## 4.1 Basic Modules

The QR Share project can be divided into several key modules, each responsible for specific functionalities. Below, we'll describe these modules and their functionalities in detail:

1. User Authentication Module

Description: This module handles user registration, login, and authentication processes. It ensures that only authorized users can access the application's features.

Functionality:

* User Registration: Allows users to create new accounts by providing necessary details like username, email, and password.
* User Login: Enables registered users to log in securely with their credentials.
* Authentication: Verifies user identity and grants access to the app's features.
* Password Recovery: Provides a mechanism for users to reset their passwords if forgotten.

1. QR Code Generation Module

Description: This module is responsible for generating customized QR codes with various data types, catering to users' specific needs and preferences.

Functionality:

* Data Input: Accepts different data types, including text, URLs, contact information, and more.
* QR Code Generation: Converts input data into QR codes using QR code generation algorithms.
* Storage: Manages the storage of generated QR codes for user access.

1. QR Code Scanning Module

Description: This module facilitates the scanning of QR codes using device cameras. It ensures accurate and efficient scanning to retrieve shared digital resources.

Functionality:

* Camera Integration: Utilizes the device's camera for QR code scanning.
* Scanning Algorithm: Implements algorithms to detect and decode QR codes.
* Data Retrieval: Extracts information from scanned QR codes.

1. Data Transfer Module

Description: This module offers flexible data transfer options, allowing users to choose various methods for sharing decoded data, including messaging apps and email.

Functionality:

* Data Sharing: Enables users to send the decoded data to other users or platforms.
* Messaging Apps: Integrates with messaging apps for quick sharing.
* Email Integration: Allows users to send data via email.
* Local Storage: Provides the option to save data locally on the device.

1. Data Security Module

Description: This module ensures the secure handling and transmission of user data and generated QR codes. It adheres to relevant data protection regulations, fostering user trust and confidence.

Functionality:

* Encryption: Encrypts user data and QR codes to prevent unauthorized access.
* Privacy Compliance: Ensures compliance with data protection laws and regulations.
* Secure Storage: Safeguards user data stored within the app.
* Access Controls: Manages user permissions and access levels.

1. History Tracking Module

Description: This module incorporates a history feature that allows users to manage and conveniently reference previously generated and scanned QR codes.

Functionality:

* History Storage: Stores a record of generated and scanned QR codes.
* Search and Retrieval: Enables users to search for specific codes.
* Organization: Organizes the history for easy access and management.
* Data Reference: Allows users to re-scan or share QR codes from the history.

Following the divide and conquer approach, these modules can be developed separately and then integrated to create the complete QR Share system. Each module serves a specific purpose and contributes to the overall functionality and user experience of the application.

## 4.2 Data Design

Data design in the "QR Share: QR Code Generator and Scanner for Data Transfer" project plays a crucial role in ensuring efficient organization, management, and manipulation of data. This section outlines the key aspects of data design, including schema design and data integrity constraints.

### 4.2.1 Schema Design

* User

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| UserID | Username | Password | Email | FirstName | LastName |

Table 3: User Schema Design

* Admin

|  |  |  |  |
| --- | --- | --- | --- |
| AdminID | AdminUsername | Password | Email |

Table 4: Admin Schema Design

* QRCode

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| QRCodeID | CodeContent | CodeType | DateCreated | QRCodeImage | Favorite |

Table 5: QRCode Schema Design

* History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HistoryID | QRCodeID | Action | Timestamp | UserID |

Table 6: History Schema Design

* Favourite

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| FavoriteID | QRCodeID | CodeContent | CodeType | Timestamp | QRCodeImage | UserID |

Table 7: Favorite Schema Design

### 4.2.2 Data Integrity and Constraints

* User

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Size** | **Constraint** | **Description** |
| UserID | integer | - | Primary key | A primary key serving as a unique identifier for each User. |
| Username | varchar | 30 | Not null, Unique | Stores the Username of the user. |
| Password | varchar | 20 | Not null | Stores the Password of the user. |
| Email | varchar | 50 | Not null, Unique | Stores the Email of the user. |
| FirstName | varchar | 20 | Not null | Stores the First name of the user. |
| LastName | varchar | 20 | Not null | Stores the Last Name of the user. |

