

*SYSTEM DESIGN DOCUMENT*

**TEAM MEMBERS**: *Luciano Corvino, Cristian Carotenuto, Maria Chiara Gregorio, Vito Francesco Maistrini, Giovanni De Caro, Gabriele Milone, Carlo Antonio Caserta, Nicola Luciano, Giovanni Esposito, Riccardo Di Girolamo*

**PROJECT MANAGER**: *Emanuele D'Auria, Antonio Botticchio*

Sommario

[**1. DESIGN GOALS** 3](#_Toc221960852)

[**2. TRADE-OFF** 4](#_Toc221960853)

[2.1 Usabilità vs Sicurezza 4](#_Toc221960854)

[2.1.2 Scelte di sicurezza implementate 4](#_Toc221960855)

[2.1.3 Impatto sull’usabilità 4](#_Toc221960856)

[2.1.4 Motivazione del compromesso 4](#_Toc221960857)

[2.2 Privacy vs Affidabilità 5](#_Toc221960858)

[2.2.1 Scelte che privilegiano l’affidabilità 5](#_Toc221960859)

[2.2.2 Misure adottate per mitigare l’impatto sulla privacy 5](#_Toc221960860)

[2.2.3 Motivazione del compromesso 5](#_Toc221960861)

[2.3 Conclusione 6](#_Toc221960862)

[**3. ARCHITETTURA DEL SISTEMA** 7](#_Toc221960863)

[3.1 Introduzione al Sistema 7](#_Toc221960864)

[3.2 Architettura Three‑Tier 7](#_Toc221960865)

[3.2.1 Presentation Tier – Frontend Web 7](#_Toc221960866)

[3.2.1.1 Responsabilità del frontend 7](#_Toc221960867)

[3.2.2 Application Tier – Backend Flask 8](#_Toc221960868)

[3.2.2.1 Responsabilità del backend 8](#_Toc221960869)

[3.2.3 Data Tier – Database SQLite + SQLAlchemy ORM 8](#_Toc221960870)

[3.2.3.1 Responsabilità del Data Tier 9](#_Toc221960871)

[**3.4 Comunicazione tra i Livelli 9**](#_Toc221960872)

[**3.5 Deployment 9**](#_Toc221960873)

[**3.6 Conclusione 10**](#_Toc221960874)

# 

# DESIGN GOALS

Lo sviluppo di **MAP4AID** si basa su una serie di principi cardine, definiti per garantire l'efficacia del sistema e la tutela degli utenti. Di seguito vengono illustrati i sei obiettivi di design che guidano l'intero progetto:

1. **Usabilità del sistema**

* Le interfacce del sistema devono essere chiare e comprensibili

1. **Sicurezza**

* Il sistema deve trattare i dati degli utenti in modo sicuro

1. **Privacy**

* Il sistema deve garantire la privacy degli utenti chiedendo solo dati essenziali

1. **Accessibilità**

* Il sistema deve essere accessibile da diversi dispositivi

1. **Affidabilità**

* Il sistema deve garantire la correttezza delle informazioni sui beni disponibili.

1. **Scalabilità**

* Il sistema deve poter gestire un gran numero di utenti e beni in modo efficiente

# 2. TRADE-OFF

## 2.1 Usabilità vs Sicurezza

Nel progetto Map4Aid questo trade‑off è stato affrontato privilegiando la sicurezza, pur mantenendo un livello di usabilità adeguato al contesto sociale dell’applicazione.

### 2.1.2 Scelte di sicurezza implementate

**Registrazione con OTP**: ogni nuovo account deve confermare la propria identità tramite codice monouso inviato via email.

* Questo introduce un passaggio aggiuntivo per l’utente, ma garantisce che l’indirizzo email sia reale e controllato.

**Upload del documento d’identità** per beneficiari e enti erogatori:

* aumenta la complessità della registrazione, ma permette di verificare l’autenticità dei profili e prevenire abusi.

**Ruoli e permessi rigidi** (require\_roles, require\_admin):

* ogni endpoint è accessibile solo alle categorie autorizzate, riducendo la superficie di attacco ma aumentando la complessità di gestione.

**Sessioni server‑side**:

* scelta più sicura rispetto ai token client‑side, ma meno comoda per l’utente e meno scalabile.

