# SneakerSecure System Requirements

May 2025

Justin Abraham

# Table of Contents:

# Customer Problem Statements & System Requirements

**Customer Problem Statement:**

“Sneaker collectors and enthusiasts have been dealing with increasing amounts of challenges regarding ensuring authenticity as the market has continued to grow over the years. Currently the market is flooded with counterfeits, compromising the value for collectors, and leading to financial loss for both the consumer and the company. Customers would like to see a system that is secure and can verify the authenticity of the sneakers, while also providing an ownership history for each item to support market value.”

**Glossary of Terms:**

*QR Code:* A machine-readable code used for storing URLs or other information, read by the camera of a smartphone.

*Authenticity Verification:* The process of confirming a product's legitimacy through technological means.

*Ownership History:* A record of all previous and current owners of a particular item.

**Functional Requirements:**

| no. | weight | desc, |
|---|---|---|
| **REQ -1** | **high** | **System must assign each pair of sneakers a unique blockchain ID<br>REQ-2 High** |
| **REQ -2** | **high** | **System must allow users to verify sneaker authenticity through QR codes** |

| | | |
|---|---|---|
| **REQ -3** | **high** | **Ownership history must be recorded securely and allow users to access this history** |

**Non-Functional Requirements:**

Reliability: The system should have a 24/7 uptime ideally, this is crucial to maintain user trust and provide a consistent experience.

Performance: Requests to receive blockchain data should be processed quickly, ensuring a smooth process when checking authenticity or ownership transfer.

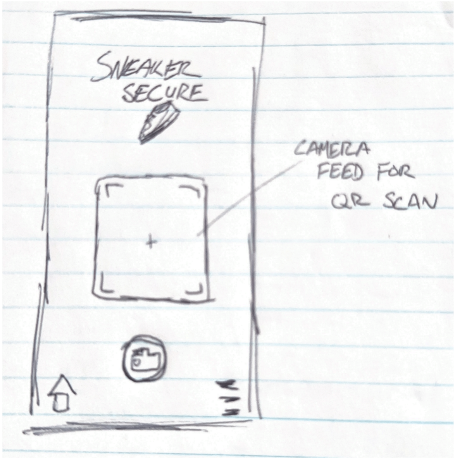
Usability: The interface should be simplistic and intuitive, allowing widespread adoption by all stakeholders.

Functionality: The system should perform all tasks with accuracy, and should store/retrieve data with no errors. This ensures the core objective of the system, which is maintaining authenticity, is achieved.

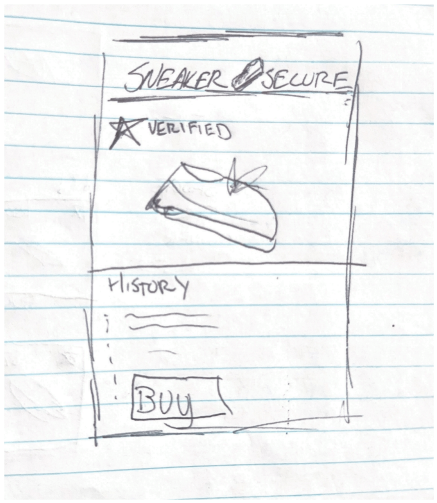
Supportability: Systems should be easy to update and maintain as blockchain features and security continue to evolve.

**User-Interface Requirements**

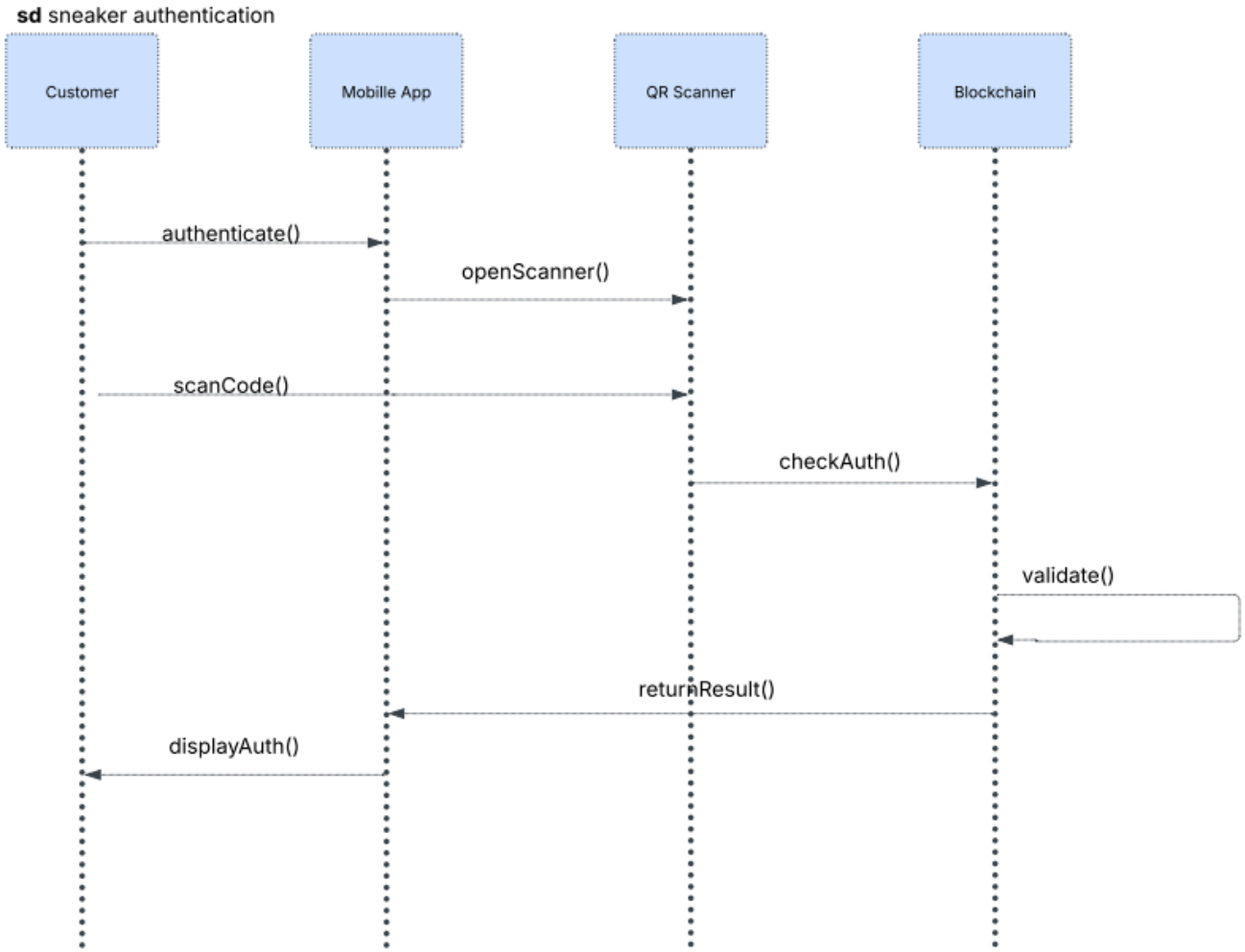
1. Home Screen/Landing Page:

