GCash Online Banking System Documentation

Created by:

Jonathan Guevara  
AiTeam Pinoy Technology Inc.  
[jguevara@aiteampinoy.com](mailto:jguevara@aiteampinoy.com)

*Documentation Version: 1.0*  
*Last Updated: August 2025*  
*System Version: GCash Banking v1.0*

Table of Contents

[1 System Overview 1](#_Toc206493008)

[2 Project Structure 2](#_Toc206493009)

[3 Architecture 3](#_Toc206493010)

[4 Database Design 4](#_Toc206493011)

[5 Component Documentation 5](#_Toc206493012)

[6 Security Features 7](#_Toc206493013)

[7 Installation & Setup 8](#_Toc206493014)

[8 User Guide 10](#_Toc206493015)

[9 API Reference 11](#_Toc206493016)

[10 Development Guidelines 13](#_Toc206493017)

# System Overview

GCash Online Banking System is a Java Swing-based desktop application that simulates core banking functionalities including user registration, authentication, and account management.

## Key Features

* User Registration & Authentication
* Secure PIN-based Login
* Session Management
* SQLite Database Integration
* Modern GUI with GCash Branding
* Input Validation & Security

## Technology Stack

* Language: Java 24.0.1 (Java SE)
* Runtime: Java HotSpot™ 64-Bit Server VM
* GUI Framework: Java Swing
* Database: SQLite
* Security: SHA-256 Hashing, Salt-based Security
* Pattern: MVC (Model-View-Controller)

# Project Structure

gcash-banking-app/

│

├── src/

│ └── com/

│ └── gcashapp/

│ ├── GcashApp.java # Main application entry point

│ │

│ ├── model/ # Data models and business logic

│ │ ├── User.java # User entity model

│ │ ├── Database.java # Database connection and initialization

│ │ └── UserAuthentication.java # Authentication logic

│ │

│ ├── view/ # User interface components

│ │ ├── LoginFrame.java # Login screen

│ │ ├── RegistrationFrame.java # Registration screen

│ │ └── DashboardFrame.java # Main dashboard

│ │

│ └── controller/ # Application controllers

│ └── SessionController.java # User session management

│

├── database/

│ └── gcash\_banking.db # SQLite database file (auto-generated)

│

├── lib/ # External libraries

│ └── sqlite-jdbc-x.x.x.jar # SQLite JDBC driver

│

├── resources/ # Application resources

│ ├── images/ # UI images and icons

│ └── config/ # Configuration files

│

├── docs/ # Documentation

│ ├── api/ # API documentation

│ ├── user-manual/ # User manuals

│ └── technical/ # Technical documentation

│

├── build/ # Compiled classes

├── dist/ # Distribution files

├── README.md # Project overview

└── build.xml # Ant build file (optional)

## Package Structure Details

com.gcashapp/

├── GcashApp.java # Application launcher

├── model/ # Data layer

│ ├── User.java # User entity

│ ├── Database.java # Database operations

│ └── UserAuthentication.java # Auth services

├── view/ # Presentation layer

│ ├── LoginFrame.java # Login UI

│ ├── RegistrationFrame.java # Registration UI

│ └── DashboardFrame.java # Dashboard UI

└── controller/ # Control layer

└── SessionController.java # Session management

# Architecture

## MVC Architecture Pattern

The application follows the Model-View-Controller (MVC) architectural pattern:

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ VIEW │ │ CONTROLLER │ │ MODEL │

│ │ │ │ │ │

│ - LoginFrame │◄──►│ SessionController│◄──►│ - User │

│ - Registration │ │ │ │ - Database │

│ - Dashboard │ │ │ │ - UserAuth │

└─────────────────┘ └─────────────────┘ └─────────────────┘

## Component Interactions

graph TB

A[GcashApp Main] --> B[LoginFrame]

B --> C[UserAuthentication]

C --> D[Database]

B --> E[RegistrationFrame]

E --> C

C --> F[SessionController]

F --> G[DashboardFrame]

G --> F

# Database Design

## Database Schema

### Users Table

CREATE TABLE users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

email TEXT UNIQUE NOT NULL,

number TEXT UNIQUE NOT NULL,

pin TEXT NOT NULL,

salt TEXT NOT NULL,

status TEXT DEFAULT 'active',

created\_at TEXT DEFAULT CURRENT\_TIMESTAMP,

updated\_at TEXT DEFAULT CURRENT\_TIMESTAMP,

last\_login TEXT,

last\_logout TEXT

);