### 2.1.3 Impatto sull’usabilità

* La registrazione richiede più passaggi rispetto a un’app tradizionale.
* L’utente deve caricare documenti e attendere approvazione.
* Alcune operazioni (es. prenotazione medicinali) richiedono conferme multiple.

### 2.1.4 Motivazione del compromesso

Il sistema gestisce:

* dati sensibili (documenti, patologie, allergeni)
* prenotazioni di beni essenziali
* medicinali con ricetta
* enti verificati

In questo contesto, la sicurezza non è un optional ma un requisito fondamentale.

Il compromesso scelto è quindi:

**Sicurezza elevata, anche a costo di una minore immediatezza nell’esperienza utente.**

## 2.2 Privacy vs Affidabilità

Trade off più evidente nel progetto, Map4Aid deve garantire che:

* i beneficiari siano reali
* gli enti siano verificati
* le prenotazioni siano corrette
* i medicinali siano distribuiti solo a chi ne ha diritto

Per farlo, il sistema deve raccogliere dati che normalmente non verrebbero richiesti in un’app generica.

### 2.2.1 Scelte che privilegiano l’affidabilità

* **Raccolta di dati sanitari** (allergeni, patologie) per migliorare la qualità del servizio.
* **Upload della carta d’identità** per verificare l’identità del beneficiario.
* **Upload della ricetta medica** per i medicinali.
* **Geolocalizzazione precisa** per inviare segnalazioni agli enti più vicini.
* **Stati delle prenotazioni** (in\_validazione → in\_attesa → ritirata) per garantire tracciabilità e correttezza.

Questi elementi aumentano l’affidabilità del sistema, ma riducono la privacy dell’utente.

### 2.2.2 Misure adottate per mitigare l’impatto sulla privacy

* Raccolta **solo dei dati strettamente necessari** al servizio.
* Nessuna condivisione dei dati con terze parti.
* Documenti salvati con nomi univoci e non riconducibili direttamente all’utente.
* Accesso ai dati limitato tramite ruoli e permessi.
* Nessuna esposizione dei dati sensibili nelle API pubbliche.

### 2.2.3 Motivazione del compromesso

Il sistema deve garantire che:

* i beni vengano distribuiti correttamente
* i medicinali siano consegnati solo a chi ha una ricetta valida
* gli enti possano verificare l’identità dei beneficiari

Per questo motivo, la privacy non può essere assoluta:

**una parte dei dati sensibili è necessaria per garantire affidabilità e correttezza del servizio.**

# 2.3 Conclusione

Nel complesso, Map4Aid adotta scelte progettuali che privilegiano:

* **Sicurezza** rispetto alla pura usabilità
* **Affidabilità** rispetto alla privacy assoluta

Questi compromessi sono coerenti con la natura del sistema, che gestisce:

* beneficiari reali
* enti verificati
* medicinali
* documenti sensibili
* prenotazioni critiche

richiede quindi un livello di controllo superiore rispetto a un’applicazione consumer tradizionale, essendo infatti un servizio prettamente aid.

Il compromesso scelto è:

**Affidabilità e correttezza dei processi > Minimizzazione assoluta dei dati**

pur mantenendo un livello di privacy adeguato e conforme al principio di necessità.

# 3. ARCHITETTURA DEL SISTEMA

## 3.1 Introduzione al Sistema

Map4Aid è una piattaforma web progettata per supportare la distribuzione di beni essenziali e medicinali a persone in difficoltà, facilitando la collaborazione tra beneficiari, donatori ed enti erogatori.

Il sistema gestisce processi critici come:

* registrazione con verifica tramite OTP
* validazione documentale dei beneficiari
* prenotazione di beni alimentari e medicinali
* validazione manuale delle ricette
* gestione dei punti di distribuzione
* gestione delle donazioni di beni
* gestione delle donazioni monetarie
* invio automatico di notifiche email

Per garantire sicurezza, modularità e manutenibilità, Map4Aid adotta un’architettura **three‑tier**, che separa nettamente interfaccia, logica applicativa e gestione dei dati.

In questo modello, **la view non può accedere direttamente al model**, ma deve sempre passare attraverso il controller e il backend Flask.