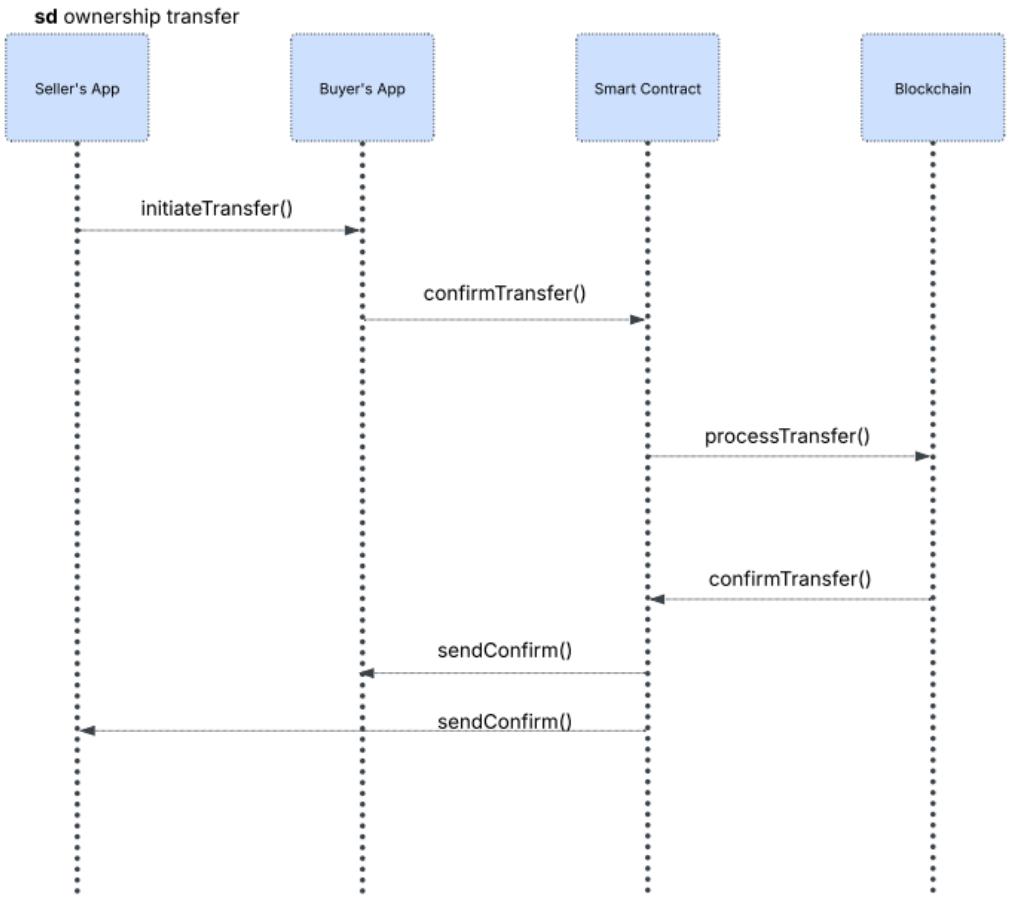



2. Sneaker Details Interface:


SNEAKER SECURE
VERIFIED
HISTORY
BUY

# System Sequence Diagram and Activity Diagram



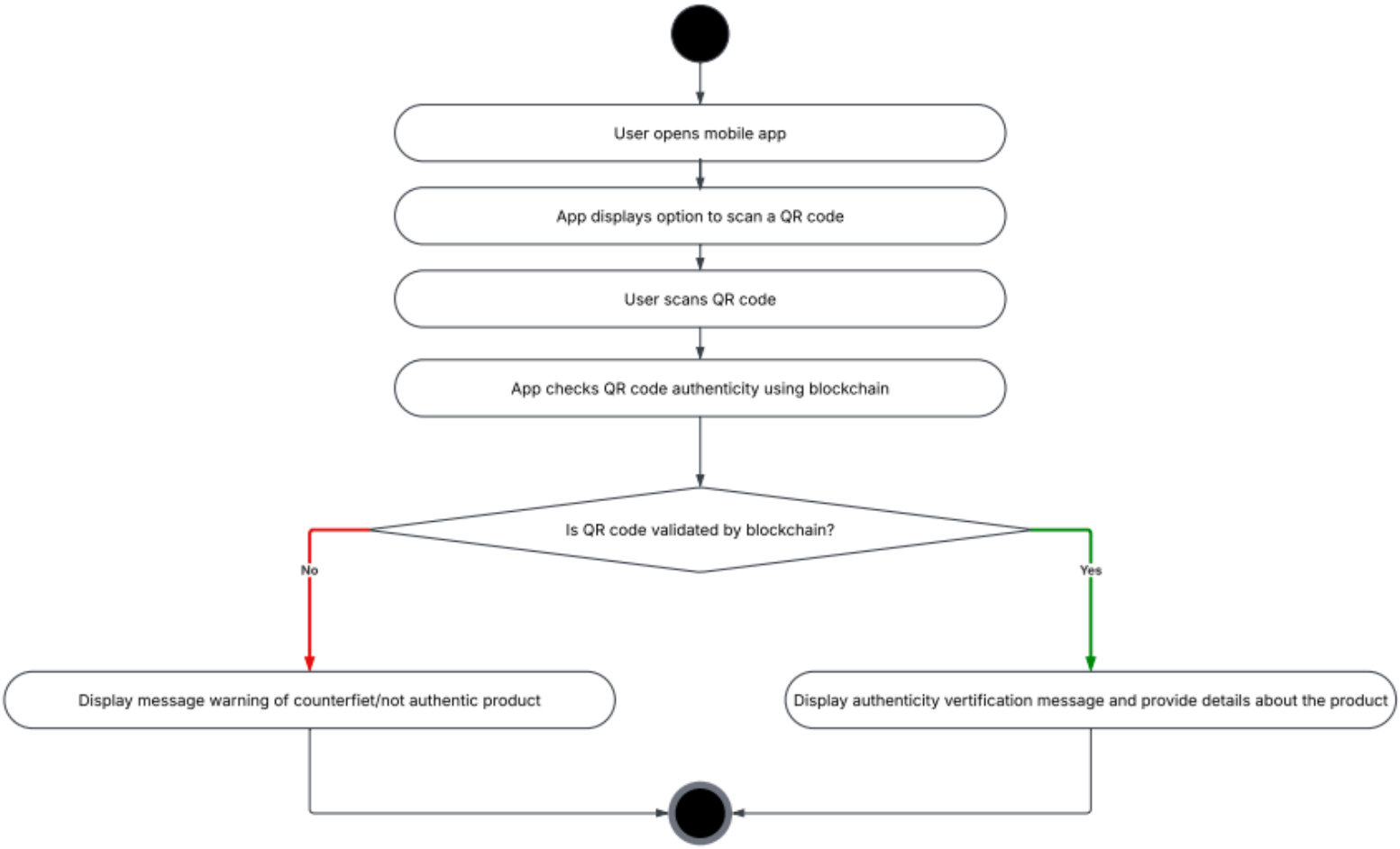

sd ownership transfer
Seller's App
Buyer's App
Smart Contract
Blockchain
initiateTransfer()
confirmTransfer()
processTransfer()
confirmTransfer()
sendConfirm()
sendConfirm()

**STATES**

- **Initial State:** The user initiates the authentication process.
- **Final States:**
  - The user receives confirmation of the sneaker's authenticity along with detailed ownership information.
  - The sneaker cannot be authenticated, and the user is informed of counterfeit status.