Table 8: User Data Integrity and Constraints

* Admin

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Size** | **Constraint** | **Description** |
| AdminID | integer | - | Primary key | A primary key serving as a unique identifier for Admin. |
| AdminUsername | varchar | 30 | Not null, Unique | Stores the Admin Username of the Admin. |
| Password | varchar | 20 | Not null | Stores the Password of the Admin. |
| Email | varchar | 50 | Not null, Unique | Stores the Email of the Admin. |

Table 9: Admin Data Integrity and Constraints

* QRCode

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Size** | **Constraint** | **Description** |
| QRCodeID | integer | - | Primary key | A primary key serving as a unique identifier for QRCode event. |
| CodeContent | varchar | 150 | Not null | Stores the actual data encoded in the QR code, which can vary in content, such as text, URLs, contact information, etc. |
| CodeType | varchar | 50 | Not null | Indicates the type of data encoded within the QR code, allowing differentiation between text, URLs, contacts, and more. |
| DateCreated | datetime | - | Not null | Stores timestamp of generated and scanned QR codes |
| QRCodeImage | blob | - | Not null | Image of Generated/Scanned QR Codes |
| UserID | integer | - | Foreign key | A foreign key referencing the User entity. |
| FavoriteId | integer | - | Foreign key | A foreign key referencing the Favorite entity |

Table 10: QRCode Data Integrity and Constraints

* History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Size** | **Constraint** | **Description** |
| HistoryID | integer | - | Primary key | A primary key serving as a unique identifier for History event. |
| QRCodeID | integer | - | Foreign key | A foreign key referencing the QRCode entity. |
| Action | varchar | 50 | Not null | Describes the action taken by the user, such as QR code generation, scanning, or data transfer. |
| Timestamp | datetime | 30 | Not null | Records the exact date and time when the action occurred. |
| UserID | integer | - | Foreign key | A foreign key referencing the User entity. |

Table 11: History Data Integrity and Constraints

* Favorite

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Size** | **Constraint** | **Description** |
| FavoriteID | integer | - | Primary key | A primary key serving as a unique identifier for Favorite event. |
| QRCodeID | integer | - | Foreign key | A foreign key referencing the QRCode entity. |
| CodeContent | varchar | 150 | Not null | Stores the actual data encoded in the QR code, which can vary in content, such as text, URLs, contact information, etc. |
| CodeType | varchar | 50 | Not null | Indicates the type of data encoded within the QR code, allowing differentiation between text, URLs, contacts, and more. |
| Timestamp | datetime | 30 | Not null | Records the exact date and time when the action occurred. |
| QRCodeImage | blob | - | Not null | Image of generated/scanned QR Codes |
| UserID | integer | - | Foreign key | A foreign key referencing the User entity. |

Table 12: Favorite Data Integrity and Constraints

## 4.3 Procedural Design

Procedural design is a pivotal phase in the development of the "QR Share: QR Code Generator and Scanner for Data Transfer" project. It involves the systematic development of algorithms and procedures to drive the functionality of the application. In this section, we will discuss logic diagrams, data structures, and algorithm design.

### 4.3.1 Logic Diagrams

Logic diagrams are invaluable tools in software development, including the "QR Share: QR Code Generator and Scanner for Data Transfer" project. They define the systematic flow of procedures, improving comprehension for both developers and project stakeholders. In this project, logic diagrams take the form of Control Flow Charts and Process Diagrams, aiding in the understanding of the application's inner workings

In the context of the QR Share project, logic diagrams are used to visually represent the flow of procedures and operations within the application. They enhance comprehension, facilitate communication among team members, and assist programmers during the implementation phase.

* Control Flow Chart: A Control Flow Chart is a visual representation of the procedural logic within the application. It outlines the sequence of steps and decision points that guide the flow of control in the software.

