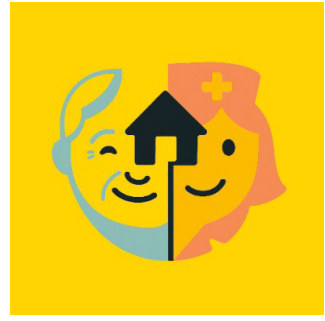


Progetto View4Life del corso di Ingegneria del Software

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:

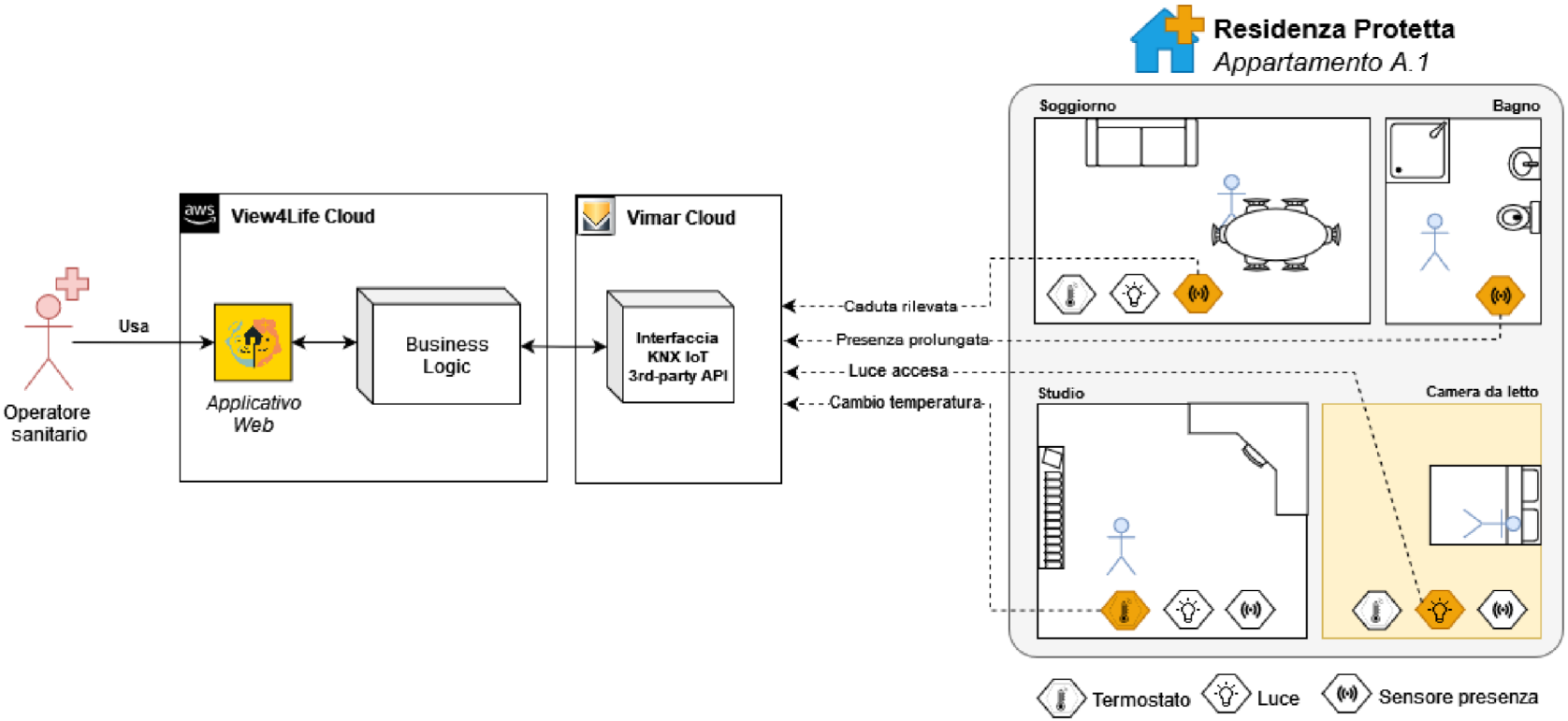
- Visualizzare le informazioni dei vari **dispositivi**;
- gestire gli stati degli **allarmi**, che verranno presi in carico dal personale sanitario;
- visualizzare le **analytics** relative alla piattaforma e agli impianti, con relativi suggerimenti per il **risparmio energetico**;
- visualizzare una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