**ACTIONS**

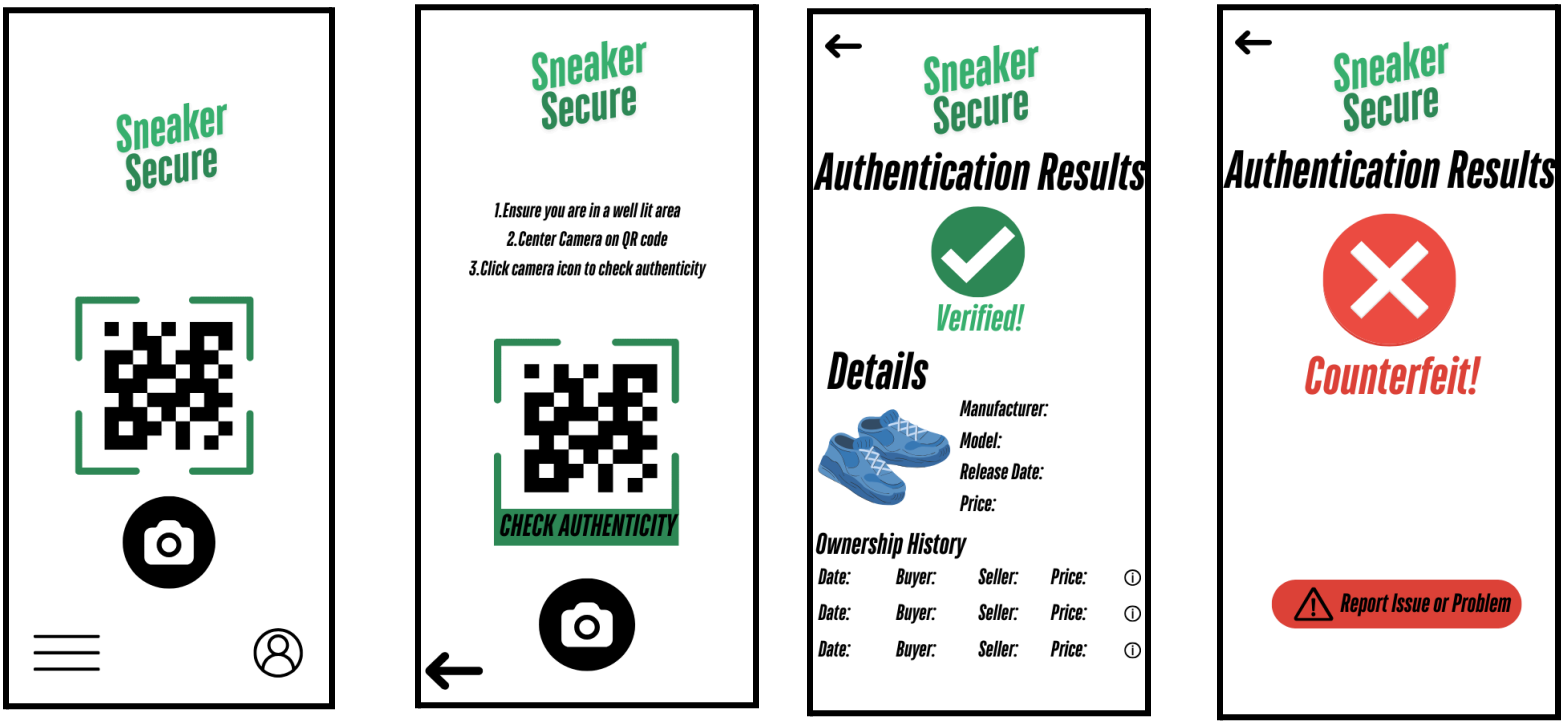
The user opens the mobile app, selects 'Authenticate Sneaker', scans the QR code on the sneaker, and the app sends the data to the blockchain network for validation. Depending on the blockchain's validation results, the app displays either the sneaker's authenticity and detailed information or alerts the user of potential counterfeiting.



# User Interface Specifications

## Preliminary Designs

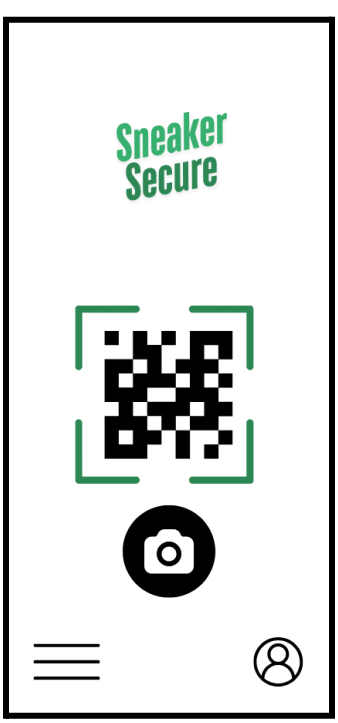
Use Case #1: User needs to authenticate sneakers they own/plan to buy

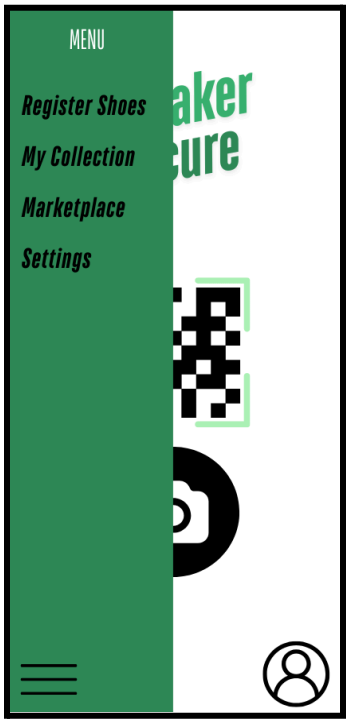



1.User clicks camera or QR code icon
2. Tips for taking QR code photo appear, user clicks camera icon to capture

3. Authentication results display verified if valid QR code, showing sneaker details as well
4.Authentication results display counterfeit if invalid QR code, option to report issue to support