Below is a simplified Control Flow Chart illustrating the core processes of generating and scanning QR codes in the application:

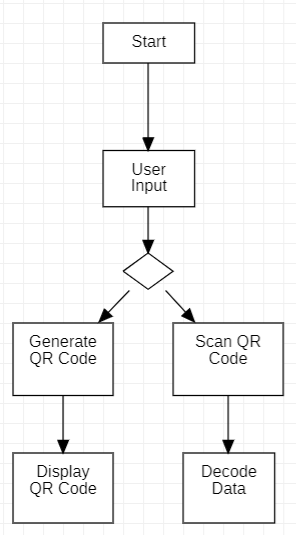


Figure 11: Control Flow Chart example

Description:

Start: The process begins.

User Input: The application gathers input data from the user.

Generate QR Code: The application generates a QR code based on the user's input.

Display QR Code: The generated QR code is displayed to the user.

End: The process concludes.

### 

### 4.3.2 Data structures

Data structures play a pivotal role in procedural design by defining how data is organized and accessed within the application. In the QR Share project, key data structures include:

* User Structure:

Attributes: UserID, Username, Password, Email, FirstName, LastName.

* QRCode Structure:

Attributes: QRCodeID, CodeData, CodeType, DateCreated, UserID, CustomizationID.

* ScanQRCode Structure:

Attributes: ScanID, QRCodeID, UserID, ScannedDate.

These data structures help organize user-related information and QR code data efficiently, enabling seamless operations within the application.

## 4.4 User Interface Design

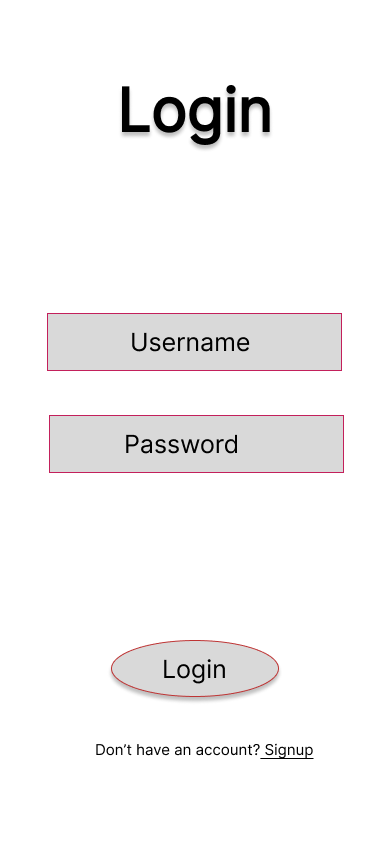


Figure 13: Login Page UI

* The Login page provides a secure entry point for registered users to access the app's features.
* Users will enter their credentials (username and password) to log in.

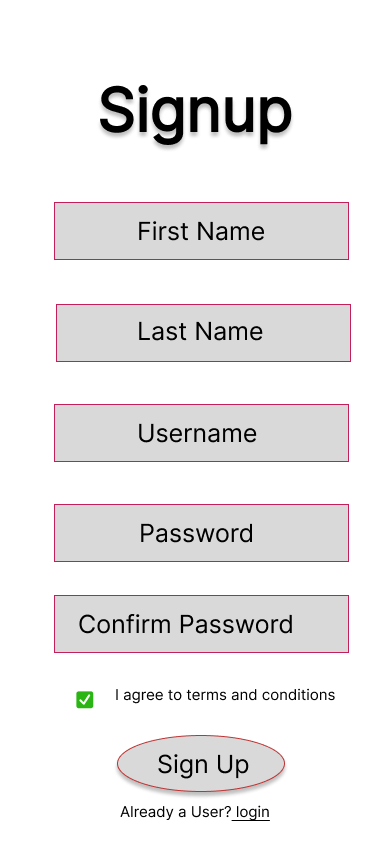


Figure 14: Signup Page UI

* The Signup page allows new users to create an account.
* It collects essential information (First name, Last name, Username, password) to set up their profile.