Metodo di lavoro

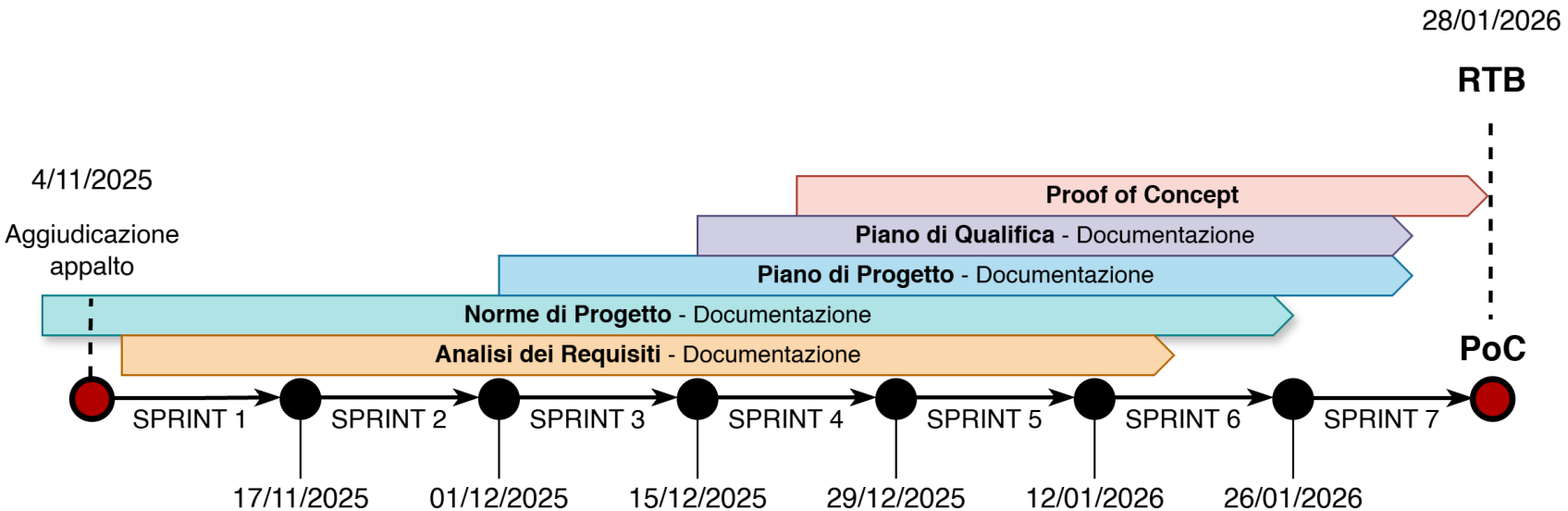
- Abbiamo scelto una metodologia **agile** con framework **scrum**, composta da **sprint** di 2 settimane ciascuno;
- Issue Tracking System: **GitHub Projects**.
  Le issue non completate in uno sprint vengono spostate e gestite nello sprint successivo;
- Ad ogni sprint viene effettuata una rotazione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

Ruoli
Responsabile
Amministratore
Analista
Progettista
Programmatore
Verificatore

