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:

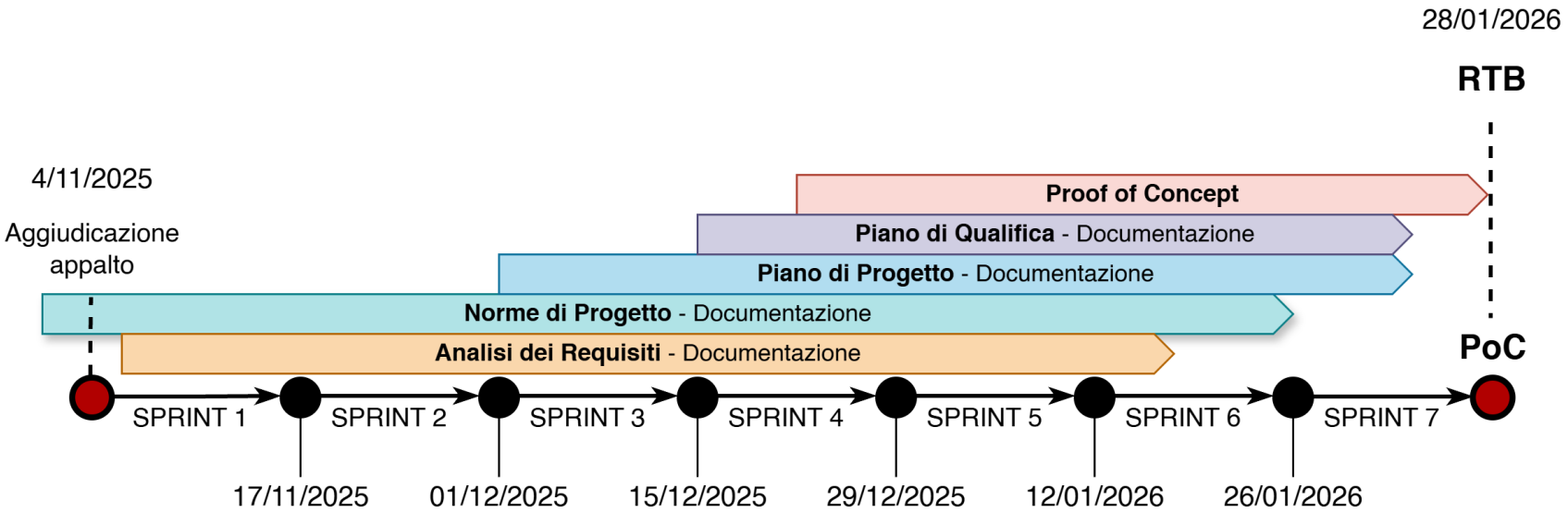
- Visualizzare le informazioni dei vari dispositivi *IoT*;
- gestire gli **allarmi**, che verrano presi in carico dal personale sanitario;
- visualizzare le **analytics** relative alla piattaforma e agli impianti, con relativi suggerimenti per il risparmio energetico;
- visualizzare una **dashboard**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

