



# **Progetto View4Life del corso di Ingegneria del Software**

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* della gamma *View Wireless*.

In particolare, l'applicativo deve:

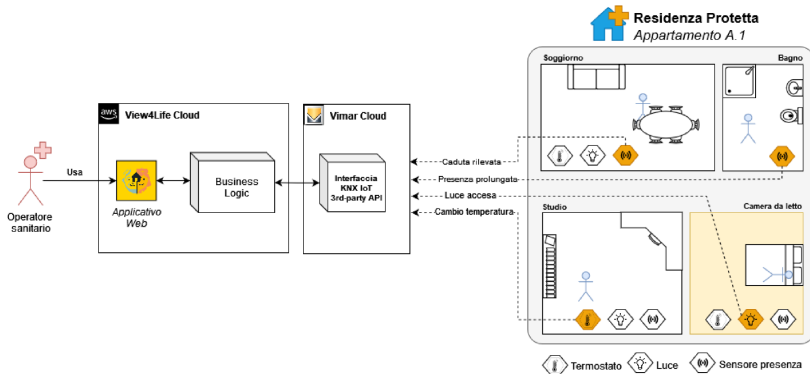
- ▶ Visualizzare le informazioni dei vari **dispositivi**;
- ▶ gestire gli stati degli **allarmi**, che verranno presi in carico dal personale sanitario;
- ▶ visualizzare le **analytics** relative alla piattaforma e agli impianti, con relativi suggerimenti per il **risparmio energetico**;
- ▶ visualizzare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

# Metodo di lavoro

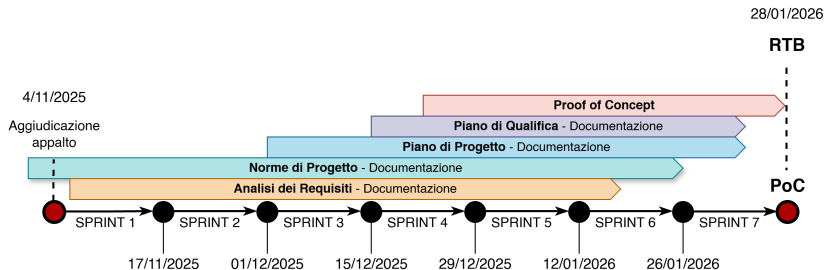
- ▶ Abbiamo scelto una metodologia **agile** con framework **scrum**, composta da **sprint** di 2 settimane ciascuno;
- ▶ Issue Tracking System: **GitHub Projects**.  
Le issue non completate in uno sprint vengono spostate e gestite nello sprint successivo;
- ▶ Ad ogni sprint viene effettuata una rotazione dei seguenti ruoli tra i 7 membri del gruppo:

Ruoli
Responsabile
Amministratore
Analista
Progettista
Programmatore
Verificatore