# 3.2 Architettura Three‑Tier

L’architettura del sistema è suddivisa in tre livelli indipendenti:

1. **Presentation Tier (Frontend Web)**
2. **Application Tier (Backend Flask)**
3. **Data Tier (Database SQLite + ORM)**

Questa separazione garantisce sicurezza, scalabilità e una chiara divisione delle responsabilità.

## 3.2.1 Presentation Tier – Frontend Web

Il frontend è sviluppato come applicazione web tradizionale, organizzata nelle cartelle:

* **html/** – pagine e template
* **css/** – fogli di stile
* **js/** – script lato client
* **imgs/** – risorse grafiche

### 3.2.1.1 Responsabilità del frontend

* mostrare le informazioni provenienti dal backend
* raccogliere input tramite form (registrazione, login, prenotazioni, upload documenti)
* inviare richieste HTTP alle API REST
* gestire la sessione tramite cookie
* permettere ai donatori di effettuare donazioni monetarie tramite form dedicato

Il frontend **non accede mai direttamente al database**.

Ogni operazione passa attraverso il backend, garantendo sicurezza e controllo degli accessi.

## 3.2.2 Application Tier – Backend Flask

Il backend è il cuore logico del sistema.

È implementato in Python tramite Flask ed è organizzato in moduli coerenti e indipendenti.

La struttura include:

* **controllers/** – gestione delle richieste HTTP e orchestrazione della logica
* **models/** – definizione delle entità e delle relazioni
* **service\_email/** – invio email tramite pattern Bridge
* **app.py** – configurazione dell’applicazione
* **run.py** – avvio del server
* **config.py** – configurazione generale
* **migrations/** – gestione delle migrazioni del database

### 3.2.2.1 Responsabilità del backend

* autenticazione e autorizzazione (OTP, sessioni, ruoli)
* gestione delle prenotazioni di beni e medicinali
* validazione manuale delle ricette
* gestione dei pacchi alimentari
* gestione delle donazioni di beni
* gestione delle donazioni monetarie, con registrazione dell’importo e notifica all’ente
* invio email tramite EmailControlBridge
* geolocalizzazione tramite Geopy/Nominatim
* applicazione della logica di business
* protezione degli endpoint tramite middleware di permessi

Il backend è l’unico livello autorizzato a interagire con il database.

## 3.2.3 Data Tier – Database SQLite + SQLAlchemy ORM

Il livello dati utilizza:

* **SQLite** come database relazionale locale
* **SQLAlchemy** come ORM per mappare le entità Python alle tabelle SQL

### 3.2.3.1 Responsabilità del Data Tier

* memorizzazione persistente dei dati
* integrità referenziale
* gestione delle relazioni tra entità
* transazioni atomiche
* nessun accesso diretto da parte del frontend

Il database contiene:

* utenti e ruoli
* pending accounts
* enti erogatori
* punti di distribuzione
* beni e sottocategorie
* pacchi alimentari
* prenotazioni
* donazioni di beni
* donazioni monetarie (importo, donatore, ente destinatario, data)

# 3.4 Comunicazione tra i Livelli

La comunicazione segue rigorosamente il modello three‑tier:

Codice

Frontend → Backend → Database

Frontend ← Backend ← Database

Il backend:

1. riceve la richiesta
2. valida i dati
3. applica la logica di business
4. interroga il database
5. restituisce una risposta JSON

Questo garantisce:

* sicurezza
* consistenza dei dati
* controllo degli accessi
* separazione delle responsabilità

# 3.5 Deployment

Il sistema è eseguito in ambiente locale tramite:

Codice

flask run

con:

* backend Flask
* database SQLite
* esecuzione single‑instance
* virtual environment dedicato (venv/)

Questa configurazione è ideale per sviluppo e test, ma l’architettura three‑tier permette una futura migrazione verso:

* server dedicati
* container Docker
* database più performanti (PostgreSQL, MySQL, DJANGO)
* hosting cloud

senza modificare la struttura logica del sistema.

# 3.6 Conclusione

L’architettura three‑tier adottata da Map4Aid garantisce:

* separazione chiara tra interfaccia, logica e dati
* maggiore sicurezza (il frontend non accede ai dati)
* modularità e manutenibilità
* possibilità di scalare in futuro
* integrazione naturale delle donazioni monetarie e materiali

È una scelta robusta e perfettamente adeguata alla natura del sistema, che gestisce dati sensibili, processi critici e attori con ruoli diversi.