Metodo di lavoro

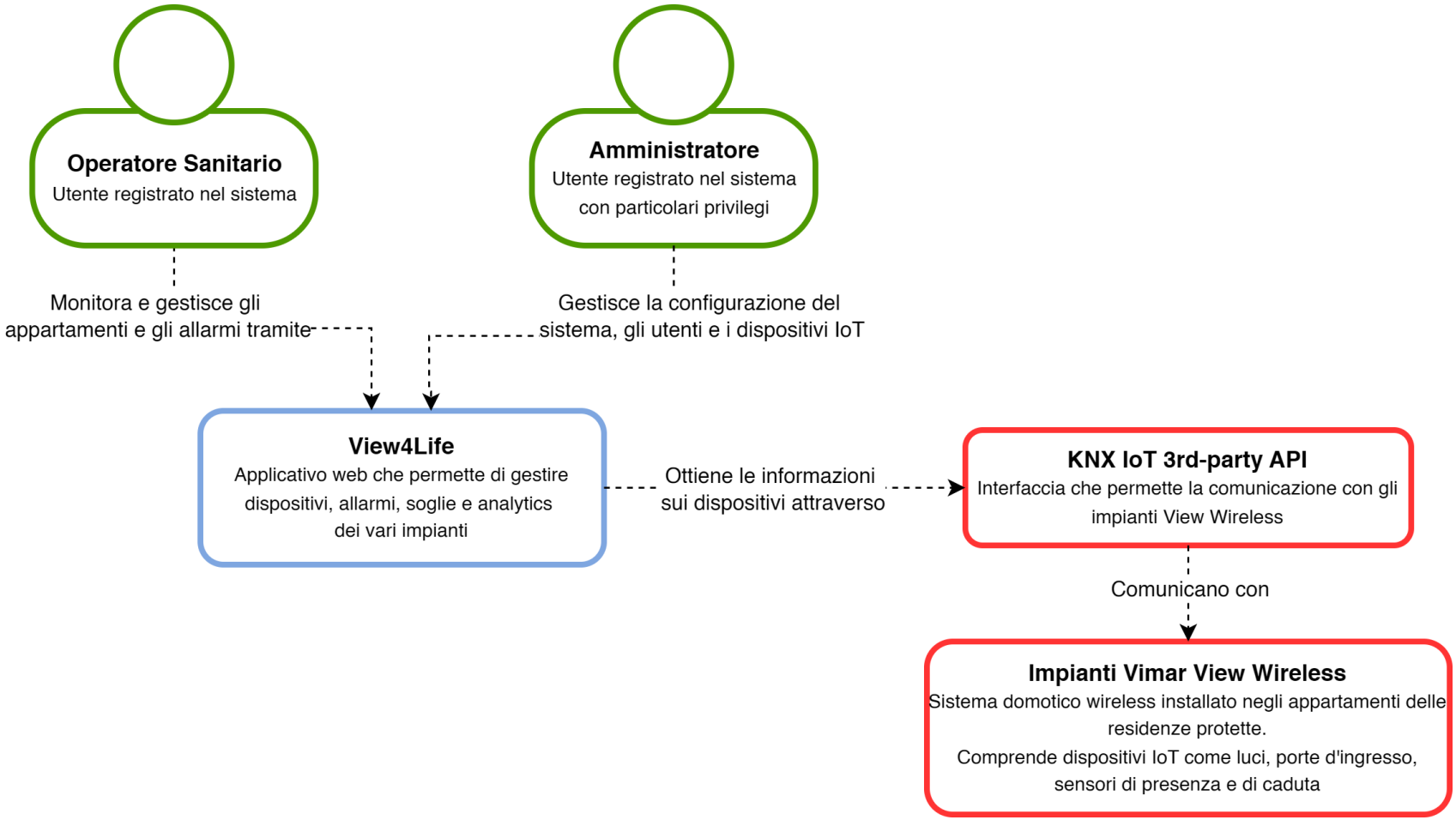
- Abbiamo scelto una metodologia **agile** con framework **scrum**, composta da **sprint** di 2 settimane ciascuno;
- Issue Tracking System: **GitHub Projects**.
  Le issue non completate in uno sprint vengono spostate e gestite nello sprint successivo;
- Ad ogni sprint viene effettuata una rotazione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

Ruoli
Responsabile
Amministratore
Analista
Progettista
Programmatore
Verificatore

