Project 01: Home Safe

Software Architecture Design
SAD Version 3

Team 01

Marina Seheon (Manager)
Andrei Phelps (Document Manager)
Luke McDougall (Lead Software Engineer)
Jack Vanlyssel
Spoorthi Menta
Vamsi Krishna Singara

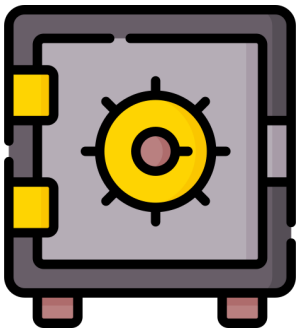

Image courtesy of Flaticon.com [2].

CS460: Software Engineering

Table of Contents

1 Introduction

This document presents the architectural framework for the HomeSafe software solution. Grounded in the Software Requirements Specification (SRS), this documentation provides a thorough breakdown of the core requirements defining the HomeSafe ecosystem. For those keen on exploring the intricate facets of HomeSafe's functionalities, interfacing elements, and logical flow, the SRS serves as a reservoir of in-depth knowledge. The design principles here are meticulously aligned with the directives established in the SRS.

In addition to an overarching outline of the system's layout, enthusiasts will discover in-depth explanations of individual design elements, accompanied by illustrative samples and potential design limitations. For an organized reading experience, key terminologies pivotal to the document's context are clarified in Section 2. Sections 3 and 4, respectively, peel back the layers on the software's architectural nuances and the intricate details of its components. Meanwhile, Section 5 enriches readers with real-world scenarios, highlighting practical applications and use cases.

2 Definition of Terms

This section provides definitions for critical terms recurrently utilized throughout the document. This section can be a reference point for readers engaging with the content.

I. **Auxiliary**: An additional or secondary power source that supports the main or primary power supply. An auxiliary power source is typically used to provide backup, redundancy, or temporary power when the main power source is unavailable, disrupted, or insufficient.

II. **Bio-Metric Scanner**: A technology that identifies and authenticates users based on their unique biological characteristics, typically fingerprints, retina patterns, or other traits.

III. **Central Processing Unit (CPU)**: Also known as a central processor or a main processor, it serves as the core component of any computer. This unit carries out the essential functions of executing program instructions, including tasks like calculations, logical operations, control functions, and managing input/output processes.

IV. **Microcontroller**: A microcontroller is a small integrated circuit serving as the central processing unit (CPU) of a safe's electronic system. It contains a processor, memory, and input/output ports and can include programmable capabilities. Microcontrollers manage the safe's tasks, user input, security protocols, and control functions, including locks, interface interactions, and external device communication.

V. **Personal Identification Number (PIN)**: A numerical code that serves as a security credential used to authenticate and verify the identity of an individual. PINs are commonly used in various systems, such as electronic devices, bank accounts, and access control systems, to ensure that only authorized users can gain access.

VI. **Two-Factor Authentication (2FA)**: A security protocol that requires users to provide two distinct forms of verification to access a system. This commonly involves a combination of something known (such as a password) and something possessed (such as a generated code or biometric information), adding an extra layer of security and protection against unauthorized access [1].

3 Architecture Design Overview

This section delves deeply into the intricate software architecture underpinning the HomeSafe lock system. As we navigate, we will emphasize understanding the individual software components, their respective roles, and the complex web of interactions among them. A comprehensive diagram will provide a holistic view, illustrating these software modules' dynamic interplay and control flow. Following the visual representation, there will be a discussion on each module's specific functionalities, their significance in the overall system, and the synergies they create when working together. By the end of this section, readers will gain a comprehensive insight into the software backbone that powers the HomeSafe lock system.