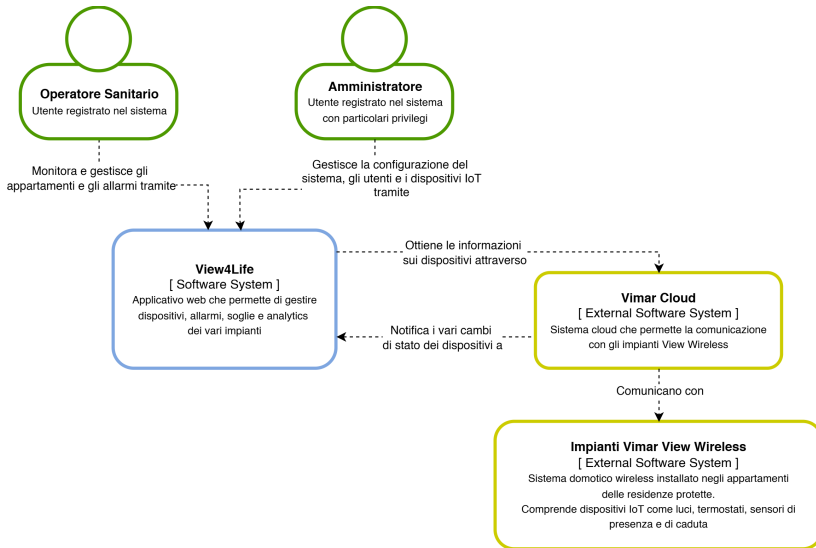
# Schema riassuntivo dei componenti



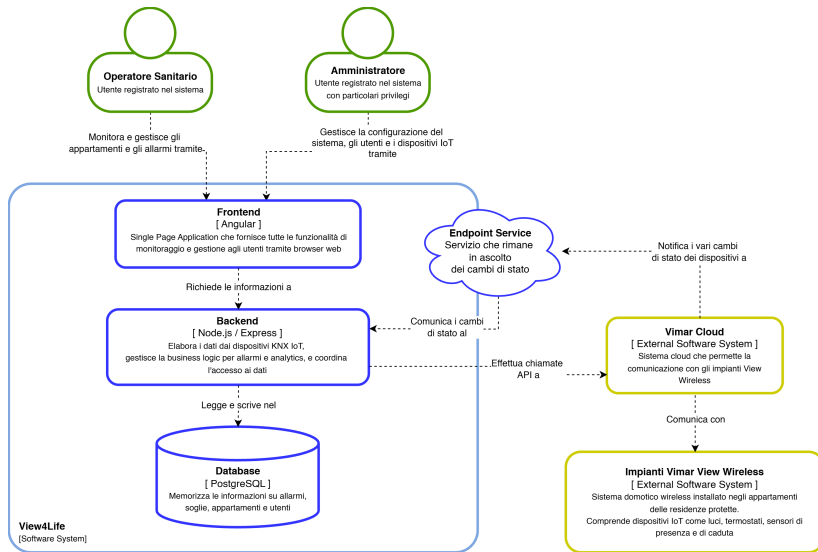
# Avanzamento fino ad oggi



# Disegno architeturale - Sistema



# Disegno architetturale - Container



- ▶ **Angular** è stato scelto per la sua struttura solida e scalabile, adatta ad applicazioni complesse e manutenibili nel tempo in contesti enterprise.
- ▶ Offre un tooling integrato (Angular CLI, build, testing) che garantisce uno sviluppo coerente, riducendo la necessità di configurazioni aggiuntive rispetto a soluzioni più flessibili come React.
- ▶ L'uso obbligatorio di **TypeScript** migliora l'affidabilità del codice grazie alla tipizzazione statica.
- ▶ Presenta una curva di apprendimento più ripida dovuta alla complessità iniziale e ai concetti architetturali da comprendere. (Angular Docs)
- ▶ La scelta privilegia solidità e standardizzazione dell'architettura rispetto alla rapidità di sviluppo iniziale.



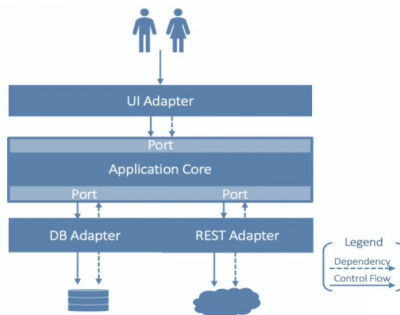
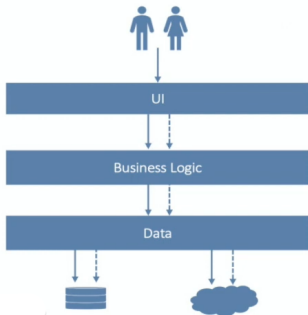
- ▶ Il modello event-driven e non bloccante di Node.js garantisce ottime prestazioni nelle operazioni di I/O e nella gestione di richieste concorrenti, rendendolo particolarmente adatto per mantenere attive le subscription richieste dallo standard KNX IoT senza ricorrere a polling.
- ▶ Express offre una struttura minima e poco prescrittiva, permettendo di modellare l'architettura secondo le esigenze del progetto senza la complessità di framework più pesanti.
- ▶ Il tooling modulare basato su npm consente di integrare facilmente librerie per OAuth2, sicurezza validazione e logging
- ▶ **Node.js + Express** è stato scelto per la leggerezza e la flessibilità che offre, oltre a essere molto in linea per ciò che il progetto richiede, in particolare per la gestione delle API REST.