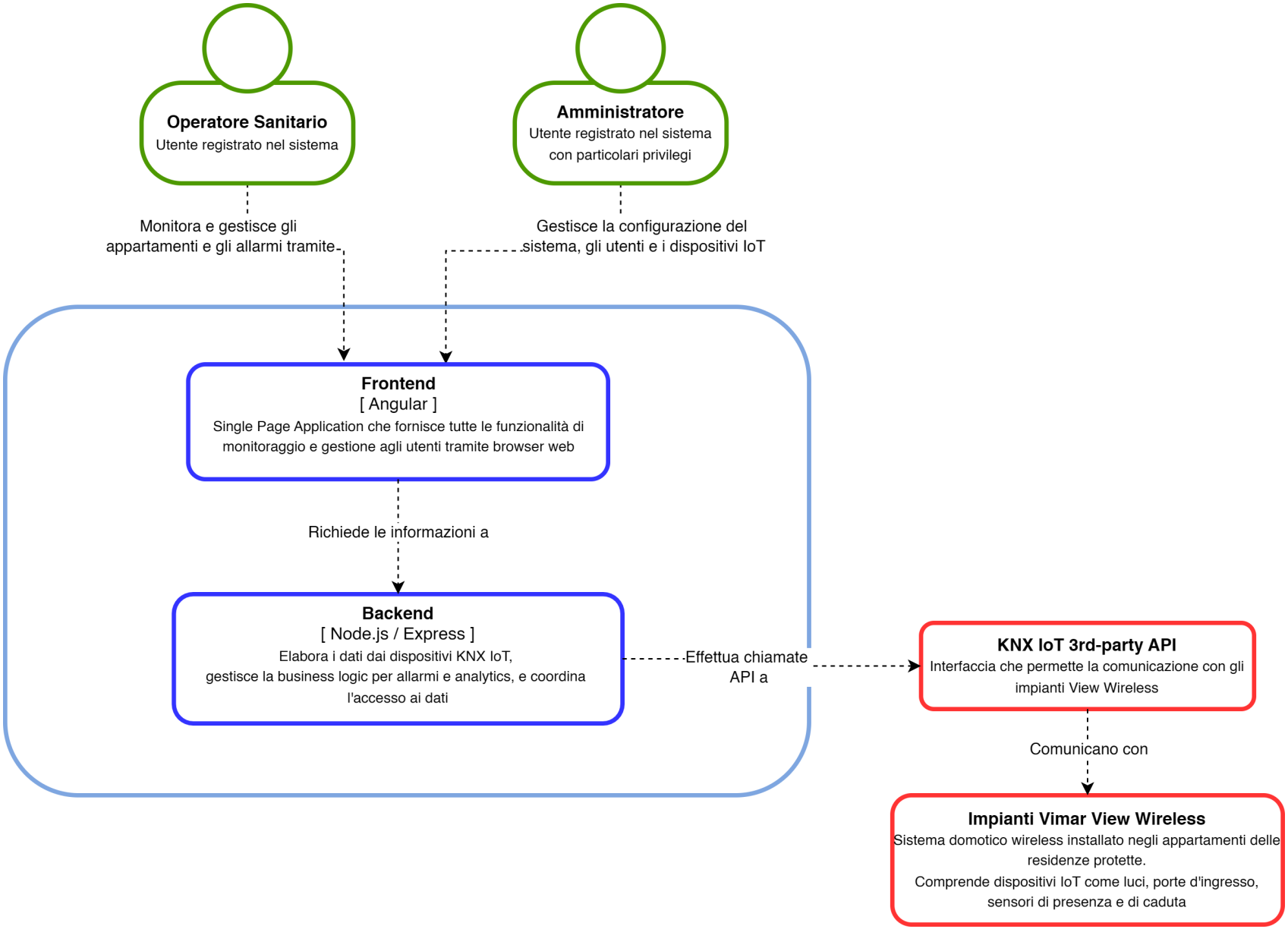
Avanzamento fino ad oggi



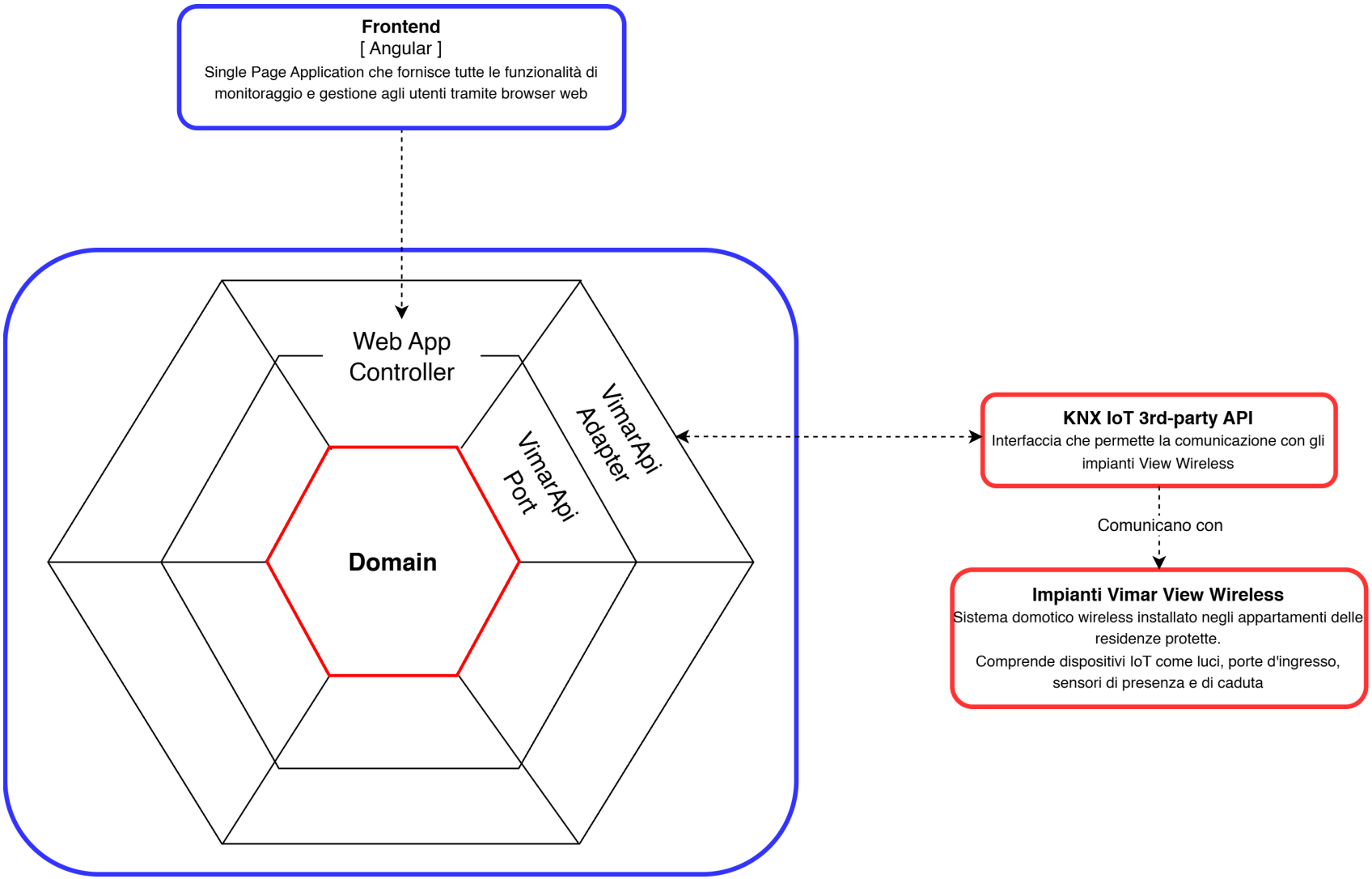
Disegno architetturale - Sistema



Disegno architetturale - Container



Disegno architetturale - 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**

Scelta del Database

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

Backend: Architettura esagonale

	Layered	**Esagonale**
Dominio	Dipende da Persistent Logic	Isolato, guida lo sviluppo
Dipendenze	App→Bus→Pers. Dati guidano sviluppo	App→Bus←Pers. Domain guida sviluppo
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)