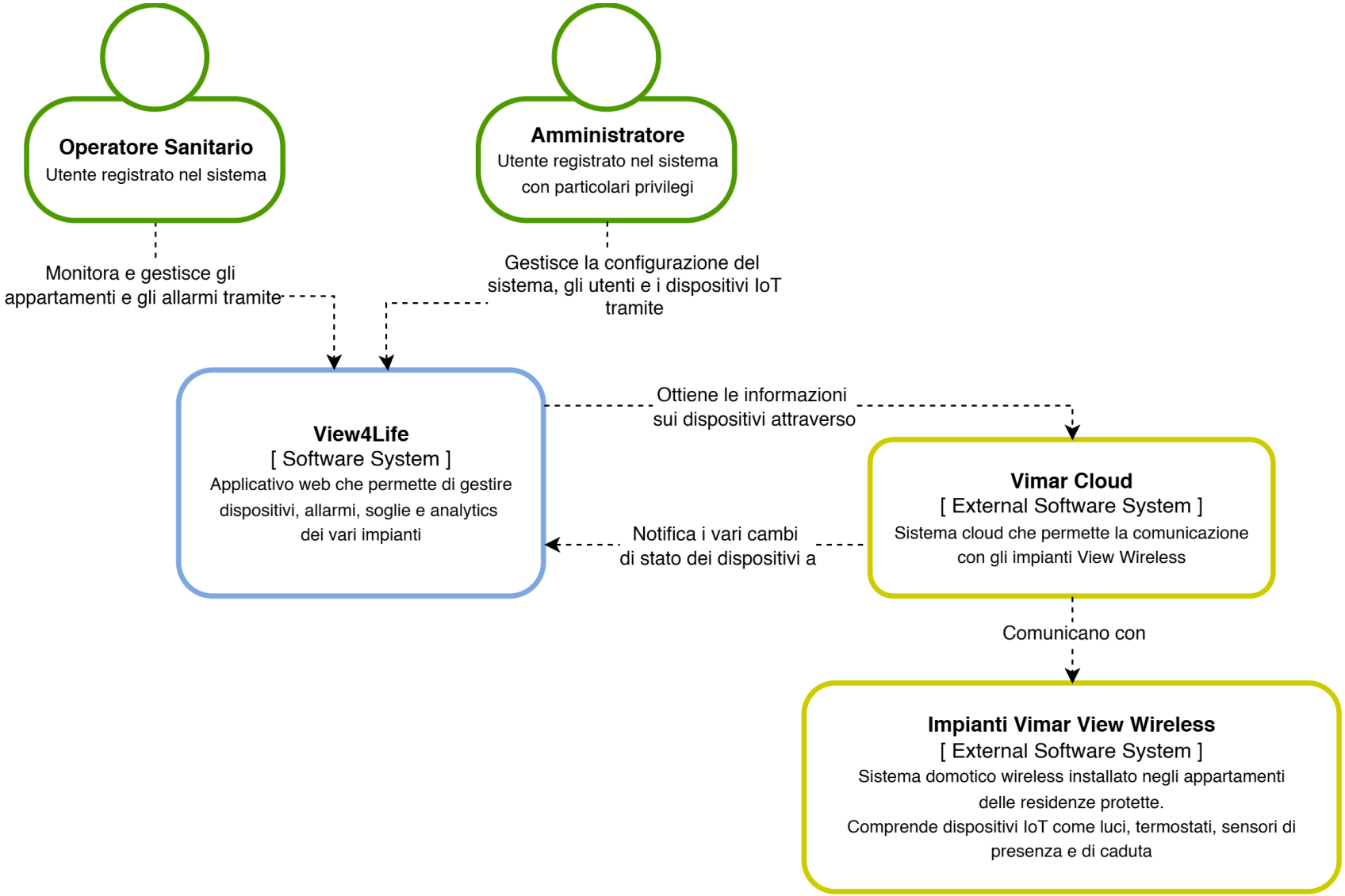
Schema riassuntivo dei componenti



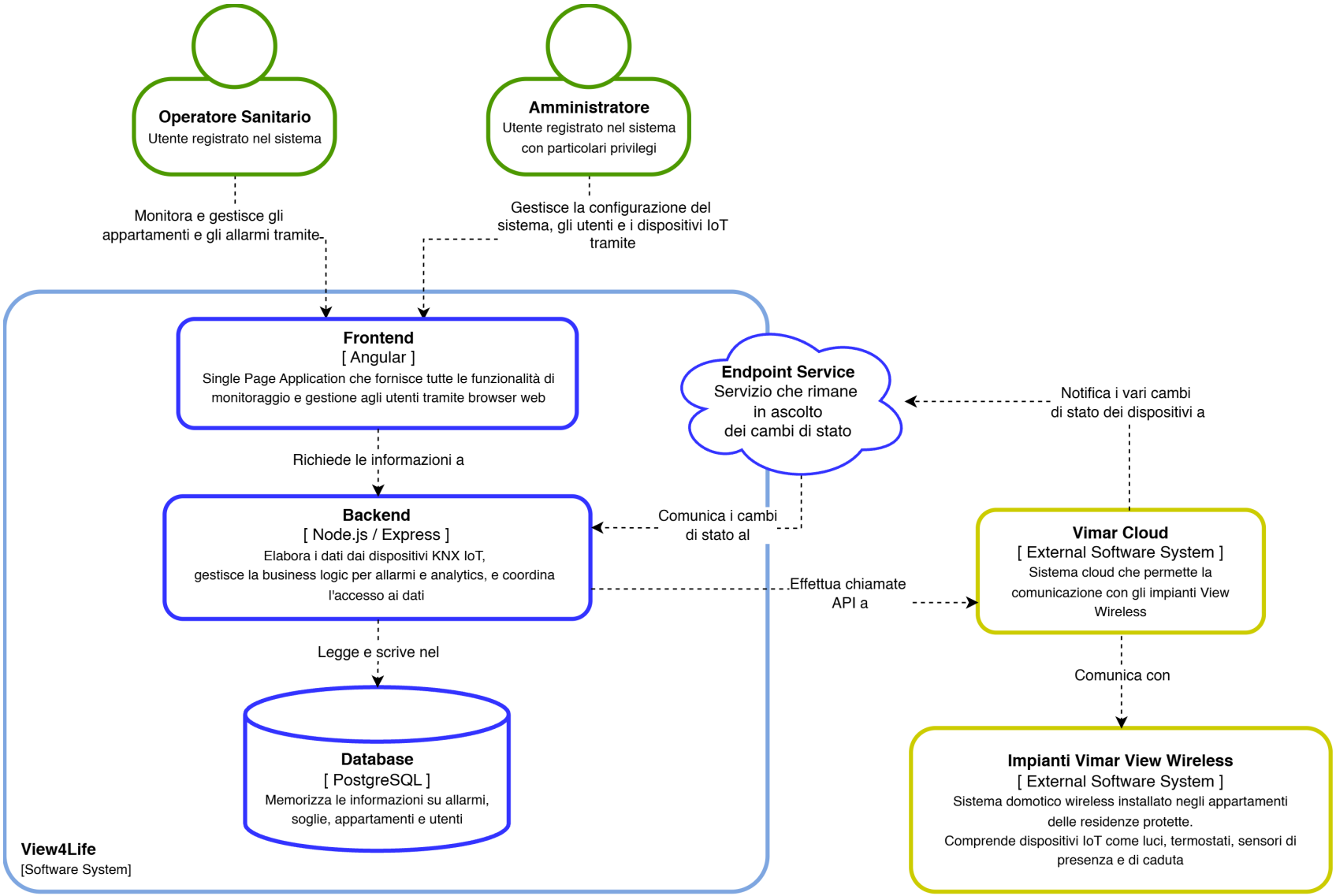
Avanzamento fino ad oggi



Disegno architetturale - Sistema



Disegno architetturale - Container



Tecnologie frontend

- **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.

Tecnologie backend

- 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 o MySQL) per garantire integrità e coerenza.
- Le metriche per le analytics saranno memorizzati in un database come TimescaleDB, InfluxDB o MongoDB, ottimizzato per serie temporali e aggiornamenti frequenti.

