Requirements Analysis – Gamer CV App

# Introduction

The "Gamer CV" is a mobile application developed as part of a secure software engineering minor, focusing on cryptographic best practices and modern privacy-aware mobile design. The application allows users to aggregate and display public statistics from multiple games into a secure, tamper-proof, device-signed persona profile.

Unlike traditional stat aggregators, this app emphasizes data sovereignty and security, removing the need for centralized accounts or cloud storage. Every record is signed using a hardware-backed Ed25519 keypair and optionally encrypted, ensuring integrity, privacy, and long-term verification — even offline.

This pivot away from relying on brittle or deprecated APIs allows the team to focus on security-focused design, secure coding standards, and optimal user control over their gaming data. Optional features such as wellness notifications further enhance the app’s value without compromising its core privacy principles.

This document details the functional, non-functional, and security-focused requirements, and outlines both the SDLC and SSDLC approaches adopted throughout development.

# MVP Overview

Goal: Aggregate public game stats into a secure, tamper-proof player résumé.  
Focus: Strong on-device cryptography, offline-first support, no traditional user accounts.

# Functional Requirements

## Game Stats Aggregation

* Fetch public stats via third-party game APIs.
* Aggregate per-game, per-genre, and globally.
* Support multiple games per persona.

## Persona System

* Generate & store Ed25519 keypair using hardware-backed keystore (e.g., Android Keystore).
* Use the private key to sign persona records.
* Encrypt stats and metadata on-device.

## Username & Game Management

* Add/edit/remove in-game usernames per supported title.
* Validate username format and check for duplicates.
* Maintain sync state and timestamps per record.

## Stats Comparison

* Allow secure stat comparison with friends via a shared link or QR Code.
* Show side-by-side comparisons across games and genres.

## Wellness Dashboard (Optional)

* Detect late-night or excessive play patterns.
* Alert user with optional push/local notifications.

# Front-End: Android App (Kotlin + Jetpack Compose)

## Key Screens

* Home / Dashboard - Aggregated stats summary, Recently updated games
* Game Browser - Search/filter supported games
* Stats Viewer - Core metrics per game, Genre and global aggregations
* Friends Compare - Securely share persona, Compare signed stats
* Settings & Profile - Edit avatar, display name, favorite genres
* Wellness UI - Optional visualizations (bar graphs, alerts)

## Local Storage

* Room DB or Encrypted SharedPreferences for:
  + Stats cache
  + Game list metadata
  + User settings & preferences
  + Signed persona records

## Local Crypto

* Android Keystore for Ed25519 keypair
* Tink or Bouncy Castle for AES-256, EdDSA ops

## Removed / Reconsidered (Legacy Architecture)

* PostgreSQL backend — Replaced by local encrypted storage
* RESTful backend API — Replaced by direct frontend API access
* Redis / server caching — Stats are cached locally and invalidated via TTL
* Traditional login/signup — Replaced with local keypair and optional backup/export

# Non-function Requirements

## Security Requirements

* Key Generation: Ed25519 via Android Keystore (non-exportable private key)
* Encryption at Rest: AES-256-GCM using Tink or AndroidX Security
* Encryption in Transit: HTTPS + TLS 1.3 with cert pinning for all API calls
* Tamper-proof History: Persona records signed with private key
* Data Backup (optional): Exportable encrypted JSON payload of persona

## Performance Requirements

* App should load local stats within 300~ms.
* Sync with public APIs must complete within 5 seconds under normal network conditions.

## Availability Requirements

* App must be fully useable offline with cached stats and sync when reconnected.
* Core features (stats viewing, persona generation) must not depend on live API access.

## Usability Requirements

* App must follow Material Design guidelines and be fully navigable using accessible gestures.
* Compose UI should adapt seamlessly across devices.

## Maintainability Requirements

* Code must follow clean architecture principles, and all cryptographic operations should be modular and testable.
* All dependencies must be open-source or MIT/BSD-licensed when possible.

## Reliability Requirements

* Stats syncing must retry on failure with exponential backoff.
* The app must gracefully handle corrupted or tampered local data (signature invalidation).

## Notifications (Optional)

* Local notifications for playtime alerts
* Firebase Cloud Messaging (FCM) for wellness push alerts (opt-in)

## Data Sync Strategy

* Offline-first: Use cached stats from last fetch
* TTL-based Invalidation: Refresh only if data is stale
* Sync Queue: Batch failed fetches for retry when online

## To Research / Evaluate

|  |  |
| --- | --- |
| Topic | Options |
| Game API Providers | Native APIs, RAWG, OpenDota, Riot API, OSRS Hiscore |
| Wellness Detection | Timestamp-based usage heuristics |
| Charting | MPAndroidChart, Compose Charts |
| Key Management | Android Keystore (strongbox), Libsodium |
| Persona Sharing | QR Code (ZXing), Near Share, NFC |

# **Secure Software Development Life Cycle (SSDLC)**