### Accounts Table

CREATE TABLE accounts (

id INTEGER PRIMARY KEY AUTOINCREMENT,

user\_id INTEGER NOT NULL,

account\_number TEXT UNIQUE NOT NULL,

balance REAL DEFAULT 0.0,

created\_at TEXT DEFAULT CURRENT\_TIMESTAMP,

updated\_at TEXT DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES users(id)

);

### Transactions Table

CREATE TABLE transactions (

id INTEGER PRIMARY KEY AUTOINCREMENT,

account\_id INTEGER NOT NULL,

transaction\_type TEXT NOT NULL,

amount REAL NOT NULL,

balance\_after REAL NOT NULL,

description TEXT,

created\_at TEXT DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (account\_id) REFERENCES accounts(id)

);

## Entity Relationship Diagram

Users (1) ─────── (1..n) Accounts (1) ─────── (1..n) Transactions

│ │ │

├── id (PK) ├── id (PK) ├── id (PK)

├── name ├── user\_id (FK) ├── account\_id (FK)

├── email ├── account\_number ├── transaction\_type

├── number ├── balance ├── amount

├── pin ├── created\_at ├── balance\_after

├── salt └── updated\_at ├── description

├── status └── created\_at

├── created\_at

├── updated\_at

├── last\_login

└── last\_logout

# Component Documentation

## 1. Main Application (GcashApp.java)

**Purpose**: Application entry point and initialization

**Key Functions**:

* Sets system look and feel
* Initializes database
* Launches login interface

public class GcashApp {

public static void main(String[] args)

}

## 2. User Model (User.java)

**Purpose**: User entity representation

**Properties**:

* id: Unique user identifier
* name: User's full name
* email: User's email address
* number: Phone number
* lastLogin: Last login timestamp
* status: Account status

## 3. Database (Database.java)

**Purpose**: Database connection and schema management

**Key Methods**:

* connect(): Establishes SQLite connection
* initialize(): Creates database schema

## 4. UserAuthentication (UserAuthentication.java)

**Purpose**: Handles user authentication and registration

**Key Methods**:

### Registration

public static RegistrationResult registration(String name, String email, String number, String pin)

* Validates input data
* Checks for existing users
* Hashes PIN with salt
* Creates new user record

### Login

public static AuthenticationResult login(String emailOrPhone, String pin)

* Validates credentials
* Verifies PIN hash
* Updates last login
* Returns user session data

### Security Methods

private static String hashPin(String pin)

private static String generateSalt()

private static boolean verifyPin(String inputPin, String storedPin, String salt)

## 5. View Components

### LoginFrame.java

* **Purpose**: User login interface
* **Features**: Email/phone input, PIN entry, navigation to registration

### RegistrationFrame.java

* **Purpose**: User registration interface
* **Features**: Form validation, PIN confirmation, user creation

### DashboardFrame.java

* **Purpose**: Main application interface after login
* **Features**: User welcome, logout functionality

## 6. SessionController (SessionController.java)

**Purpose**: Manages user session state

**Key Methods**:

* setCurrentUser(User user): Sets active user
* getCurrentUser(): Returns current user
* clearSession(): Clears session data
* isUserLoggedIn(): Checks login status

# Security Features

## 1. Password Security

* **SHA-256 Hashing**: All PINs are hashed using SHA-256
* **Salt Generation**: Unique salt for each user
* **Secure Random**: Cryptographically secure random salt generation

## 2. Input Validation

* **Email Validation**: Regex pattern matching
* **Phone Number Validation**: Philippine format validation
* **PIN Validation**: 4-6 digit requirement

## 3. Database Security

* **Prepared Statements**: Prevents SQL injection
* **Parameterized Queries**: Safe database operations
* **Connection Management**: Proper resource cleanup

## 4. Session Management

* **User Session Tracking**: Secure session state
* **Logout Functionality**: Proper session cleanup
* **Authentication Verification**: Route protection

# Installation & Setup

## Prerequisites

* Java Development Kit (JDK) 24 or higher
  + Java™ SE Runtime Environment (build 24.0.1+9-30)
  + Java HotSpot™ 64-Bit Server VM (build 24.0.1+9-30)
* SQLite JDBC Driver (compatible with Java 24)
* IDE (Eclipse, IntelliJ IDEA, or NetBeans) with Java 24 support

## Setup Steps

### Clone/Download Project

1. # Create project directory
2. mkdir gcash-banking-app
3. cd gcash-banking-app
4. Create Directory Structure
5. mkdir -p src/com/gcashapp/{model,view,controller}
6. mkdir -p database
7. mkdir -p lib
8. mkdir -p resources/{images,config}
9. Add SQLite JDBC Driver
   1. Download sqlite-jdbc-x.x.x.jar
   2. Place in lib/ directory
   3. Add to classpath