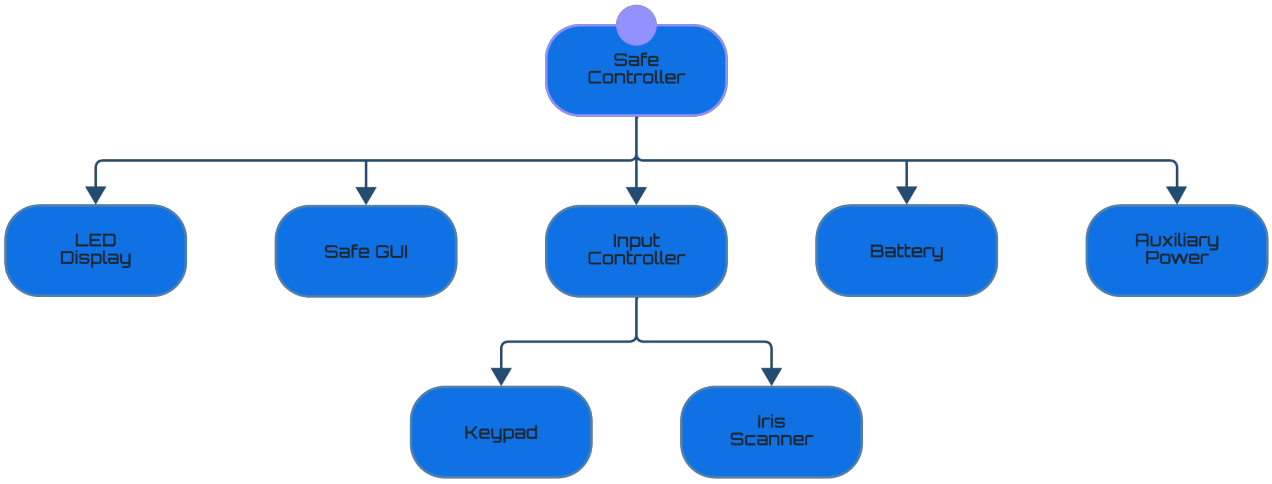

Figure 1: Architecture Design Diagram [3]

3.1 Design Description

This section delves into the core software components: SafeController(), IrisScanner(), and Keypad() classes and their interactions.

The SafeController() acts as the central command, promptly initializing the InputController(), Screen(), AuxPower(), and SafeGUI() components. When users present their eye to the biometric module, the IrisScanner() class springs into action, recognizing the eye pattern and sending the data to the InputController(). This data is then passed to the SafeController() for validation against stored iris patterns.

Concurrently, the Keypad class oversees the user interactions with the keypad. When a user keys in a 6-digit PIN, the InputController() processes this input and coordinates with the SafeController() for verification, similar to the iris scanning procedure.

After consolidating the authentication details, the SafeController() manages the overarching system dynamics. Depending on the verification outcomes, the SafeController() triggers reactions in the Screen(), Battery(), and tactile feedback mechanisms. These software components seamlessly bridge with their tangible equivalents: the display unit, power units, and the tactile feedback system of the keypad().

4 Component Specifications

This section delves into the intricacies of the software components illustrated in the design diagram outlined in Figure 1. It offers a comprehensive perspective on each software component and underscores their interdependencies.

4.1 SafeController

The `SafeController` class acts as the microcontroller in the software architecture. It manages the state of the safe and coordinates between different components, such as the LED Display, Keypad, and SafeGUI. It holds a list of users and maintains the current user and state.

- `setState(SafeState newState)`: This method is used to change the state of the safe. It handles states like waiting for an iris scan, setting iris, initial PIN setup, normal, closed, adding a new user, locked, and master verification.
- `handleMasterVerification()`: This method is invoked when the state is set to master verification. It displays a message to enter the master PIN.
- `handleInitialPinSetup()`: and `handleNormalState()`: These methods handle the initial PIN setup and normal states by displaying appropriate messages.

- `handleUnlockedState()`: This method handles the unlocked state by displaying a message and opening the safe GUI after a delay.
- `checkPIN(String enteredPIN)`: This method checks the entered PIN against the stored PINs for various states and displays appropriate messages.

Dependencies:

- Depends on the `Screen` class to display messages.
- Depends on the `SafeGUI` class to handle the GUI-related tasks.
- Depends on the `User` class to manage user information.
- Depends on the `SafeState` enum to manage the state of the safe.

4.2 Input Controller

The `InputController` class handles the input from the user via the IRIS scanner and Keypad. It manages the power state and interacts with the `Screen`, and `SafeController` to handle various inputs.

- `handleKeyInput(String key)`: This method handles the key inputs from the user. If the power is on, it appends the key entry to the screen. Otherwise, it does nothing.
- `handleCancel()`: This method handles the cancel operation by removing the last key entry from the screen.
- `handleEnterButton()`: This method is invoked when the enter button is pressed. It checks the PIN entered by calling the `checkPIN` method of the `SafeController` and clears the key entry from the screen.
- `handlePowerButton()`: This method handles the power button press. It toggles the power state and sets the state of the `SafeController` accordingly.