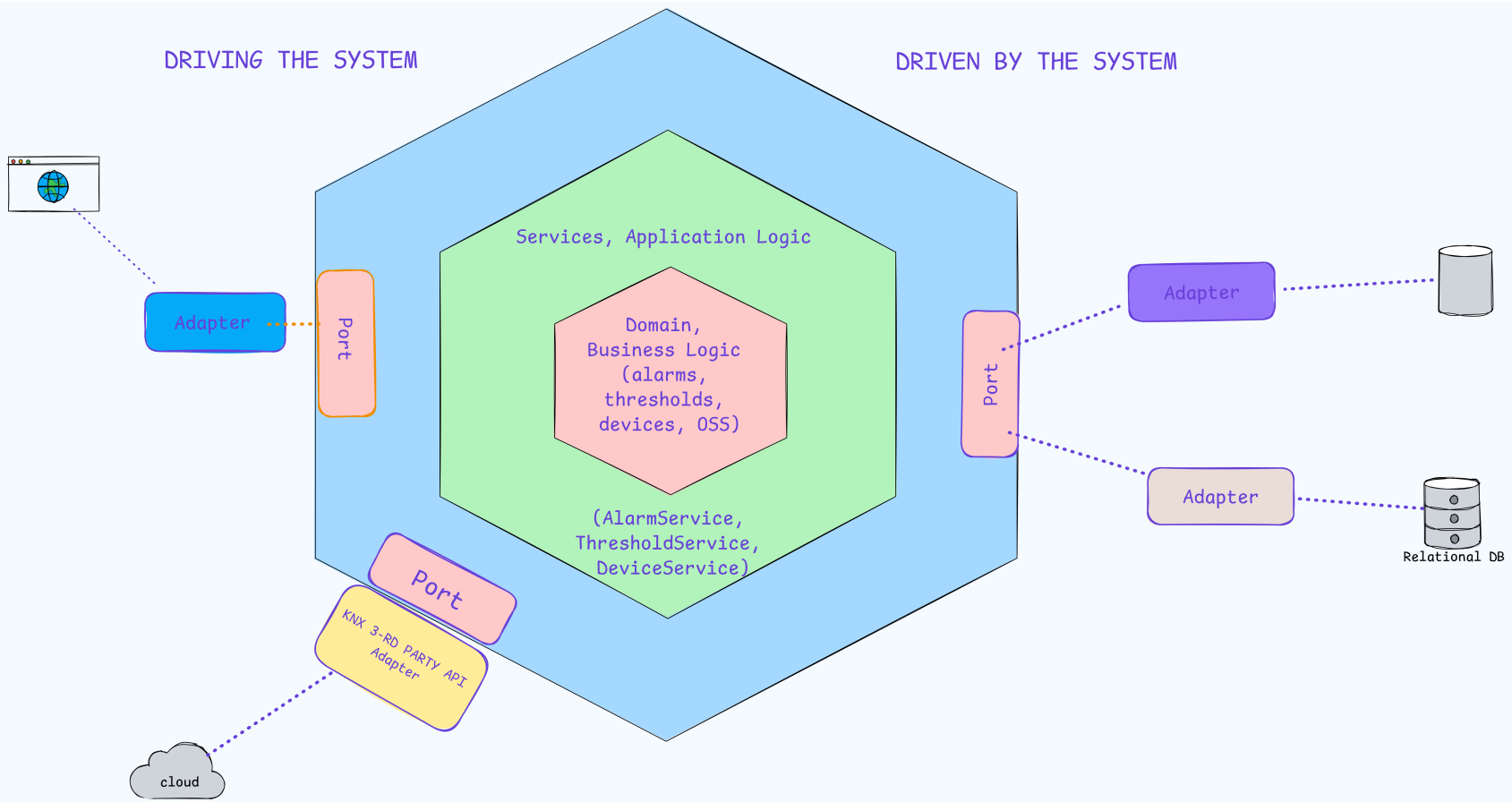
	Layered	**Esagonale**
Database	Ruolo chiave, fondamenta. Sistema rigido e fragile	Dettaglio implementativo, ampia possibilità di cambiamento
Struttura e Manutenibilità	Layer nascondono la complessità. Difficoltà orientarsi con progetto che avanza	Indirizza lo sviluppo, mantiene chiarezza e trasparenza 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 le statistiche riportate nel capitolato sotto forma di grafici.
È stato previsto l'utilizzo di librerie per la visualizzazione dei grafici, le proposte sono:

