SOFTWARE DESIGN SPECIFICATION

Project: GiveOne — $1-a-day Donation App

Author: Ahmed Shady

Course: CIS 434 — Software Engineering

Date: October 2025

1.0 Introduction

This document describes all data, architectural, interface, and component-level design for the GiveOne system. It serves as the bridge between the Software Requirements Specification (SRS) and the implementation phase, defining the internal design and structural blueprint of the system.

1.1 Goals and objectives

The goal of GiveOne is to provide a simple, transparent donation experience where users can contribute $1 at a time to verified charitable cases. The software enables donors to manage their wallets, track progress on donation cases, and receive completion updates once goals are met.

Primary objectives:

- Minimize barriers to donation through one-tap $1 contributions.
- Provide transparent progress tracking and donor history.
- Design a modular, scalable architecture for future enhancements (e.g., auto-donate, reminders).

1.2 Statement of scope

Inputs: user wallet deposits, donation selections, organization case data.
Processing: validation, wallet deduction, case progress updates, transaction logging.
Outputs: updated progress bars, donation receipts, history tables, and final completion updates.

1.3 Software context

GiveOne operates as a standalone **Python/Tkinter** desktop application. It stores all data locally using either **SQLite** or **JSON**, without requiring an internet connection. The project contributes to the broader initiative of micro-philanthropy—encouraging small, frequent donations with maximum impact.

1.4 Major constraints

- Solo developer project with limited time and scope.
- Offline operation only (no backend server).
- Python + Tkinter UI framework only.
- Data persistence via JSON or SQLite.

2.0 Data design

This section defines the data structures and database models used in GiveOne..

2.1 Data structures

Structure	Description	Used By
User	Stores donor account info	WalletManager
Wallet	Maintains current balance	WalletManager
Case	Represents donation opportunities	CaseManager
Donation	Records each $1 transaction	DonationManager
Update	Holds organization updates	CaseManager

2.2 Database description

GiveOne uses a lightweight local database (**SQLite**) or a structured JSON file

Tables:

- `users(user_id, name, email, created_at)`
- `wallets(wallet_id, user_id, balance, last_updated)`
- `cases(case_id, title, goal_amount, raised_amount, status, org_name)`
- `donations(donation_id, user_id, case_id, amount, timestamp)`
- `updates(update_id, case_id, type, content, timestamp)`

Relationships:

- User 1—1 Wallet
- User 1—* Donation

- Case 1—* Donation
- Case 1—* Update

3.0 Architectural and component-level design

This section explains the architecture, components, and interactions between modules in the system.

3.1 Architecture diagrams

UI Layer

(Tkinter Windows, Buttons)

↓

Application Logic Layer

| (WalletManager, CaseManager, DonationManager) |

↓

Data Access Layer

(StorageHandler: JSON/SQLite)

Process View

1. User opens app → home screen loads cases.
2. Donor adds funds → wallet balance updates.
3. Donor clicks “Donate $1” → transaction logged → case progress updates.
4. UI refreshes to display new balance and case status.

Physical View

Single-machine desktop setup (no client-server).

3.2 Description for Components

3.2.1 Component n description

Purpose: Handles all user interaction and displays results.
Interface Description: Tkinter buttons, inputs, and display frames.
Static Models: `MainWindow`, `CaseListFrame`, `WalletFrame`, `HistoryFrame`.
Dynamic Models: User event triggers → calls application logic methods → updates interface.

3.2.1.1 Interface description

3.2.2 Component 2 — Business Logic Layer

Purpose: Processes core functionality and enforces business rules.
Interface Description:

- Inputs: wallet actions, donation selections.
- Outputs: balance updates, donation logs, case status changes.
  Static Models:
- `WalletManager`: handles balance validation and updates.
- `DonationManager`: manages $1 donations and logging.
- `CaseManager`: loads and modifies case progress.
  Dynamic Models (Activity Flow):

[User clicks Donate $1]

↓

[Check Wallet ≥ $1?]

↓ (if yes)

[Deduct $1]

↓

[Update Case progress]

↓

[Record Donation]

↓

[Update UI]

3.2.3 Component 3 — Data Layer

Purpose: Manage local data persistence and retrieval.
Interface Description: file I/O or SQL read/write operations.
Static Models:

- `StorageHandler`: save/load JSON or SQLite data.
- `BackupManager`: handles data recovery.
  Dynamic Models: triggered after every transaction to write updated state.

3.3 External Interface Description

There are **no external interfaces** for the MVP.
Future versions may include payment gateway or cloud sync APIs.

4.0 User interface design

A description of the user interface design of the software is presented.

4.1 Description of the user interface

Main Screens:

1. **Home / Cases Screen** – Displays available cases with progress bars and "Donate $1" buttons.
2. **Wallet Screen** – Shows current balance and "Add Funds" field.
3. **History Screen** – Displays past donations with timestamps and running balance.
4. **Case Detail Screen** – Shows full description and updates from organizations.

Sample Mockup (Text Sketch):

| GiveOne |

| [Wallet: $25] [Add Funds] [History] |

| Case: Rebuild a Home [Donate $1] |

| Goal: $500 | Raised: $420 |

| Case: Feed a Family [Donate $1] ✅ |

| Goal: $300 | Raised: $300 |

4.2 Interface design rules

- Simple and minimal layout using Tkinter.
- Consistent button placement and color coding.
- Accessibility: large fonts and clear spacing.
- Color scheme:
  - Blue → Action
  - Green → Success
  - Red → Errors

5.0 Restrictions, limitations, and constraints

- No internet connection required.
- Single-user offline operation.
- No cross-device synchronization.
- Limited to Python's Tkinter UI framework.
- Future cloud integration is out of scope.

6.0 Appendices

Presents information that supplements the design specification.

6.1 Requirements traceability matrix

SRS Requirement ID	Design Component	Implementation Module
FR-1 (Browse Cases)	UI Layer	CaseListFrame

FR-2 (Wallet)	Business Logic	wallet.py
FR-3 (Donate $1)	Logic Layer	donation.py
FR-4 (History)	UI + Logic Layer	history.py
FR-5 (Case Updates)	Data Layer	case.py
FR-6 (Persistence)	Data Layer	storage.py

6.2 Implementation issues

- SQLite vs JSON: final choice depends on testing.
- Tkinter’s responsiveness improved by asynchronous UI updates.
- Future work: integrate notifications and recurring donations.