Dependencies:

- Depends on the `Screen` class to display and manage key entries.
- Depends on the `SafeController` class to check the PIN and set the state of the safe.

4.3 GUI Manager (SafeGUI)

The `SafeGUI` class is responsible for managing the graphical user interface of the safe. It handles the display of images representing the safe's state (open, closed), the keypad for PIN entry, and buttons for various controls.

- `start(Stage primaryStage)`: This method initializes the primary stage of the application, setting up the images, button panel, screen, and keypad. It also handles mouse clicks on the image view to transition to a close-up view of the safe.
- `openSafe()`: This method changes the image to show the safe in an open state and hides the screen component, keypad, and button box. It also provides a button to close the safe.
- `closeSafe()`: This method is used to close the safe. It changes the state of the `safeController` to closed, updates the image to show the safe in a closed state, and makes the screen component, keypad, and button box visible again.

Dependencies:

- Depends on the `Screen` class to manage the display screen of the safe.
- Depends on the `KeyPad` class for handling keypad-related operations.
- Depends on the `ButtonPanel` class for managing the button panel.
- Depends on the `SafeController` class to manage the state of the safe.
- Depends on the `SafeState` enum to manage the state of the safe.

This class is crucial for providing a user-friendly interface, allowing users to interact with the safe effectively and efficiently, and ensuring that the state of the safe is visually represented to the user at all times.

4.4 Iris Scanner (Software)

The Iris Scanner functionality is embedded within the `SafeController` class. It is responsible for handling the iris scanning and verification process to ensure secure access to the safe.

- `handleWaitingForIris()`: This method displays a message on the screen indicating that the system is waiting for an iris scan.
- `handleSettingIris()`: This method prompts the user to scan their iris for setting or updating the iris data in the system.
- `checkIris(String irisName)`: This method checks the scanned iris data against the stored iris data for the current user. If the iris data matches, it transitions the state to unlocked; otherwise, it displays an error message and reverts to the normal state.
- `setIrisForCurrentUser(String irisName)`: This method sets the iris data for the current user in the system.

Dependencies:

- Depends on the `Screen` class to display messages related to the iris scanning process.
- Depends on the `User` class to manage and verify user data, including iris data.
- Interacts with various states defined in the `SafeState` enum to manage the iris scanning and verification process flow.

This component ensures that only authorized users whose iris data matches the stored data can access the safe, providing an additional layer of security beyond the PIN authentication.

4.5 Battery

The `Battery` class in the software simulates the hardware component responsible for powering the safe and its associated features. It is essential for ensuring the continuous and reliable operation of the safe.

- `start()`: This method begins the battery depletion process, continuously checking the battery level and updating the charge level.
- `stop()`: This method stops the battery depletion process and resets the timer.
- `discharge(double amount)`: This method discharges the battery by a specified amount.
- `isLow()`: This method checks if the battery is low based on the predefined low battery threshold.
- `getChargeLevel()`: This method returns the battery's current charge level.

Dependencies:
The `Battery` class does not depend on other classes but is crucial for the operation of any component that requires power. It uses the `Timer` and `TimerTask` classes to manage the battery depletion process.

4.6 LED Display

The `Screen` class represents the digital display of the safe. It provides functionalities to display messages, handle user input, and manage timeouts.

- `displayMessage(String message)`: This method displays a given message on the screen. If the message is longer than 20 characters, it splits into two lines.
- `appendKeyEntry(String key)`: Appends a key entry (like a digit from a PIN) to the screen and represents it as an asterisk (*).
- `clearKeyEntry()`: Clears the current key entries displayed on the screen.
- `timeout()`: Handles the scenario when there's no activity on the screen for a certain duration. It clears the key entry and displays a "Timed out" message.

- `tempDisplayMessage(String message)`: Temporarily displays a message on the screen and then reverts to the previous message after a short duration.
- `removeLastKeyEntry()`: Removes the last key entry from the screen.
- `turnOn()`: Makes the screen visible.
- `turnOff()`: Makes the screen invisible.
- `getScreenComponent()`: Returns the main component of the screen, which is a `StackPane`.

Dependencies:

- Depends on the `InputController` class to handle user input and manage the display accordingly.
- Uses `PauseTransition` from JavaFX to handle timeouts and temporary message displays.

The `Screen` class is essential for user interaction, providing feedback, and guiding the user through the various operations of the safe.

5 Sample Use Cases

This section offers two use cases showcasing varied scenarios involving HomeSafe. Each example elucidates the user's actions sequentially. Furthermore, with every user action, the reactions from the related system components are detailed.