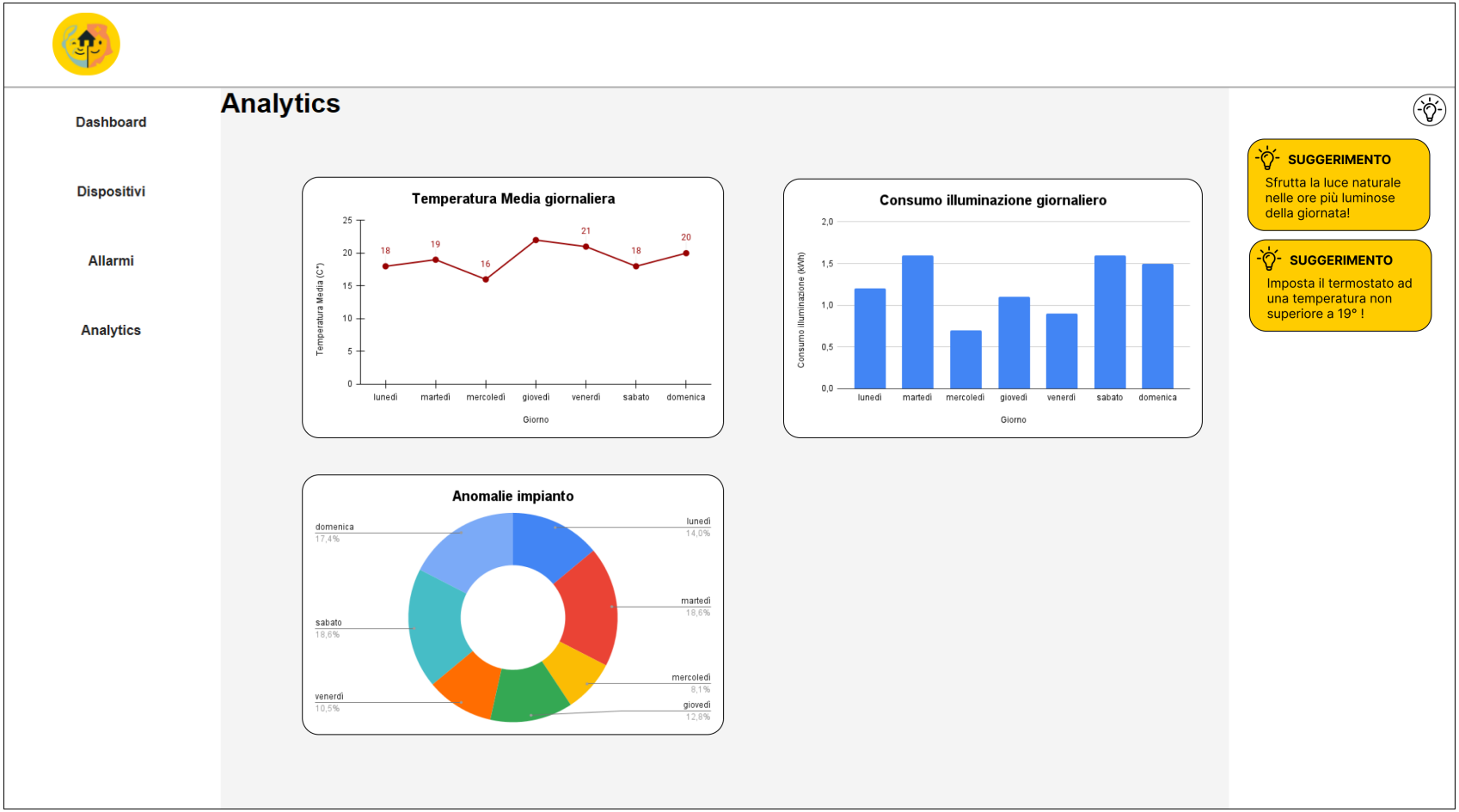
- *ng2-charts*
  - wrapper di *Chart.js* per Angular
  - semplice e leggero
  - licenza MIT
- ngx-charts
  - framework per Angular
  - licenza MIT

Suggerimenti

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

- una raccolta fissa di consigli (definita staticamente);
- visualizzazione del suggerimento qualora dei valori istantanei o statistici violino delle soglie (eventualmente parametrizzabili). Ad esempio:
  - Valore istantaneo: la temperatura supera 19° (simile al meccanismo degli allarmi)
  - Valore statistico: il consumo giornaliero di una luce supera un certo valore

Interfaccia Analytics



Gestione degli allarmi

- E' possibile creare delle `Threshold`, unità di controllo dei valori rilevati da un *datapoint*.
- Vengono definiti valore di scatto e operatore di confronto;
- `Alarm` è l'evento generato alla violazione della soglia;
- I valori che possono far scattare la soglia provengono dalle *subscription*

`Ampliamo a voce`

Interfaccia grafica

screen

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;