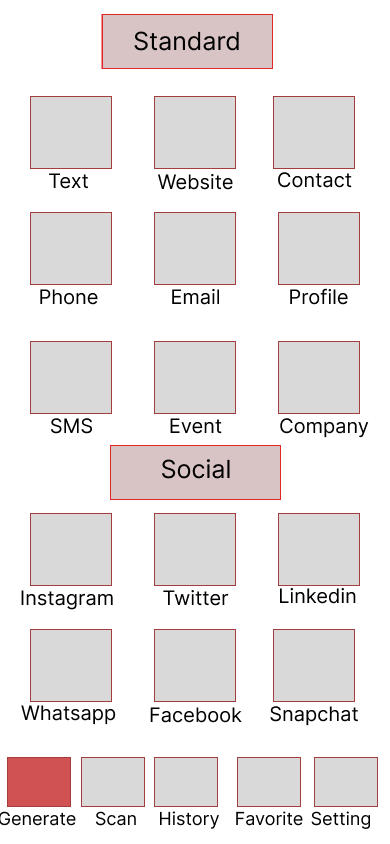


Figure 15: Generate Page UI

* This page enables users to Generate QR codes from various types of data, such as Website, text, email, phone, Contact information and many more.

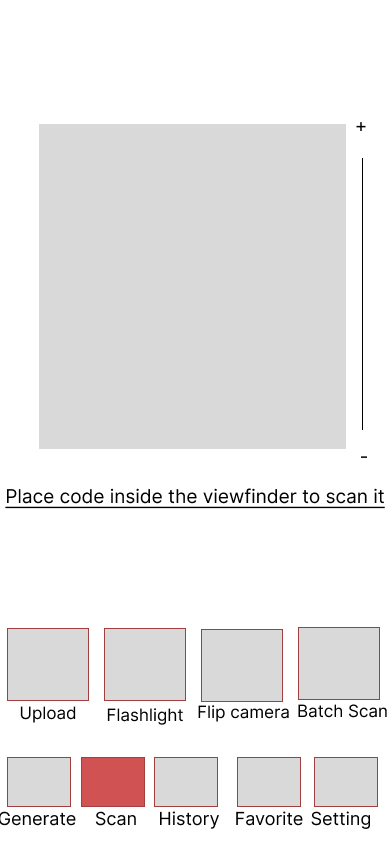


Figure 16: Scan Page UI

* The Scan QR Code page activates the device's camera to scan QR codes or user can upload image of QR Code.
* When a QR code is successfully scanned, it displays the relevant information or takes the specified action.

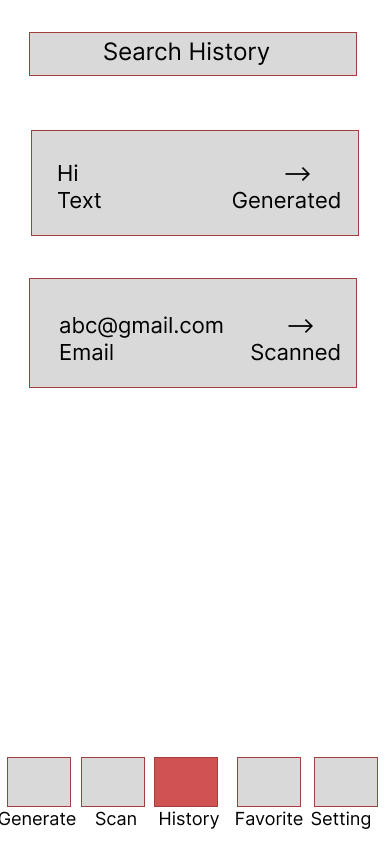


Figure 17: History Page UI

* The History page maintains a record of previously generated and scanned QR codes.
* Users can review their QR code activity history through History page.