### Compile Application

1. # Using Java 24 compiler
2. javac -cp "lib/\*" -d build src/com/gcashapp/\*\*/\*.java

### Run Application

1. # Using Java 24 runtime
2. java -cp "build:lib/\*" com.gcashapp.GcashApp

## IDE Setup (Eclipse with Java 24)

1. Create new Java Project (Java 24)
2. Set project compliance level to 24
3. Add source files to appropriate packages
4. Add SQLite JAR to build path (ensure Java 24 compatibility)
5. Run GcashApp.java as Java Application

## Java 24 Specific Considerations

### Module System (if using modules)

module com.gcashapp {

requires java.desktop;

requires java.sql;

requires java.base;

exports com.gcashapp;

exports com.gcashapp.model;

exports com.gcashapp.view;

exports com.gcashapp.controller;

}

### Text Blocks (Enhanced in Java 24)

private static final String CREATE\_USERS\_TABLE = """

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

email TEXT UNIQUE NOT NULL,

number TEXT UNIQUE NOT NULL,

pin TEXT NOT NULL,

salt TEXT NOT NULL,

status TEXT DEFAULT 'active',

created\_at TEXT DEFAULT CURRENT\_TIMESTAMP,

updated\_at TEXT DEFAULT CURRENT\_TIMESTAMP,

last\_login TEXT,

last\_logout TEXT

)

""";

### Enhanced Switch Expressions

public String getStatusMessage(AuthenticationResult result) {

return switch (result.isSuccess()) {

case true -> "Login successful!";

case false -> result.getMessage();

};

}

# User Guide

## First Time Setup

### Launch Application

* + Run the application
  + Login screen appears

### Create Account

* + Click "Register" button
  + Fill in required information:
    - Full Name
    - Email Address
    - Phone Number (Philippine format)
    - 4-6 digit PIN
  + Confirm PIN
  + Click "Register"

### Login

* + Enter email or phone number
  + Enter PIN
  + Click "Login"

## Using the Application

### Registration Process

* + Enter personal information
  + Create secure PIN
  + Confirm PIN matches
  + Submit registration
  + Note your User ID for reference

### Login Process

* + Enter email or phone number
  + Enter your PIN
  + Access dashboard upon successful login

### Dashboard Features

* + View welcome message
  + Access account information
  + Logout securely

## 3. Validation Rules

### Email Format

* + Must contain @ symbol
  + Valid domain extension
  + Example: user@example.com

### Phone Number Format

* + Philippine mobile format
  + Accepts: +639xxxxxxxxx, 639xxxxxxxxx, 09xxxxxxxxx
  + Example: +639123456789

### PIN Requirements

* + 6 digits only
  + Numbers only
  + Example: 123456

# API Reference

## UserAuthentication Class

### registration()

public static RegistrationResult registration(String name, String email, String number, String pin)

**Parameters:**

* + - name: User's full name (required)
    - email: Valid email address (required, unique)
    - number: Phone number (required, unique)
    - pin: 4-6 digit PIN (required)

**Returns:** RegistrationResult

* isSuccess(): Boolean success status
* getMessage(): Result message
* getUserId(): Generated user ID

### login()

public static AuthenticationResult login(String emailOrPhone, String pin)

**Parameters:**

* emailOrPhone: Email address or phone number
* pin: User's PIN

**Returns:** AuthenticationResult

* isSuccess(): Boolean success status
* getMessage(): Result message
* getUser(): User object
* getUserId(): User ID

### changePin()

public static boolean changePin(String userId, String oldPin, String newPin)

**Parameters:**

* userId: User's ID
* oldPin: Current PIN
* newPin: New PIN (4-6 digits)

**Returns:** Boolean success status

## Database Class

### connect()

public static Connection connect() throws SQLException

**Returns:** SQLite database connection

### initialize()

public static void initialize()

Creates database schema if not exists

## SessionController Class

### Session Management

public static void setCurrentUser(User user)

public static User getCurrentUser()

public static void clearSession()

public static boolean isUserLoggedIn()

# Development Guidelines

## Code Standards

Java 24 Language Features

### Text Blocks for SQL Queries

private static final String CREATE\_USERS\_SQL = """

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

email TEXT UNIQUE NOT NULL,

number TEXT UNIQUE NOT NULL

)

""";