## Use Case #2: User needs to register ownership for new pair of shoes





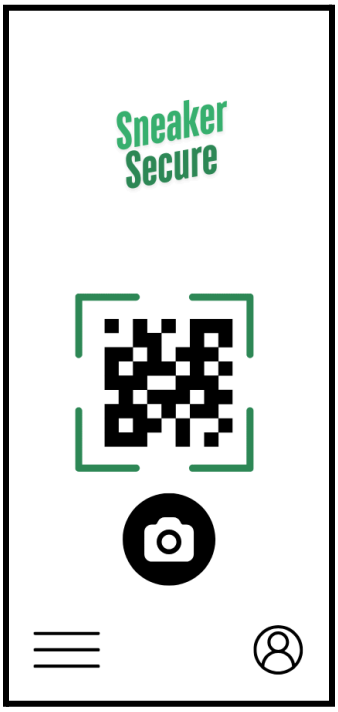

1. User clicks menu icon
2. User clicks “Register Shoes”User fills out registration form and uploads necessary files
3. User clicks confirm registration
4. Confirmation appears, allows user to add shoes to My Collection

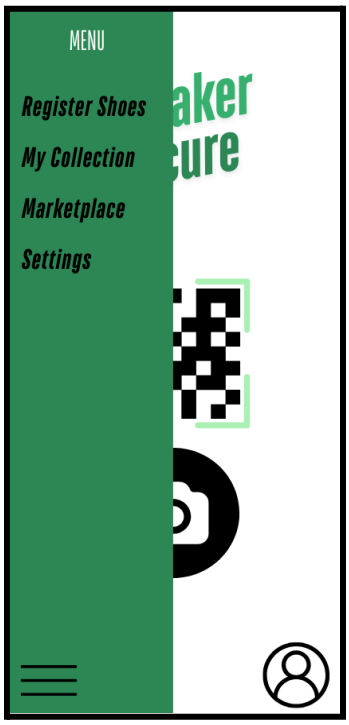
Use Case #3: User wants to browse on marketplace and purchase sneakers

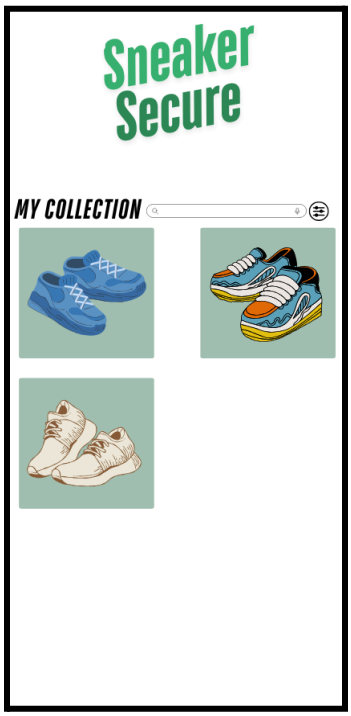

1. User clicks menu icon
2. User selects marketplace tab
3. User can browse for shoes and select one for purchase
4. Shoe details provided to user before purchase
5. User clicks buy now and fills out payment and shipping form

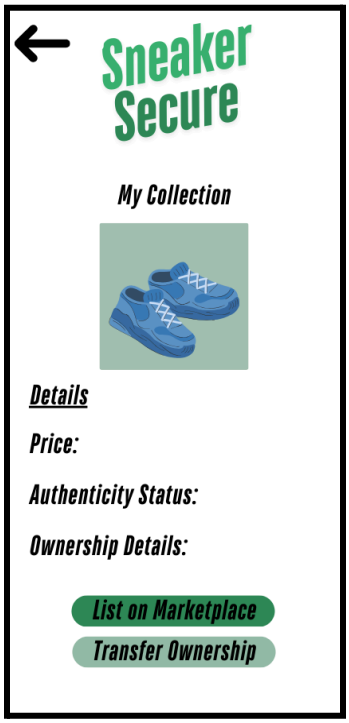
6. Receive confirmation

Use Case #4: User would like to transfer ownership on purchase made outside of marketplace