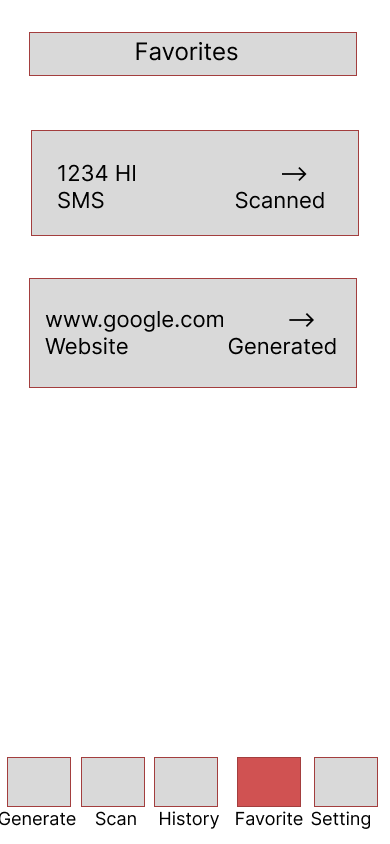


Figure 18: Favorite Page UI

* The Favorites page allows users to mark QR codes as favorites, making it easy to access and use frequently used QR codes.

## 4.5 Security Issues:

Security is a critical aspect of any android application, and QR Share is no exception. Here are some potential security issues and considerations related to the project, along with strategies to mitigate them:

1. Data Privacy and User Authentication:

* Issue: Protecting user data, including personal details and shared information, is paramount. Unauthorized access to user accounts can lead to privacy breaches.
* Mitigation: Implementing secure user authentication mechanisms, such as password hashing and encryption. Implementing multi-factor authentication for added security.

2. Secure QR Code Generation:

* Issue: QR codes can contain sensitive information. If QR code generation is not secure, attackers might generate malicious QR codes.
* Mitigation: Ensuring that the QR code generation process is secure and reliable. Using cryptographic libraries to generate QR codes securely. Validating user-generated content before encoding it into a QR code to prevent malicious data.

3. QR Code Scanning Security:

* Issue: Malicious QR codes can lead to various security threats, such as phishing attacks or directing users to harmful websites.
* Mitigation: Implementing a robust QR code scanning library that can detect and alert users about potentially harmful QR codes

4. Data Transmission Security:

* Issue: During data transfer, especially when sharing sensitive information, data interception and eavesdropping are concerns.
* Mitigation: Using secure communication protocols like HTTPS to encrypt data in transit. Ensuring that all API endpoints are secure and validating requests on the server side.

5. Authorization and Access Control:

* Issue: Unauthorized users gaining access to admin functions or other restricted areas can disrupt the application.
* Mitigation: Implementing role-based access control (RBAC) to restrict access to sensitive functionalities. Use proper session management and enforce user permissions at the server-side.

6. Data Storage Security:

* Issue: Data stored locally on the user's device can be vulnerable if not adequately protected.
* Mitigation: Encrypting sensitive data stored on the device using strong encryption algorithms. Implementing secure data storage practices and clear user data when necessary.

7. Code Security and Vulnerability Scanning:

* Issue: Security vulnerabilities in the application's code can be exploited by attackers.
* Mitigation: Regularly conducting security code reviews and use automated vulnerability scanning tools to identify and fix security issues. Staying updated with security best practices and libraries.

8. Secure File Upload and Download:

* Issue: Uploading and downloading files securely is essential, especially if QR codes contain attachments or documents.
* Mitigation: Implementing secure file upload and download mechanisms, including file type validation, ensuring that uploaded files do not execute code.

By proactively addressing these security issues and following a robust security plan, the QR Share project can provide users with a safe and secure environment for generating, sharing, and scanning QR codes.