5.1 Use Case 1: Initial Setup

- **Actor**: User
- **Goal**: Successfully initialize and configure the HomeSafe with personalized access credentials.
- **Preconditions**: The HomeSafe is powered off and is in its initial, unconfigured state.
- **Trigger**: User powers on the HomeSafe.
- **Scenario**: In this scenario, the user has just acquired their new HomeSafe and wishes to initialize and configure their access credentials before turning off the safe for subsequent use.
    1. **Unpacking and Placing the Safe**: The user removes the HomeSafe and its manual, which contains the Master PIN, from the packaging, placing it in a preferred location.
    2. **Powering Up**: The user powers up the system by pressing the power button, activating the safe's components, and prompting an LED display message to enter the Master PIN.
    3. **Entering Master PIN**: User inputs the provided 6-digit Master PIN and presses ENTER, prompting the system to verify it and request the creation of a new personal PIN.
    4. **Setting a New Personal PIN**: User inputs a desired 6-digit PIN and presses ENTER.
    5. **Biometric (Iris) Configuration**: The user aligns their eye with the Iris Scanner, waiting for the confirmation animation on the LED Display. This concludes the initial setup process, and the user's credentials are saved.
    6. **Powering Down**: The user presses the POWER button, deactivating the HomeSafe and all its internal components.

5.2 Use Case 2: Accessing Contents of HomeSafe

- **Actor**: User
- **Goal**: Gain access to the HomeSafe utilizing pre-configured PIN and Iris Scan credentials.
- **Preconditions**: The HomeSafe is configured with the user's PIN and Iris Scan and is powered off.
- **Trigger**: The user wishes to access the contents of the HomeSafe.
- **Scenario**: The user, having previously configured their PIN and Iris Scan, seeks to gain secure access to the safe utilizing these credentials.

1. **Initiating Access**: The user begins by powering up the HomeSafe. The LED display comes to life, prompting users to enter their 6-digit PIN to proceed.
2. **PIN Verification**: The user enters their personal 6-digit PIN and presses ENTER. In response, the LED display instructs the user to perform an Iris Scan, activating the scanner.
3. **Iris Scan Verification**: The user carefully positions their eye before the Iris Scanner. Upon seeing a confirmation animation on the LED Display, which signifies a successful scan, the locking mechanism shifts to an UNLOCKED state.
4. **Accessing and Managing Safe Contents**: With access granted, the user opens the safe, revealing its contents. The LED display notifies "Door Open," deactivating input controls and initiating a timer to sound an alarm if the door remains open beyond 300 seconds (5 minutes).
5. **Securing the Safe Post-Use**: After managing the contents, the user closes the HomeSafe. The LED indicates "Door Closed," reactivating the input controls and initializing a shutdown timer to power down the safe after 300 seconds (5 minutes).
6. **Concluding Interaction**: Optionally, to immediately cease all activity within the HomeSafe's components, the user can manually power it down by pressing the POWER button.

5.3 Use Case 3: PIN Timeout

- **Actor**: User
- **Goal**: To access the HomeSafe despite encountering a PIN entry timeout and subsequent system lockout.
- **Preconditions**: The HomeSafe is configured with the user's PIN and Iris Scan and is powered off.
- **Trigger**: The user desires access to the HomeSafe but experiences interruption during PIN entry.
- **Scenario**: To access the HomeSafe, the user initiates the typical access protocol. However, they become subject to an unintended pause during PIN entry, initiating a timeout and subsequent challenges to regain access to the system.

  1. **Initiating Access**: The user powers up the HomeSafe and begins entering their 6-digit PIN. The LED display lights up, showing each character of the partly entered PIN as asterisks (*) while an internal counter begins silently ticking down in the background.
  2. **Experiencing PIN Entry Timeout**: Due to hesitation or interruption of over 3 seconds, the system enforces a timeout, resetting the PIN entry procedure and prompting the user again for the 6-digit PIN via the LED display.
  3. **Encountering Consecutive Timeout Penalties**: After three consecutive timeout events, the HomeSafe takes a precautionary security measure, transitioning into a "LOCKED OUT" state. The LED display, adopting a more stringent security protocol, now requests the entry of the Master PIN.
  4. **Providing Master PIN to Resolve Lockout**: In response to the "LOCKED OUT" state, the user enters the Master PIN as instructed. Successful verification removes the system from the lockout, reactivating standard access protocols.
  5. **Resuming Standard Access Procedure**: The LED display reverts to requesting the user's 6-digit PIN, offering another opportunity to access the HomeSafe by proceeding with the authorization process.