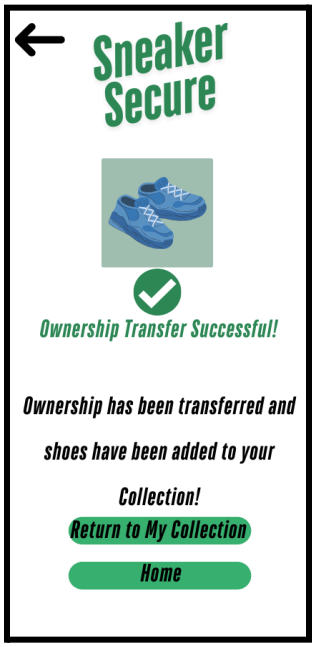


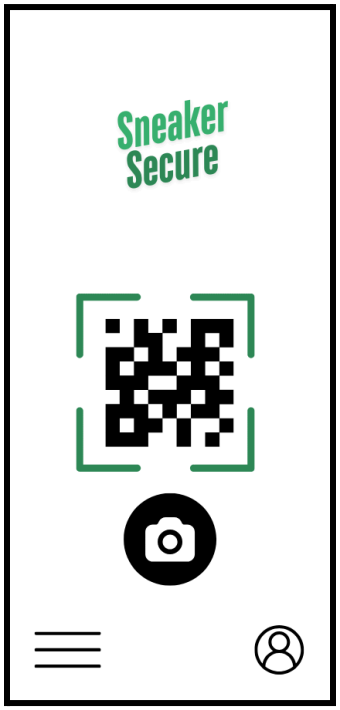





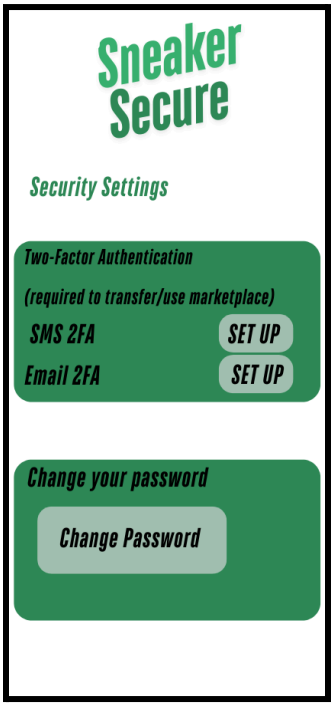


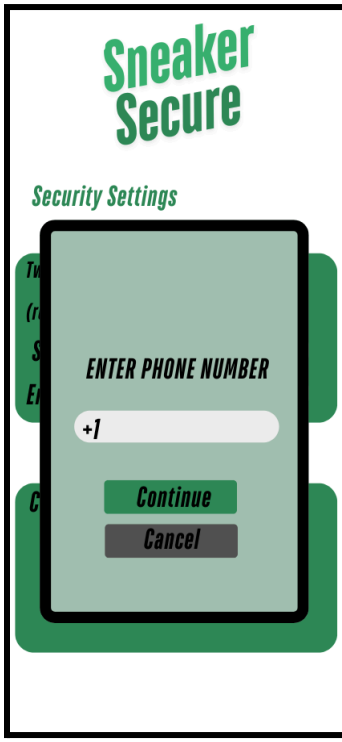
1. User clicks menu icon
2. User selects My Collection
3. User can view each sneaker in collection, shows details as well as option to list for sale or transfer ownership
4. User selects transfer ownership
5. User fills out recipient details form and presses submit
6. Receive confirmation

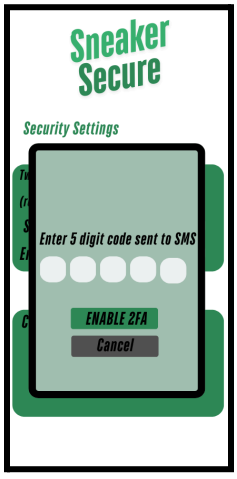
Use Case #5: <u>User wants to edit profile to update security settings and enable 2 Factor Authentication</u>

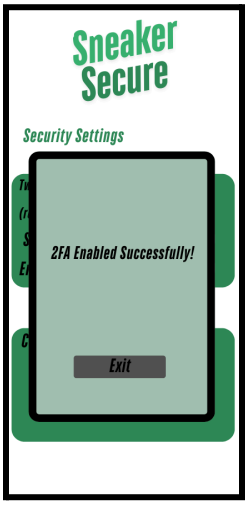






1. User selects user icon on home page
2. User selects security settings
3. User selects “Set up” for SMS 2FA
4. User enters phone number
5. User enters code sent to phone number
6. Receive confirmation

**User Effort Estimation**

| USAGE SCENARIO | NAVIGATION | CLICKS | KEYSTOKES |
|---|---|---|---|
| User needs to authenticate sneakers they own/want to buy | Home, QR capture, confirmation | 2 | 0 |
| User needs to register ownership for new pair of shoes | Home, menu, register shoes, registration form, confirmation | 8 | <30 |
| User wants to browse on marketplace and purchase sneakers | Home, menu, marketplace, sneaker details, payment and shipping details, confirmation | 12 | <100 |

| | | | |
|---|---|---|---|
| User would like to transfer ownership on purchase made outside of marketplace | Home, collection, sneaker details, transfer, recipient form, confirmation | 9 | <50 |
| User wants to edit profile to update security settings and enable 2 Factor Authenticatio n | Home, account, security settings, 2FA, confirmation | 7 | <20 |

# Traceability Matrix

| Req ID | Requirement Description | Component | File Location | Test Case(s) |
|---|---|---|---|---|
| **REQ-001** | **User Login/Authentication** | **LoginScreen** | **LoginScreen.js** | **TC-001: Login with valid credentials<br>TC-002: Admin login validation<br>TC-003: Current User (Unit Test) in Testing.js** |
| **REQ-002** | **QR Code Scanning** | **QR Scanner Camera** | **HomeScreen.js** | **TC-004: Scanning valid QR code<br>TC-005: Handling invalid QR format** |
| **REQ-003** | **Sneaker Verification** | **UUID Verification** | **uuid_database.js** | **TC-006: Test UUID Verification (Integration Test) in Testing.js<br>TC-007: Verify authentic sneaker UUID<br>TC-008: Reject counterfeit sneaker UUID** |
| **REQ-004** | **Sneaker Details Display** | **SneakerDetail** | **SneakerDetailScreen.js** | **TC-009: Display all sneaker information<br>TC-010: Gallery navigation<br>TC-011: History display** |