Docker

- **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 esagonale

	Layered	**Esagonale**
Dominio	Dipende da Persistent Logic (mentre per PoC trascuriamo DB)	Isolato, guida lo sviluppo. Ci permette di concentriarci sulla modellazione del dominio
Dipendenze	App→Bus→Pers. Dati guidano sviluppo (non sappiamo ancora come rappresentarli)	App→Bus←Pers. Domain guida sviluppo (conosciamo le principali entità)
Testabilità	Difficile testare senza mocking oneroso o avviare PostGre	Facile e veloce, in particolare per Domain (Unit Test)
Integrazioni	Chiamate API nei Service. Cambio API = cambio Business Logic	API come Secondary Adapters, interscambiabili rispetto al dominio

Backend: Architettura esagonale (continua)

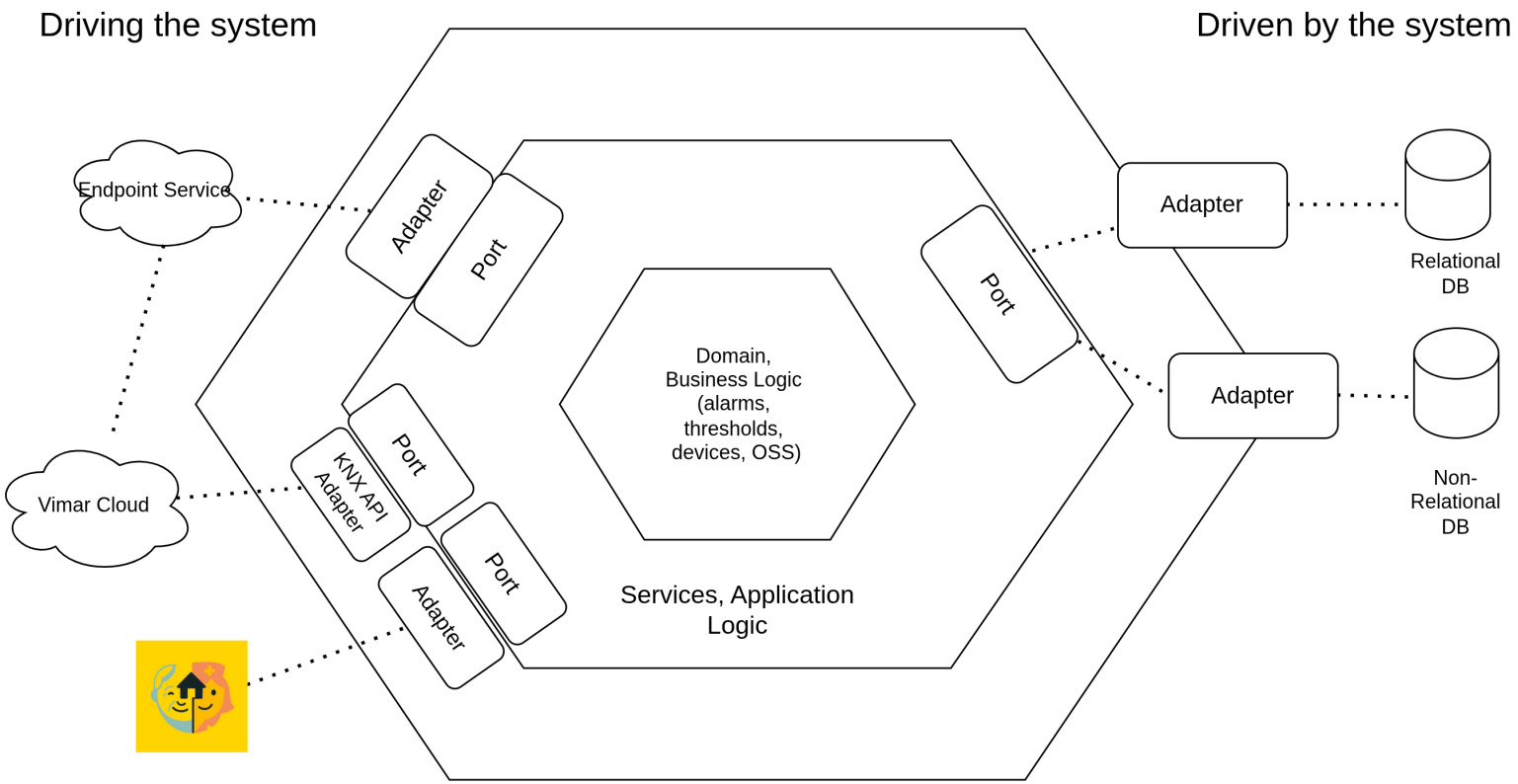
	Layered	**Esagonale**
Database	Ruolo chiave, fondamenta. Sistema rigido e fragile	Dettaglio implementativo, ampia possibilità di cambiamento
Struttura e Manutenibilità	Layer nascondono la complessità. Difficoltà orientarsi con progetto che avanza	Indirizza lo sviluppo, mantiene chiarezza e trasparenza nelle integrazioni esterne