# Scelta del Database

- ▶ I dati dell'applicativo web, come utenti, credenziali e configurazioni, saranno gestiti con un database relazionale (PostgreSQL) per garantire integrità e coerenza.
- ▶ Le metriche per le analytics saranno memorizzate in un database come TimescaleDB, InfluxDB o MongoDB, ottimizzato per serie temporali e aggiornamenti frequenti.
- ▶ La scelta definitiva del database non relazionale è ancora in fase di valutazione.

- ▶ **Docker** sarà utilizzato per rendere l'ambiente completamente replicabile e portabile tra diversi cloud provider, come richiesto dal progetto.
- ▶ Ogni servizio backend, database e componente dell'applicativo sarà containerizzato, consentendo di avviare l'intero stack con un solo comando.
- ▶ Questo approccio rispetta il principio di Infrastructure as Code e facilita la gestione di dipendenze, aggiornamenti e scalabilità dei vari componenti.

# Backend: architettura layered VS esagonale



## Inversione delle Dipendenze

- ▶ *Layered*: il Dominio dipende dalla Logica di Persistenza. E' necessario implementare fin dall'inizio la Logica di Persistenza.
- ▶ *Esagonale*: il Dominio è isolato al centro dell'applicazione, la Logica di Business è completamente indipendente. Le dipendenze esterne si adattano al Dominio tramite porte.
- ▶ **Conclusione**: l'isolamento del Dominio offerto dal modello esagonale ci ha permesso di concentrarci fin dall'inizio sullo sviluppo delle sue principali entità, rendendo possibile effettuare la scelta del Database in un secondo momento. Questa separazione ha eliminato i vincoli infrastrutturali iniziali, facilitando l'inizio delle attività di sviluppo del *PoC* (per il quale non è prevista una fase di Design).

## Ruolo del Database

- ▶ *Layered*: fondamento del sistema, il quale tende a diventare rigido e fragile rispetto a modifiche infrastrutturali.
- ▶ *Esagonale*: trattato come dettaglio implementativo, il che garantisce ampia flessibilità e possibilità di cambiamento.
- ▶ **Conclusione**: il disaccoppiamento dal Database ci permette di evolvere il modello o cambiare tecnologia di persistenza senza rifattorizzazioni del Dominio, aspetto centrale del PoC. Ci permette inoltre di valutare a posteriori, con possibilità di cambiamento, l'utilizzo di DB relazionali e/o non relazionali.

## Testabilità

- ▶ *Layered*: complesso testare esclusivamente il Dominio, test legati a mocking complessi o Database attivi (non ancora presenti).
- ▶ *Esagonale*: consente **Unit Test** rapidi e isolati sul Dominio.
- ▶ **Conclusione**: l'architettura esagonale ci garantisce la possibilità di testare immediatamente la correttezza della Logica di Business.  
Ciò corrisponde a coprire la maggior parte del codice del *PoC*.

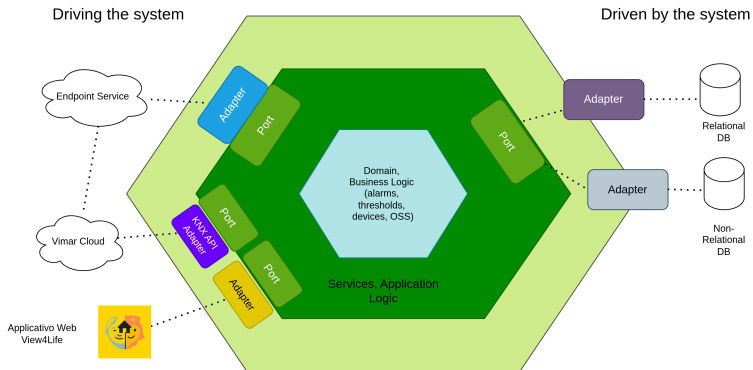
# Backend: confronto architettura layered VS esagonale (continua)