5.4 Use Case 4: Unauthorized Access (PIN)

- **Actor**: User
- **Goal**: To access the HomeSafe using established credentials while navigating through unauthorized access protocols upon multiple incorrect PIN entries.
- **Preconditions**: The HomeSafe is configured with the user's PIN and Iris Scan and is powered off.
- **Trigger**: The user intends to access the HomeSafe but fails to provide the correct PIN in multiple attempts.
- **Scenario**: The user, attempting to gain access to their HomeSafe, inadvertently fails to provide the correct PIN repeatedly. Subsequent incorrect entries enforce a system lockdown, prompting users to establish their legitimacy by entering the Master PIN as a security override.

1. **Initiating Access**: The user powers up the HomeSafe, engaging the microcontroller and keypad. The LED display prompts the user for their 6-digit PIN.
2. **Experiencing Failed PIN Entries**: The user enters an incorrect PIN and, upon pressing ENTER, is met with a failed verification message. This initiates an internal counter while the LED display prompts users to try their 6-digit PIN again.
3. **Encountering System Lockout due to Repeated Failures**: After three consecutive incorrect PIN entries, the HomeSafe, prioritizing security, transitions into a "LOCKED OUT" state. At this juncture, the LED display alters its request, seeking the 6-digit Master PIN to proceed.
4. **Overriding Lockout with Master PIN**: The user provides the Master PIN when requested. A successful entry recalibrates the system, lifting it from the "LOCKED OUT" state and reinstating the standard authorization protocols.
5. **Resuming Standard Access Procedure**: With the lockout condition removed, the LED display reverts to requesting the user's 6-digit PIN, allowing the user to proceed with the typical authorization process.

5.5 Use Case 5: Unauthorized Access (Iris Scan)

- **Actor**: User
- **Goal**: Gain access to the HomeSafe despite encountering challenges with Iris Scan verification, utilizing the Master PIN as an override upon failed attempts.
- **Preconditions**: The HomeSafe is configured with the user's PIN and Iris Scan and is powered off.
- **Trigger**: The user seeks access to the HomeSafe but encounters difficulty with the Iris Scan verification process.
- **Scenario**: Having previously established PIN and Iris Scan credentials, the user attempts to access the HomeSafe. A series of unsuccessful Iris Scan attempts propels the system into a secured "LOCKED OUT" state, prompting the user to affirm identity and intent via the Master PIN.

  1. **Initiating Access**: The user powers up the HomeSafe, engaging the microcontroller and keypad. The LED display prompts the user for their 6-digit PIN.
  2. **PIN Verification**: the user enters their personal 6-digit PIN and presses ENTER. In response, the LED display instructs the user to perform an Iris Scan, activating the scanner.
  3. **Unsuccessful Iris Scan Attempts**: The user attempts the Iris Scan, but upon being determined as invalid, the LED display persists in its request for a valid Iris Scan.
  4. **Encountering System Lockout due to Repeated Failures**: After three continuous failed Iris Scan verifications, the HomeSafe enters a "LOCKED OUT" state. In this phase, the LED display modifies its prompt, now demanding the 6-digit Master PIN to override the lockout.
  5. **Overriding Lockout with Master PIN**: The user provides the Master PIN when requested. A successful entry recalibrates the system, lifting it from the "LOCKED OUT" state and reinstating the standard authorization protocols.
  6. **Resuming Standard Access Procedure**: With the lockout condition removed, the LED display reverts to requesting the user's 6-digit PIN, allowing the user to proceed with the typical authorization process.

References

[1] Kathleen Garska. *Two-Factor Authentication (2FA) Explained: Biometric Authentication — blog.identityautomation.com* https://blog.identityautomation.com/mfa-face-off-series-biometric-authentication. [Accessed 27-AUG-2023].

[2] https://www.facebook.com/flaticon. *Safe Deposit free icons designed by Freepik — flaticon.com.* https://www.flaticon.com/free-icon/safe-deposit_3073524?term=safe&page=1&position=26&origin=search&related_id=3073524. [Accessed 25-AUG-2023].

[3] *Lucid visual collaboration suite: Log in — lucid.app.* https://lucid.app. [Accessed 09-SEP-2023].