### Enhanced Pattern Matching

public String processResult(Object result) {

return switch (result) {

case AuthenticationResult auth when auth.isSuccess() ->

"Authentication successful for user: " + auth.getUserId();

case RegistrationResult reg when reg.isSuccess() ->

"Registration completed with ID: " + reg.getUserId();

case null -> "No result provided";

default -> "Unknown result type";

};

}

### Record Classes for Data Transfer

public record UserCredentials(String emailOrPhone, String pin) {

public UserCredentials {

// Compact constructor with validation

if (emailOrPhone == null || emailOrPhone.isBlank()) {

throw new IllegalArgumentException("Email or phone cannot be blank");

}

if (pin == null || pin.isBlank()) {

throw new IllegalArgumentException("PIN cannot be blank");

}

}

}

### Naming Conventions

* + **Classes**: PascalCase (UserAuthentication)
  + **Methods**: camelCase (performLogin)
  + **Variables**: camelCase (emailField)
  + **Constants**: UPPER\_SNAKE\_CASE (EMAIL\_PATTERN)
  + **Records**: PascalCase with descriptive names

### Code Organization

* + One class per file
  + Package organization by functionality
  + Clear separation of concerns

## 2. Error Handling

### Database Operations (Java 24 Enhanced)

// Using try-with-resources with Java 24 enhancements

try (var conn = Database.connect();

var pstmt = conn.prepareStatement(SQL\_QUERY)) {

// Database operations with enhanced error handling

pstmt.setString(1, userInput);

var resultSet = pstmt.executeQuery();

} catch (SQLException e) {

// Enhanced exception handling in Java 24

System.err.printf("Database error: %s%n", e.getMessage());

throw new DatabaseException("Failed to execute query", e);

}

### Modern Error Handling Pattern

public sealed interface DatabaseResult<T>

permits Success, Failure {

record Success<T>(T value) implements DatabaseResult<T> {}

record Failure<T>(String error, Throwable cause) implements DatabaseResult<T> {}

default boolean isSuccess() {

return this instanceof Success<T>;

}

}

### Input Validation

* + Always validate user input
  + Use regex patterns for format validation
  + Provide clear error messages

## 3. Security Best Practices

### PIN Handling

* + Never store plain text PINs
  + Always use salt with hashing
  + Clear PIN variables after use

### Database Access

* + Use prepared statements
  + Validate all inputs
  + Handle SQL exceptions properly

## 4. UI Guidelines

### Swing Components

* + Use appropriate layout managers
  + Implement proper event handling
  + Provide user feedback for actions

### Visual Design

* + Consistent color scheme (GCash green: #00AF54)
  + Clear labeling and instructions
  + Responsive to different screen sizes

## 5. Testing Strategy

### Unit Testing

* + Test authentication methods
  + Validate input validation functions
  + Test database operations

### Integration Testing

* + Test complete user workflows
  + Verify database connectivity
  + Test UI interactions

### Security Testing

* + Verify PIN hashing
  + Test SQL injection prevention
  + Validate session management

## Future Enhancements

### Planned Features

1. **Account Management**
   * + Balance inquiry
     + Transaction history
     + Account statements
2. **Money Transfer**
   * + Send money functionality
     + Receive money
     + Transaction notifications
3. **Bill Payment**
   * + Utility bill payments
     + Mobile load
     + Online shopping
4. **Security Enhancements**
   * + Two-factor authentication
     + Biometric login
     + Transaction limits
5. **Reporting**
   * + Financial reports
     + Transaction analytics
     + Export functionality

### Technical Improvements

* + Database encryption
  + Multi-threading support
  + Web service integration
  + Mobile application
  + Cloud deployment

## Troubleshooting

### Common Issues

#### Database Connection Error

Problem: "Database connection failed" Solution:

* + - Check SQLite JDBC driver in classpath
    - Verify database file permissions
    - Ensure database directory exists

#### Login Failed

**Problem**: "Invalid credentials" despite correct PIN **Solution**:

* + - Verify user exists in database
    - Check account status is 'active'
    - Confirm PIN format (4-6 digits)

#### Registration Error

**Problem**: "Email already exists" **Solution**:

* + - Use unique email address
    - Check existing registrations
    - Use different email or recover existing account

### Debug Mode

Enable debug logging by setting system property:

System.setProperty("gcash.debug", "true");

## License

This project is developed for educational purposes. Please ensure compliance with relevant banking and financial regulations when adapting for commercial use.

## Support

For technical support or questions:

* + Review this documentation
  + Check troubleshooting section
  + Verify system requirements
  + Test with minimal data set