1. Purpose  2.2. Scope
3. System Architecture  3.1. Module Descriptions   3.1.1. Database Handler   3.1.2. QR Code Handler   3.1.3. Record Handler
4. Requirements  4.1. Hardware and Software Requirements  4.2. External Libraries and Dependencies
5. Setup and Installation Instructions
6. Usage Instructions
7. Code Documentation and Commenting Guidelines
8. Testing and Validation
9. Error Handling and Troubleshooting
10. Future Enhancements and Roadmap
11. Version Control and Maintenance
12. Acknowledgements and References

**1. Introduction**

This document is intended to provide a full overview and guide of the Student Cafeteria Attendance System. It records the students’ visits to the cafeteria through a system that handles student records, generates and decodes QR codes, and manages attendance logs. The information provided here is targeted at developers and maintainers who will handle future modifications, debugging, and enhancements to the project.

**2. Project Overview**

**2.1. Purpose**

The primary purpose of the project is to efficiently manage and track student visits to the cafeteria. It includes:

* **Student Records Management:** Adding, deleting, and maintaining student information.
* **QR Code Handling:** Generating unique QR codes for students and decoding them upon cafeteria entry.
* **Attendance Logging:** Creating, updating, and delivering attendance records via email.

**2.2. Scope**

The system encompasses a complete workflow from student registration to attendance reporting. It is designed to scale and be modular, enabling easy integration of future features or adaptations to different environments.

**3. System Architecture**

**3.1. Module Descriptions**

**3.1.1. Database Handler**

**Purpose:** Manages the student records database, which includes functions to add, delete, and update student profiles.

**Key Functions:**

* addStudent(std::string studentId, std::string studentName)**:** Inserts a new record into the database.
* deleteStudent(std::string studentId)**:** Removes a student record from the database.

**Design Considerations:**

* **Data Integrity:** Enforces uniqueness and validity of student records.
* **Storage Backend:** Utilizes SQL or a file-based system, with clear error handling for duplicate or missing entries.

**3.1.2. QR Code Handler**

**Purpose:** Generates and decodes QR codes which represent student identification information.

**Key Functions:**

* generateQRCode(StudentData data)**:** Encodes the student’s information into a QR code image file.
* decodeQRCode(Image qrImage)**:** Reads the QR code image to extract student identification data upon cafeteria entry.

**Design Considerations:**

* **Libraries:** Uses a third-party library such as ZXing or a QR code generator.
* **Performance:** Quick encoding and decoding with built-in error correction.
* **Scalability:** Ability to update the algorithm or data format without affecting other modules.

**3.1.3. Record Handler**

**Purpose:** Handles the creation, modification, and dispatching of attendance records based on student visits.

**Key Functions:**

* createRecord(StudentData data, Time timestamp, MealType meal)**:** Logs an attendance entry when a student scans their QR code.
* modifyRecord(RecordID recordId, RecordUpdate update)**:** Allows corrections on a previously recorded entry.
* emailRecord(Record record, EmailAddress recipient)**:** Prepares and sends the attendance record via email using SMTP (commonly through libcurl).

**Design Considerations:**

* **Record Format:** Each record includes date, time, student ID, and meal type.
* **Email Integration:** Secure handling of SMTP credentials and error reporting if there is a failure in dispatching the email.
* **Data Preservation:** Logs and backups to avoid data loss.

**4. Requirements**

**4.1. Hardware and Software Requirements**

* **Operating System:** Cross-platform (Windows, Linux, macOS).
* **Compiler:** Any modern C++ compiler supporting C++17 or later (e.g., GCC 7.3+, Clang 6+, MSVC 2017+).
* **Memory:** Adequate RAM for desktop or server execution.

**4.2. External Libraries and Dependencies**

* **Database Library:** SQLite, MySQL Connector/C++ or another chosen backend.
* **QR Code Library:** ZXing-C++, QR Code generator by Nayuki, or similar.
* **SMTP Library:** libcurl or another SMTP utility for email dispatch.
* **Filesystem and Utility Libraries:** C++17 <filesystem> for path management and file handling.
* **Documentation Generator:** Doxygen for code-level documentation.