## 4.6 Test Case Design

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Index** | **Test Case** | **Test Data** | **State** | **Test Input Values** | **Expected Result** |
| 1 | User Signup | User registration details (name, email, password) | Valid | Valid user registration details  Name: abc  Email: abc@gmail.com  Password: abc123 | User account created successfully, redirected to user dashboard |
| 2 | User Signup | User registration details (name, email, password) | Invalid | Invalid user registration details (incorrect number, email)  Name: abc  Email: abcdef  Password: abc1223!!2 | Please enter valid details |
| 3 | User Login | User login details (email, password) | Valid | Valid user login details  Email: abc@gmail.com  Password: abc123 | User successfully logged in, redirected to user dashboard |
| 4 | User Login | User login details (email, password) | Invalid | Invalid user login details (incorrect email, password)  Email: abc@gmail.com  Password: abc122323 | Please enter valid email or password |
| 5 | Admin Login | Admin login details (email, password) | Valid | Valid Admin login credentials  Email: [janhvibaraskar03@gmail.com](mailto:janhvibaraskar03@gmail.com)  Password: janhvi03 | Admin successfully logged in, redirected to Admin dashboard |
| 6 | Admin Login | Admin login details (email, password) | Invalid | Invalid Admin login details (incorrect number, email)  Email: [janhvibaraskar0333@gmail.com](mailto:janhvibaraskar0333@gmail.com)  Password: janhvi03123 | Please enter valid email or password |
| 7 | QR Code Generation | User inputs data for QR Code (text, url, email etc.) | Valid | User input data  Email: abc@gmail.com | QR code generated with user data, displayed on the screen |
| 8 | QR Code Generation | User inputs data for QR Code (text, url, email etc.) | Invalid | User input data  Email: abcde | Please Enter valis Details |
| 9 | Customization | User customizes a QR code | Valid | User customizes QR Code as per the preference | QR code appearance updated with user customizations |
| 10 | QR Code Scanning | QR code image scanned by the user's device camera | Valid | QR code for scanning or selecting QR code from device to scan | QR code decoded, displayed data matches the input data |
| 11 | QR Code Scanning | QR code image scanned by the user's device camera | Invalid | Invalid QR Code  (Some other image) | Not a valid QR Code |
| 12 | Data Transfer | User selects a data sharing method (email, etc.) | Valid | Select data transfer option like:  Gmail or messaging app, etc | Data transferred successfully to the selected method |
| 13 | History Tracking | User accesses the History feature | Valid | Accesses the history of QR Codes | System displays the user's history of generated and scanned QR codes |
| 14 | Favourite Feature | User marks a QR code as a favourite | Valid | Selects a QR code to mark as a favourite | QR code marked as a favourite for the user, stored for future reference |
| 15 | Favourite Feature | User removes a QR code from favourite | Valid | Selects a QR code to remove from favourite | QR code get removes from favourite section. |
| 16 | User/Admin logout | N/A | Valid | User/admin clicks on logout button | User/Admin is logged out of the system and redirected to the login page. |

Table 19:Test Case

## 4.7 Event Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sr. no** | **Event** | **Trigger** | **Source** | **Activity** | **Response** | **Destination** |
| 1 | User signup | User clicks on “sign Up button” | User | Provide required information and sign up | Confirmation message | Database |
| 2 | User Login | User clicks on “Login button” | User | Provide required information and Authenticate user | Redirect to user dashboard | System |
| 3 | Admin Login | Admin clicks on “Login button” | Admin | Provide required information and Authenticate admin | Redirect to admin dashboard | System |
| 4 | Generate QR Code | User selects datatype and inputs the data | User | Process data and create QR Cide | Displays generated QR Code | User interface |
| 5 | Scan QR Code | User scans QR Code | User | Capture QR Code data | Displays decoded data | User interface |
| 6 | Customize QR Code | User selects customization options | User | Apply chosen customization | Displays customized QR Code | User interface |
| 7 | Share QR Code | User selects sharing method | User | Initiate data transfer | Data is shared through chosen method | System |
| 8 | Store QR Code | User selects to store QR Code locally | User | QR code is saved on user's device | QR code is saved on user's device | Local storage |
| 9 | Manage QR Code History | User accesses QR Code History | User | Views previously generated/scanned codes | User sees a list of generated/scanned QR Codes | Database |
| 10 | Favourite feature | User marks QR code as favourite | User | Flag QR code as favourite | QR code marked as favourite | Database |
| 11 | Admin Access | Admin clicks "Admin Access" button | Admin | Gain admin-level access | Redirect to admin dashboard | System |
| 12 | Log out | User or admin clicks on log out button | User/  Admin | Log out from the App | User or Admin logged out | System |

Table 20: Event

## 4.8 Testing Approach

Testing is an integral part of the software development process, ensuring that the product functions as expected and meets its quality standards. The testing approach for the "QR Share: QR Code Generator and Scanner for Data Transfer" project follows a systematic and comprehensive methodology to guarantee the reliability and efficiency of the application.