Backend: Architettura esagonale

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*).

Backend: Architettura esagonale



Analytics

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

- l'energia consumata dall'illuminazione;
- anomalie d'impianto;
- 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

Analytics

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**, framework per Angular

Suggerimenti

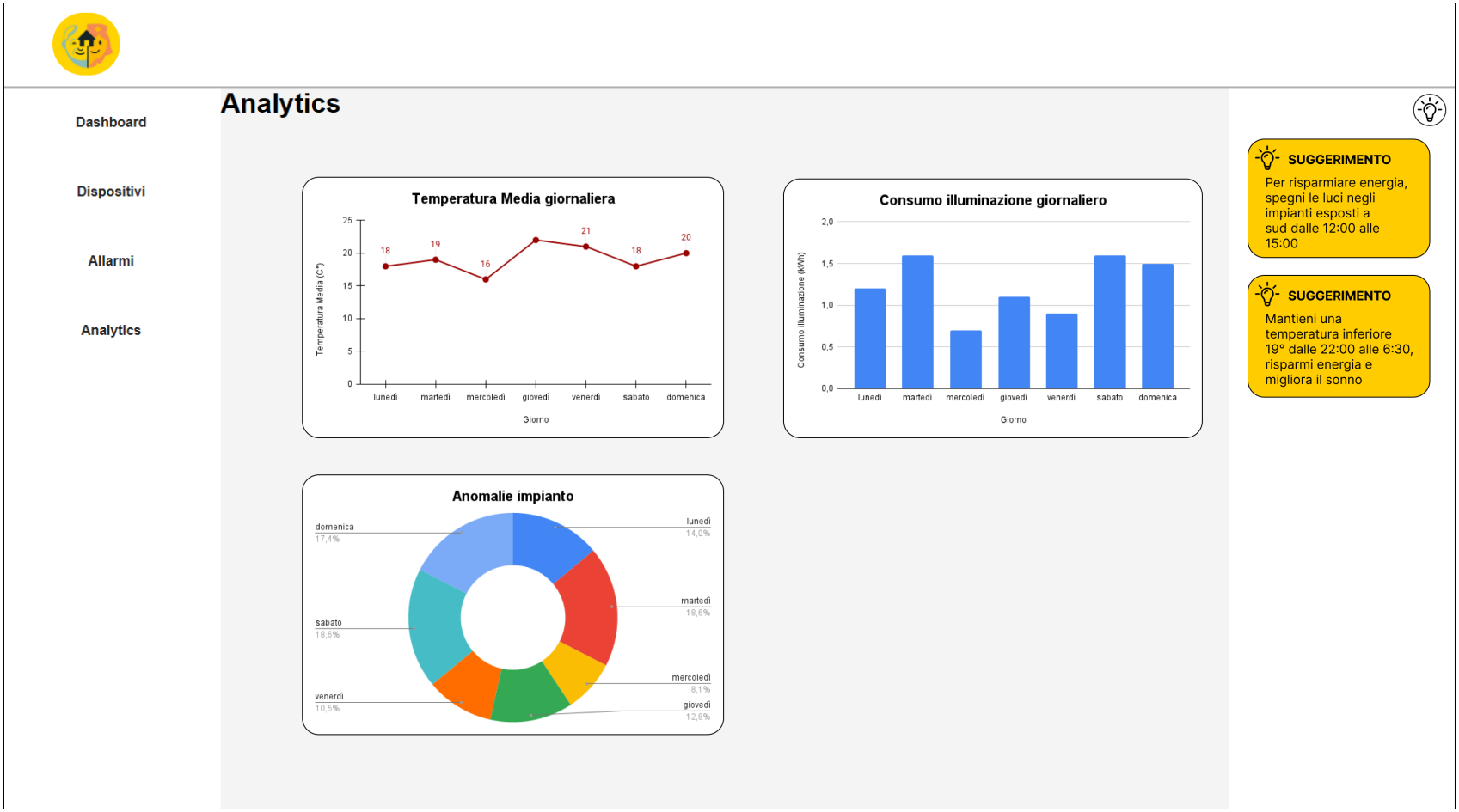
La funzionalità dei suggerimenti per limitare il consumo energetico è basata su:

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

Alcune proposte esempi di suggerimenti sono:

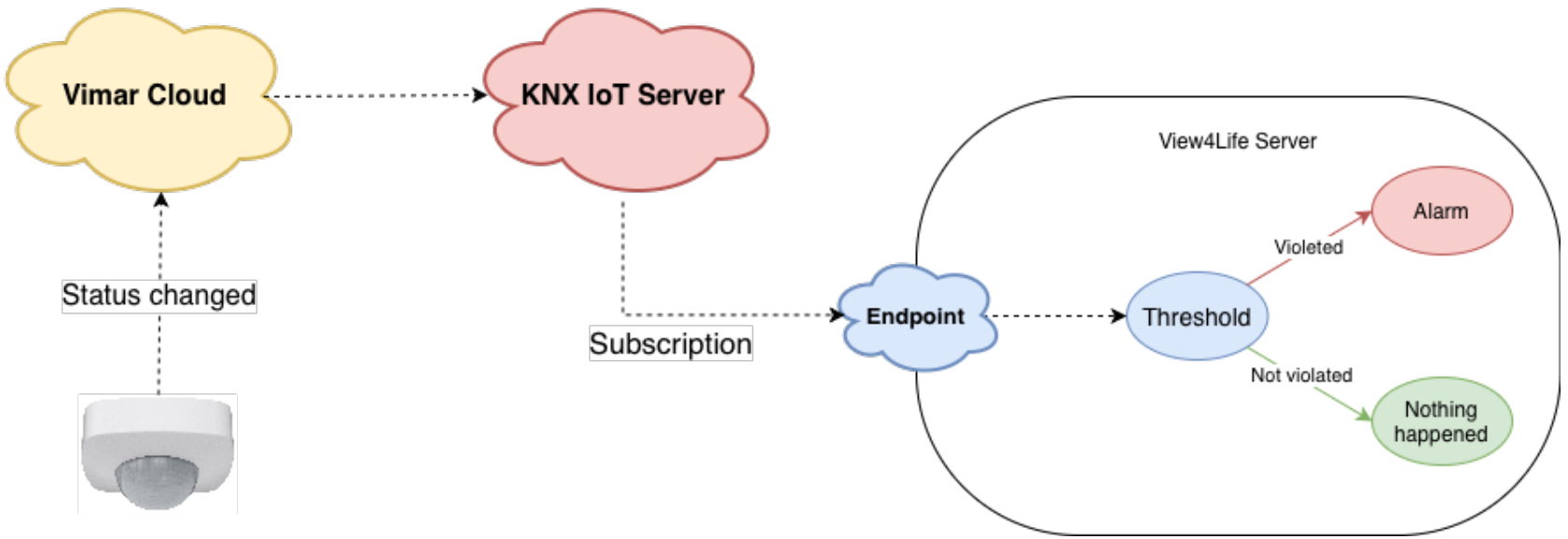
- Per risparmiare energia, spegni le luci negli impianti esposti a sud dalle 12:00 alle 15:00;
- Mantieni una temperatura inferiore 19° dalle 22:00 alle 6:30, risparmi energia e migliora il sonno;
- La luce del soggiorno dell'impianto 2 rimane accesa anche senza nessuna presenza rilevata, spegnila per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



Limiti o blocchi incontrati

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;