| | | | | |
|---|---|---|---|---|
| **REQ-005** | **Collection Management** | **MyCollectionScreen** | **DatabaseHelper.js** | **TC-012: Test Collection Operations (Data Collection Test) in Testing.js<br>TC-013: Add sneaker to collection<br>TC-014: Remove sneaker from collection** |
| **REQ-006** | **Admin Functionality** | **EditModal, Testing** | **Testing.js** | **TC-015: Admin edit sneaker details<br>TC-016: Admin verification capabilities<br>TC-017: Admin testing functions** |
| **REQ-007** | **Data Persistence** | **AsyncStorage** | **database/DatabaseHelper.js, QRGen.js** | **TC-018: View All Sneakers test in Testing.js<br>TC-019: Data survival across app restarts<br>TC-020: Collection persistence** |
| **REQ-008** | **Sneaker Information Storage** | **Sneaker Data Model** | **qrcodes/sneakerDetails.json, QRGen.js** | **TC-021: Proper formatting of sneaker data<br>TC-022: Accessing stored sneaker details** |
| **REQ-009** | **Ownership History Tracking** | **History Component** | **SneakerDetailScreen.js** | **TC-023: Display ownership chain<br>TC-024: Validate history records** |
| **REQ-010** | **UI Components** | **Badge Components** | **UnverifiedBadge.js** | **TC-025: Verified badge display<br>TC-026: Unverified badge display** |

## Implementation Evidence

User Authentication (REQ-001):

Implemented in LoginScreen.js with username storage in AsyncStorage
Admin check based on username in multiple screens
Test function in Testing.js: "Current User (Unit Test)"

QR Code Scanning (REQ-002):

Implemented with expo-camera and barcode scanning in HomeScreen.js
Handles scanned QR data and navigates to details

Sneaker Verification (REQ-003):

Verification logic in uuid_database.js with predefined verification list
Integration test in Testing.js: "Test UUID Verification (Integration Test)"
Visual indicators via VerifiedBadge and UnverifiedBadge components

Collection Management (REQ-005):

Database operations in DatabaseHelper.js
User interface in MyCollectionScreen.js
Test in Testing.js: "Test Collection Operations (Data Collection Test)"

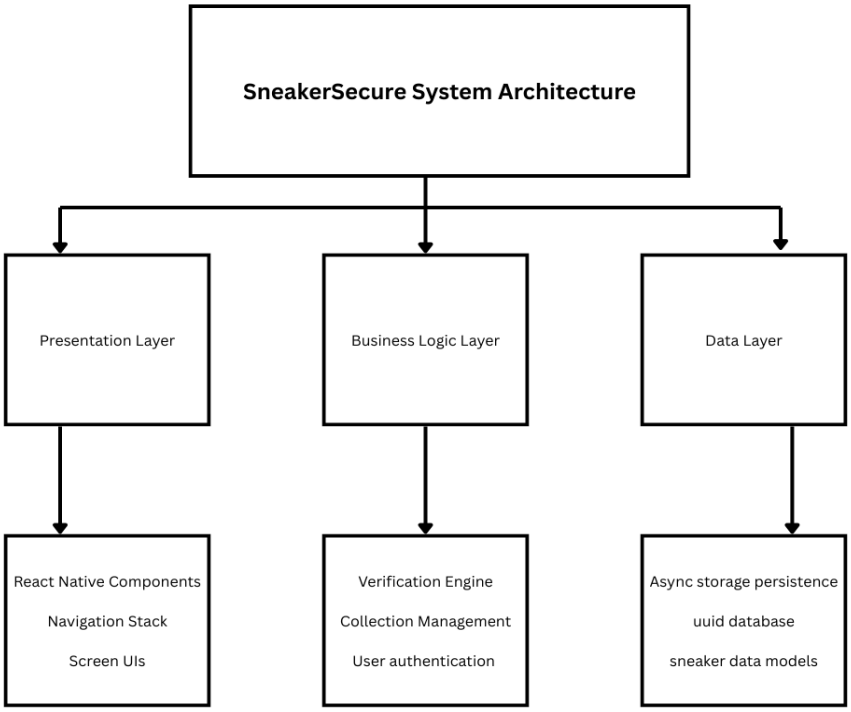
Admin Functions (REQ-006):

Admin-only edit functionality in SneakerDetailScreen.js
Admin testing screen with diagnostic functions

Data Persistence (REQ-007):

AsyncStorage implementation throughout the application
Migration functions in App.js
Database helper functions in DatabaseHelper.js

# System Architecture




**Component Architecture**
The application uses a component-based architecture with the following key components:

**Navigation System**

Stack-based navigation between screens
Conditional rendering based on authentication state

**Screen Components**

LoginScreen: User authentication entry point
HomeScreen: Main dashboard with QR scanning capability
SneakerDetailScreen: Displays verified sneaker information
MyCollectionScreen: User's collected sneakers
Testing: Admin testing functionality

**Data Components**

UUID Verification: Authenticates sneaker identifiers
Database Helper: Manages local data storage operations
QR Generator: Creates test QR codes for development

**UI Components**

VerifiedBadge/UnverifiedBadge: Status indicators
Modal components for editing and confirmation