## Struttura e Manutenibilità

- ▶ *Layered*: i *layer* tendono a mascherare la complessità, rendendo difficile l'orientamento con l'avanzare del progetto.
- ▶ *Esagonale*: indirizza maggiormente lo sviluppo. Offre chiarezza e trasparenza nelle integrazioni con sistemi esterni (*ports-adapters*).
- ▶ **Conclusione**: l'architettura esagonale ci impronta allo sviluppo di una *codebase* più ordinata e manutenibile, in particolare facilitando l'integrazione di servizi esterni. Ci permette inoltre di approcciarci al mondo dello sviluppo software con attenzione fin da subito alle *best-practices* riguardo alla separazione delle responsabilità.



# Backend: Architettura esagonale



Nella sezione Analytics saranno visualizzate, attraverso dei **grafici**, le statistiche specificate nel capitolato, tra cui:

- ▶ l'energia consumata dall'illuminazione;
- ▶ anomalie d'impianto (e.g. valori statistici sospetti come consumi fuori media o variazioni di temperatura irrealistiche);
- ▶ rilevamento presenza/assenza/caduta;
- ▶ rilevamento presenza prolungata;
- ▶ variazione e cambio di temperatura;
- ▶ allarmi inviati e risolti giornalmente;
- ▶ frequenza allarmi e cadute di un periodo a scelta.

Le statistiche risultano utili per:

- ▶ fornire una visione d'insieme sugli impianti;
- ▶ ottenere informazioni sull'efficienza degli impianti;
- ▶ sviluppare strategie per ottimizzare i consumi;
- ▶ valutare l'efficacia della gestione degli allarmi.

Per la visualizzazione dei grafici è stato previsto l'utilizzo di una tra le seguenti alternative:

- ▶ **ng2-charts**, wrapper della libreria *Chart.js*
- ▶ **ngx-charts**, libreria grafica per Angular

La funzionalità dei suggerimenti per limitare il consumo energetico è basata su:

- ▶ una raccolta consigli con struttura fissa;
- ▶ visualizzazione del suggerimento qualora i dati statistici violino delle soglie (eventualmente parametrizzabili).

Alcune proposte esempi di suggerimenti sono:

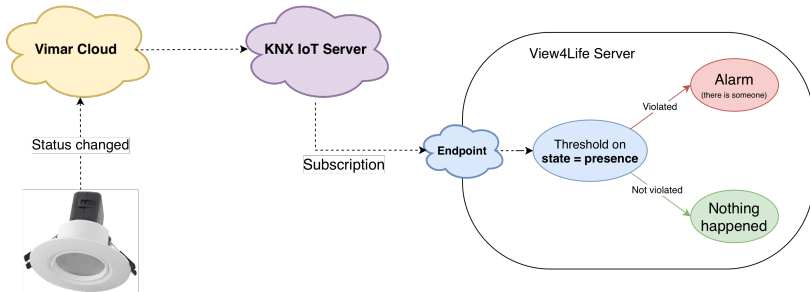
- ▶ Per risparmiare energia, spegni le luci negli impianti esposti a sud dalle 12:00 alle 15:00;
- ▶ Mantieni una temperatura inferiore 19° dalle 22:00 alle 6:30, risparmi energia e migliora il sonno;
- ▶ La luce del soggiorno dell'impianto 2 rimane accesa anche senza nessuna presenza rilevata, spegnila per evitare consumi ulteriori.

# Interfaccia Analytics (bozzetto grafico)



# Gestione degli allarmi

- ▶ Threshold, unità di controllo dei valori rilevati da un *datapoint* (valore di scatto e operatore di confronto).
- ▶ Alarm è l'evento generato alla violazione della soglia;



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;

Le attività future pianificate per i prossimi sprint sono:

- ▶ Approvazione della Requirements Baseline e Technology Baseline da parte dei professori universitari;
- ▶ Analisi del PoC e individuazione componenti rimodellabili;
- ▶ Progettazione architeturale e di dettaglio dei componenti;
- ▶ Implementazione dei componenti e delle funzionalità.