### 4.8.1 Unit Testing

* What is Unit Testing?

Unit testing is a software testing technique that focuses on evaluating individual components or units of code in isolation. A "unit" typically refers to the smallest testable part of an application, such as a function, method, or class. In unit testing, these components are examined independently to verify that they perform as intended.

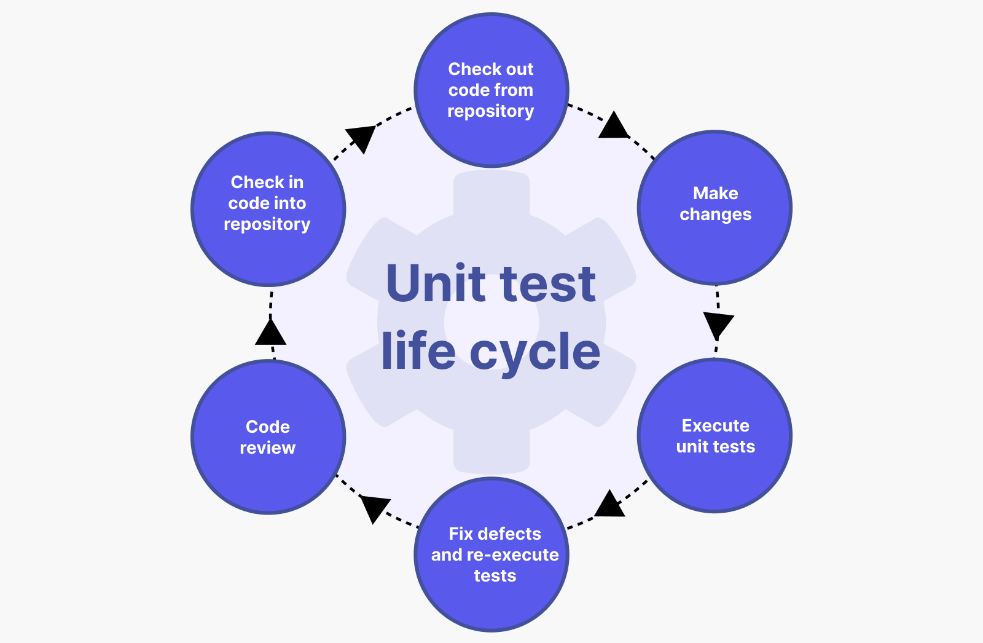


Figure 19: Unit test life cycle

* Importance of Unit Testing:

Unit testing serves several critical purposes within the software development process:

* + Bug Identification: It helps detect and rectify bugs or defects at an early stage, reducing the cost and effort required for later-stage debugging.
  + Isolation of Issues: By testing units in isolation, developers can pinpoint the source of errors more precisely, facilitating quicker resolution.
  + Code Verification: Unit tests validate that individual components conform to their design specifications, ensuring code correctness.
  + Regression Testing: Unit tests can be automatically rerun whenever code changes are made, preventing the introduction of new defects during the development process.
* Why is Unit Testing used?

Unit testing is employed for various reasons, including:

* + Quality Assurance: It enhances the overall quality and reliability of the software by identifying and fixing issues early in development.
  + Code Refactoring: It provides confidence that code changes or optimizations do not introduce new defects.
  + Documentation: Unit tests serve as executable documentation, explaining how a component is intended to function.
* Advantages of Unit Testing:
  + Early Bug Detection: Bugs and issues are identified and addressed during the coding phase, reducing the likelihood of costly defects in later stages.
  + Enhanced Code Quality: It encourages developers to write cleaner, modular, and more maintainable code.
  + Improved Software Design: Unit tests promote a modular and component-based approach to software design.
  + Rapid Feedback: Developers receive immediate feedback on the correctness of their code, enabling faster development iterations.