# Key Algorithms and Data Structure

### Verification Algorithm

```
// Based on uuid_database.js implementation
function checkUUIDVerification(uuid) {
  // 1. Normalize input
  const normalizedUUID = uuid.trim().toLowerCase();

  // 2. Check against database of verified UUIDs
  const isVerified = VERIFIED_UUIDS.includes(normalizedUUID);

  // 3. Log verification attempt
  console.log(`Verification check for UUID ${normalizedUUID}: ${isVerified ? 'Verified' :
'Not verified'}`);

  // 4. Return verification status
  return isVerified;
```

### Collection Management Algorithm

```
// Collection Retrieval Algorithm
static async getCollection() {
  try {
    // 1. Get all keys from AsyncStorage
    const keys = await AsyncStorage.getAllKeys();

    // 2. Filter for collection keys only (using prefix)
    const collectionKeys = keys.filter(key => key.startsWith(this.DB_PREFIX));

    // 3. Early return if collection is empty
    if (collectionKeys.length === 0) {
      return [];
    }

    // 4. Batch retrieve all collection items
    const items = await AsyncStorage.multiGet(collectionKeys);

    // 5. Process and parse each item
    return items
```

```
      .map(([key, value]) => {
        if (!value) return null;
        try {
          return JSON.parse(value);
        } catch (e) {
          console.error("Error parsing item:", e);
          return null;
        }
      })
      // 6. Filter out any null/invalid items
      .filter(item => item !== null);
  } catch (error) {
    console.error("Error getting collection:", error);
    return [];
  }
}

// Collection Addition Algorithm
static async addToCollection(sneakerData) {
  try {
    // 1. Validate sneaker data
    if (!sneakerData || !sneakerData.id) {
      throw new Error("Invalid sneaker data");
    }

    // 2. Create storage key with prefix
    const itemKey = `${this.DB_PREFIX}${sneakerData.id}`;

    // 3. Format data for storage
    const itemData = {
      id: sneakerData.id,
      name: sneakerData.name || "Unknown Sneaker",
      description: sneakerData.description || "",
      imageUrl: sneakerData.imageUrl || "",
      added_at: new Date().toISOString()
    };

    // 4. Store in AsyncStorage
    await AsyncStorage.setItem(itemKey, JSON.stringify(itemData));
    console.log("Added item to collection:", itemData.name);
```

```
    return itemData.id;
  } catch (error) {
    console.error("Error adding to collection:", error);
    throw error;
  }
}

// Collection Removal Algorithm
static async deleteFromCollection(id) {
  try {
    // 1. Validate ID
    if (!id) {
      throw new Error("Invalid ID for deletion");
    }

    // 2. Create storage key with prefix
    const itemKey = `${this.DB_PREFIX}${id}`;

    // 3. Remove item from AsyncStorage
    await AsyncStorage.removeItem(itemKey);
    console.log("Removed item from collection:", id);
    return true;
  } catch (error) {
    console.error("Error deleting from collection:", error);
    throw error;
  }
}
```

## Data Structure

Example of Collection Item data structure:

```
interface CollectionItem {
  id: string;              // References Sneaker.id
  name: string;            // Cached name for display
  description: string;     // Cached description
  imageUrl: string;        // Cached image URL
  added_at: string;        // timestamp of collection addition
}
```

## Storage Architecture

SneakerSecure implements a namespaced key-value storage pattern using AsyncStorage.

| AsyncStorage | |
|---|---|
| Key | Value (JSON parsed) |
| db_collection_[sneakerID] | CollectionItem |
| allSneakers | array<sneakers> |
| username | string |

The application uses a consistent prefix system to namespace collection items (db_collection_) and enable efficient filtering of AsyncStorage keys.

# User Interface Design and Implementation

## UI Implementation Approach

SneakerSecure follows a mobile-first design approach using React Native components:

Navigation Structure:

- Stack Navigator for primary navigation flow
- Consistent header styling across screens
- Back navigation where appropriate

Screen Layout Strategy:

- Flex-based responsive layouts
- ScrollView for content overflow
- Consistent padding and spacing

UI Component Hierarchy:

-SneakerSecure
    -LoginScreen
    -HomeScreen
        –Camera/QR Scanner Component
    -SneakerDetailScreen
        –VerifiedBadge/UnverifiedBadge
        –Image Component
        –Details Section
        –History Section
        –Edit Modal (Admin only)
    -MyCollectionScreen
        –Collection Item Components
    -Testing Screen (Admin only)
        –Test Function Buttons

## Design Implementation

Visual Style:

- Consistent color scheme across the application
- Typography hierarchy for readability
- Status indicators (badges) for verification status

Interactive Elements:

- Touchable components with appropriate feedback
- Form inputs with validation feedback

Responsive Considerations:

- Flexible layouts that adapt to different screen sizes
- Text wrapping for varying content lengths

## UI Testing Approach

The application implements manual UI testing through visual verification:

Visual Inspection Tests:

- Verify UI components render as expected
- Confirm responsive behavior on different screen sizes
- Validate color scheme and typography

Interactive Testing:

- Verify touch interactions produce expected results
- Confirm navigation flows work correctly
- Test form inputs and validation

Edge Case Handling:

- Test behavior with empty collections
- Verify error states are handled
- Confirm long text wraps appropriately

# Project Plan

## Roadmap:

- Implement ownership transfer details
- Implement new shoe registration and qr code generation for new shoes
- Implement security measures such as 2FA
- Implement more data editing capabilities once sneaker is owned
- Polish designs and formatting