



Progetto View4Life

Presentazione del *Proof of Concept*

28/01/2026

L'obiettivo del progetto

Sviluppare un applicativo *web* per permettere la gestione delle residenze protette da parte del personale sanitario e dell'amministratore, tramite l'uso dei dispositivi *IoT*.

In particolare, l'applicativo deve:

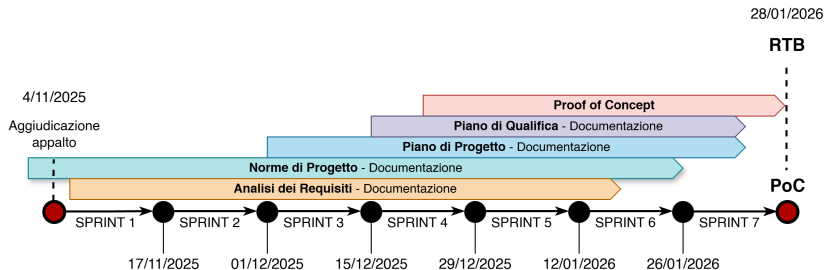
- ▶ Visualizzare le informazioni dei vari dispositivi *IoT*;
- ▶ gestire gli **allarmi**, che verranno presi in carico dal personale sanitario;
- ▶ visualizzare le **analytics** relative alla piattaforma e agli impianti, con relativi suggerimenti per il risparmio energetico;
- ▶ visualizzare una **dashboard** contenente le informazioni principali (stato dispositivi, allarmi attivi, ...).

Metodo di lavoro

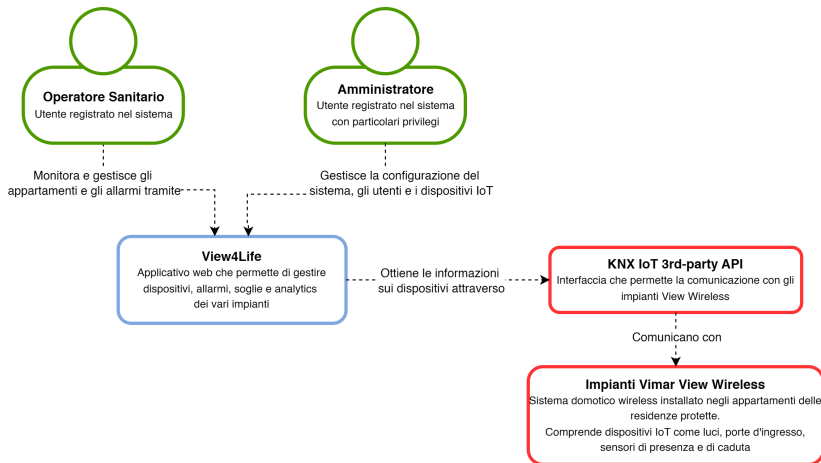
- ▶ Abbiamo scelto una metodologia **agile** con framework **scrum**, composta da **sprint** di 2 settimane ciascuno;
- ▶ Issue Tracking System: **GitHub Projects**.
Le issue non completate in uno sprint vengono spostate e gestite nello sprint successivo;
- ▶ Ad ogni sprint viene effettuata una rotazione dei seguenti ruoli tra i membri del gruppo:

Ruoli
Responsabile
Amministratore
Analista
Progettista
Programmatore
Verificatore