**5. Setup and Installation Instructions**

1. **Clone the Repository:**

bash

git clone https://your.repo.url/your-project.git

cd your-project

1. **Install Dependencies:**
   * Ensure that your system has the required libraries and compilers installed.
   * Follow any specific library instructions (e.g., install ZXing via your package manager or build from source).
2. **Build the Project:**
   * Using CMake:

bash

mkdir build && cd build

cmake ..

make

* + Alternatively, use the provided Makefile, if available.

1. **Configure Environment:**
   * Edit the configuration file to set up paths, database credentials, and SMTP email settings.
   * Ensure that the environment variables required by the application are set.

**6. Usage Instructions**

* **Launching the Application:** Run the executable from the command line:

bash

./cafeteria\_attendance\_system

* **User Operations:**
  + **Database Operations:** Use provided command-line options or a GUI to add or remove student records.
  + **QR Code Generation/Decoding:** Follow on-screen prompts or commands to generate and scan QR codes.
  + **Attendance Logging:** Ensure that students scan their QR codes on entry; the system automatically logs entries.
  + **Email Dispatch:** The system periodically sends attendance records via configured email settings. Logs provide feedback if errors occur.
* **Command-line Help:** Execute:

bash

./cafeteria\_attendance\_system --help

This displays options and usage instructions.

**7. Code Documentation and Commenting Guidelines**

* **Doxygen Comments:** All functions, classes, and modules should be documented using Doxygen. For example:

cpp

/\*\*

\* @brief Adds a student to the database.

\*

\* @param studentId The unique identifier for the student.

\* @param studentName The full name of the student.

\* @return true if the student was added successfully, false otherwise.

\*/

bool addStudent(const std::string &studentId, const std::string &studentName);

* **Coding Conventions:** Follow [insert your coding style guidelines here]. Consistency in naming, spacing, and file organization is crucial.
* **Documentation Generation:** Run Doxygen (via the provided Doxyfile) to generate HTML/PDF documentation:

bash

doxygen Doxyfile

**8. Testing and Validation**

* **Unit Tests:** A suite of unit tests is provided in the /tests folder. Use your testing framework (e.g., GoogleTest) for running tests:

bash

cd tests

./run\_tests

* **Integration Tests:** Validate each module’s interaction, especially the email dispatch and QR code decoding functionalities.
* **Continuous Integration:** Integrate tests into your CI/CD pipeline for automated builds and validations.

**9. Error Handling and Troubleshooting**

* **Error Reporting:** All modules employ both exception handling (try/catch) and error code checking, particularly for filesystem and email operations.
* **Common Issues:**
  + *Database Connection Failures*: Check configuration file for correct credentials and database file location.
  + *QR Code Read Errors*: Ensure camera or scanning device drivers are correctly installed and configured.
  + *Email Dispatch Failures*: Verify SMTP settings and network connectivity; errors are logged to logs/email\_errors.log.
* **Logs:** Logs are written to the /logs directory. Review these logs for runtime errors and status information.

**10. Future Enhancements and Roadmap**

* **Feature Enhancements:**
  + Improve the GUI for easier operation.
  + Add biometric authentication for enhanced security.
  + Provide support for additional reporting formats.
* **Maintenance Roadmap:**
  + Regular updates for dependencies.
  + Periodic security audits and code refactoring.
  + Expand unit test coverage and update CI/CD practices.

**11. Version Control and Maintenance**

* **Repository Information:** The source is maintained under Git. Please follow branch management and commit message conventions described in CONTRIBUTING.md.
* **Release Notes:** Maintain an updated CHANGELOG.md with detailed notes for every significant change.
* **Developer Communication:** For major changes, use issue tracking systems (e.g., GitHub Issues) and pull request reviews to communicate and discuss code modifications.

**12. Acknowledgements and References**

* **Libraries:**
  + ZXing-C++ (for QR code handling)
  + libcurl (for SMTP email dispatch)
  + SQLite or another chosen database engine
* **Documentation Tools:**
  + Doxygen
* **Additional References:**
  + C++17 <filesystem> documentation
  + Relevant tutorials and technical articles used during development

**Final Notes**

This document serves as a comprehensive reference manual to assist in the deployment, debugging, and future enhancement of the Student Cafeteria Attendance System. Please update this documentation as new features are added or modifications are made to keep it current.

By following the instructions above, you’ll have your project documentation in a .docx file ready to be handed over to the next programmer for maintenance and further development.

If you need any further help with the formatting or additional customization of the document, let me know!

The database is entirely handled without external databases.