### 4.8.2 Integration Testing

* What is Integration Testing?

Integration testing is a software testing approach that focuses on assessing the interactions between various components or modules of a system. Unlike unit testing, which tests individual units in isolation, integration testing examines how these units work together when integrated into a larger system.

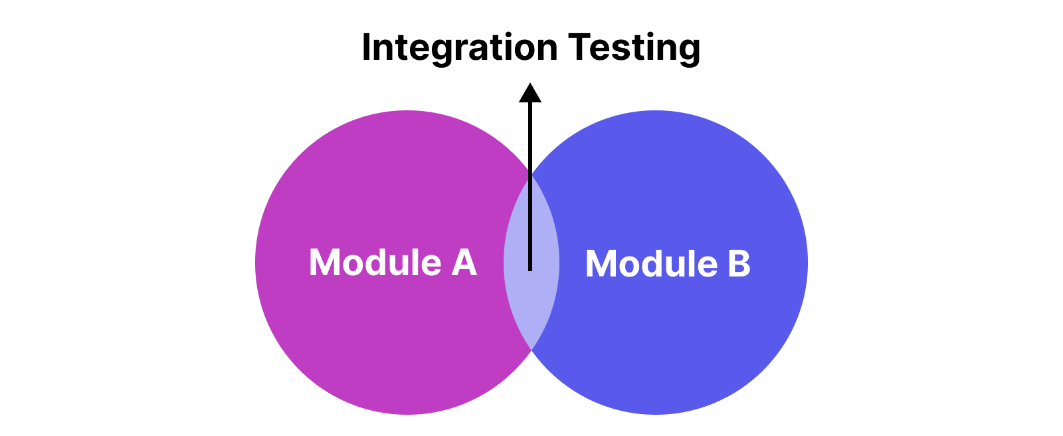


Figure 20: Integration Testing

* Importance of Integration Testing:

Integration testing plays a crucial role in ensuring the reliability and functionality of a software system:

* + Interaction Verification: It verifies that different components can interact correctly and share data as expected.
  + Data Flow: Integration testing evaluates the flow of data between modules, ensuring that data is processed correctly as it moves through the system.
  + Interface Compatibility: It ensures that the interfaces between components are compatible and that data is transmitted accurately between them.
* Why is Integration Testing Used?

Integration testing is employed for several reasons:

* Interaction Validation: It validates that various components interact seamlessly, minimizing the risk of integration-related defects.
* System Validation: It helps assess whether the integrated system meets its design specifications and functional requirements.
* Risk Mitigation: It identifies integration issues early, reducing the likelihood of costly defects emerging in later phases.
* Advantages of Integration Testing:
  + Defect Detection: It helps detect and address integration-related issues, such as data mismatches, interface problems, and communication errors.
  + End-to-End Validation: Integration testing verifies that the system functions correctly as a whole, rather than just individual components.
  + Confidence Building: Successful integration tests instil confidence in the software's reliability and readiness for subsequent testing phases and deployment.
  + Quality Assurance: By addressing integration issues early, it contributes to the overall quality and stability of the software system.

In summary, unit testing and integration testing are essential components of the testing approach for the "QR Share: QR Code Generator and Scanner for Data Transfer" project. Unit testing focuses on individual code units, ensuring their correctness and reliability, while integration testing examines the interactions and data flow between these units within the larger system, ensuring seamless operation. Both testing types are crucial for identifying and addressing defects early in the development process and ensuring the software meets its design and functional requirements.

## 4.9 References

* <https://saturncloud.io/blog/>
* <https://katalon.com/resources-center/blog/>
* <https://atlan.com/>
* <https://www.tutorialspoint.com/>
* <https://developer.ibm.com/>
* <https://github.com/zxing/zxing>
* <https://ijarsct.co.in/>
* <https://www.qrcode.com/en/about/>
* [www.researchgate.net](http://www.researchgate.net)