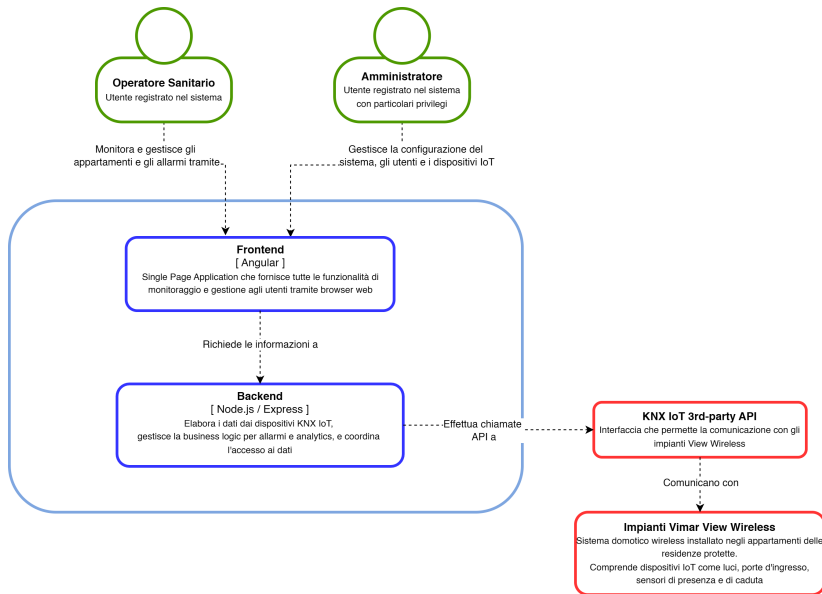
Avanzamento fino ad oggi



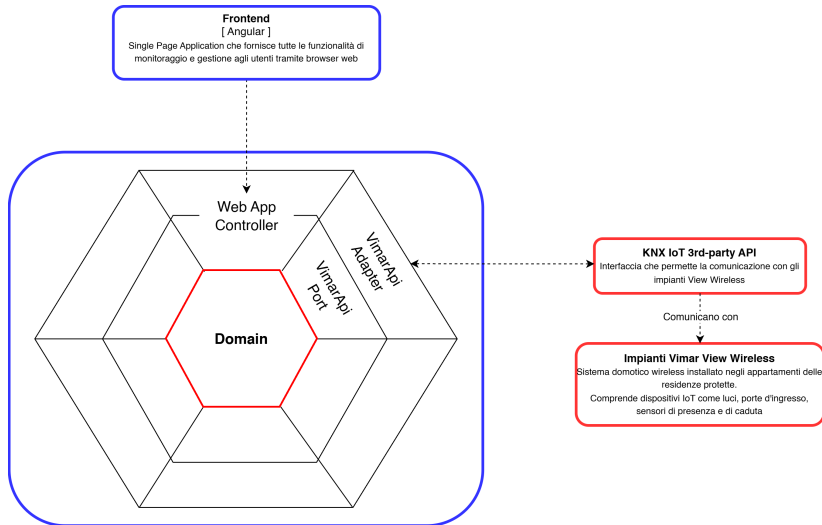
Disegno architeturale - Sistema



Disegno architetturale - Container



Disegno architettuale - Components



Tecnologie frontend

Categoria	Angular	React	Flask
Tipologia	Framework frontend completo	Libreria frontend	Micro-framework backend
Struttura	Fortemente strutturato	Flessibile, non prescrittivo	Minima
Tipizzazione	TypeScript obbligatorio	TypeScript opzionale	Dinamica
Scalabilità	Elevata (enterprise)	Media-Alta	Limitata al backend
Velocità di apprendimento	Bassa (curva ripida)	Alta	Alta
Tooling	Completo e integrato	Ecosistema frammentato	Essenziale
Svantaggi	Complessità iniziale, verbosità	Mancanza di standard nativi	Non adatto al frontend

Tecnologie backend

Categoria	Node.js + Express	Java + Spring	Python + Flask / FastAPI
Tipologia	Runtime JS con framework leggero	Framework backend completo	Framework leggero
Struttura	Poco prescrittiva	Fortemente strutturata	Minima e flessibile
Tipizzazione	Dinamica (TypeScript opzionale)	Statica (Java)	Dinamica (typing opzionale)
Prestazioni	Ottime per API e I/O	Elevate e stabili sotto carico	Buone, FastAPI molto performante
Tooling	Modulare (npm, tool esterni)	Completo e integrato	Essenziale, estendibile
Sicurezza	Gestita tramite middleware	Forte supporto nativo	Supporto base, estendibile
Svantaggi	Poca struttura nativa	Complessità e verbosità	Meno standardizzazione
Ambito ideale	API REST, microservizi	Applicazioni enterprise	API rapide, prototipi

Le opzioni sono:

1. Database relazionale (per l'applicativo web):

- ▶ MySQL
- ▶ PostgreSQL

Possibile implementazione di un *Object Relational Mapper* come **Prisma**

2. Database non relazionale (per analytics):

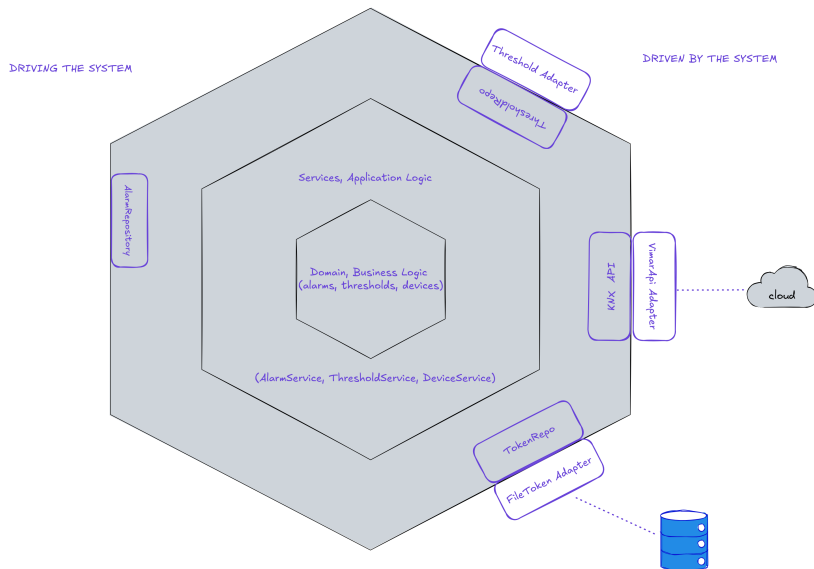
- ▶ TimescaleDB
- ▶ InfluxDB
- ▶ MongoDB

In ottica delle successive fasi di progettazione e design.

Alcuni punti di forza e ragioni della scelta:

- ▶ Il più importante, da cui gli altri seguono, **inversione delle dipendenze**;
- ▶ isolamento del dominio;
- ▶ ampia possibilità di compatibilità ed estensibilità con sistemi esterni (*adapters*);
- ▶ focus sull'implementazione di interfacce stabili (*ports*).

Architettura esagonale



analisi consumo en, anomalie impianto, rilevamento presenza,
presenza prolungata, variazione temp, allarmi inviati, allarmi risolti,
frequenza allarmi
chart.js - open source

soglie prefissate e sulla base dei dati delle analytics, vengono mostrati o meno i suggerimenti

Gestione degli allarmi

screen

Durante l'avanzamento del progetto i principali limiti incontrati sono stati:

- ▶ Pianificazione iniziale dei ruoli e delle ore per la definizione del costo complessivo;
- ▶ studio individuale delle tecnologie riguardanti il PoC;
- ▶ allineamento tra programmatori dopo lo scambio dei ruoli;
- ▶ tempo limitato a causa di esami o altri progetti universitari;