PROGETTO AA 2022/23

(variazioni sul tema)

Fase di progettazione

(dopo fase di specifica, precede l’implementazione)

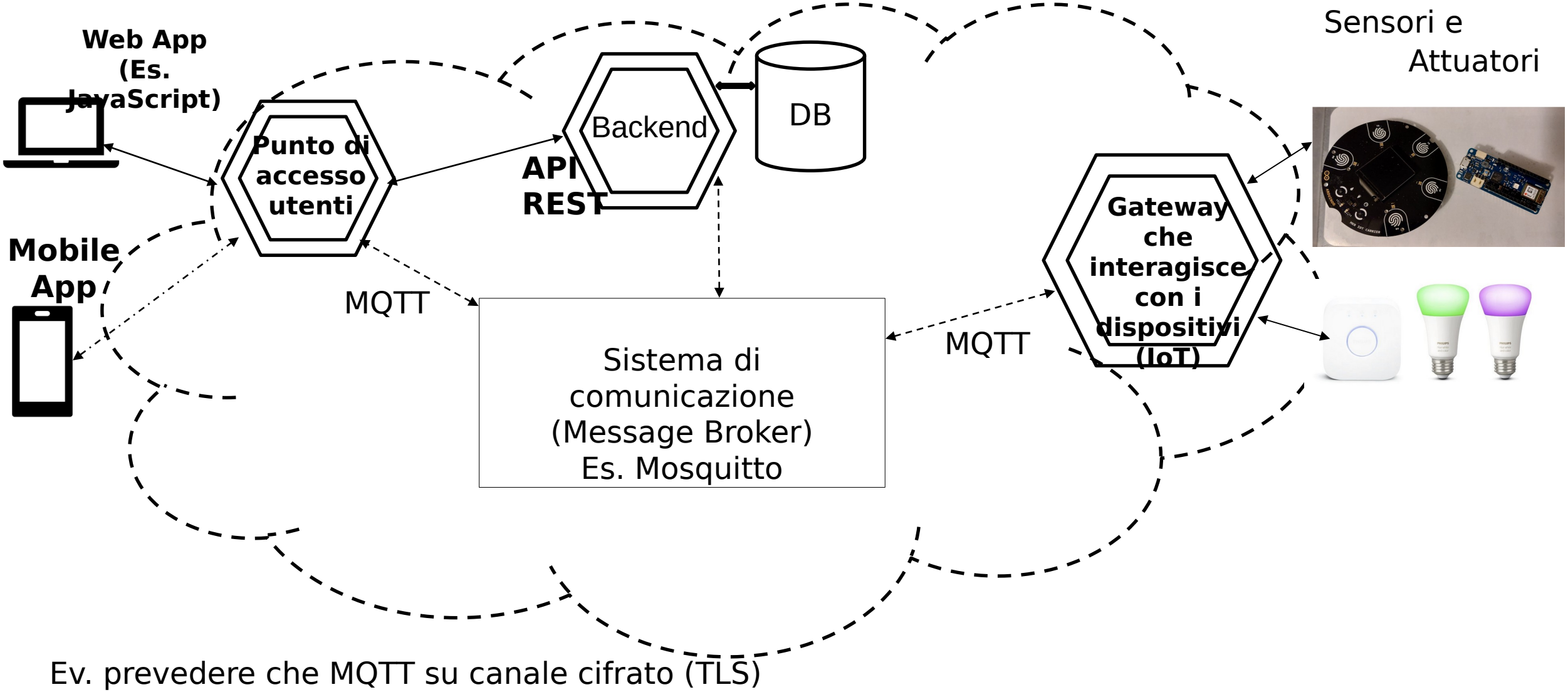
FASE 2: Progettazione

- Nella fase di specifica avete descritto COSA si doveva progettare
- Nella fase di progettazione dovrete definire COME realizzare il sistema

Alcuni suggerimenti su come impostare questa fase:

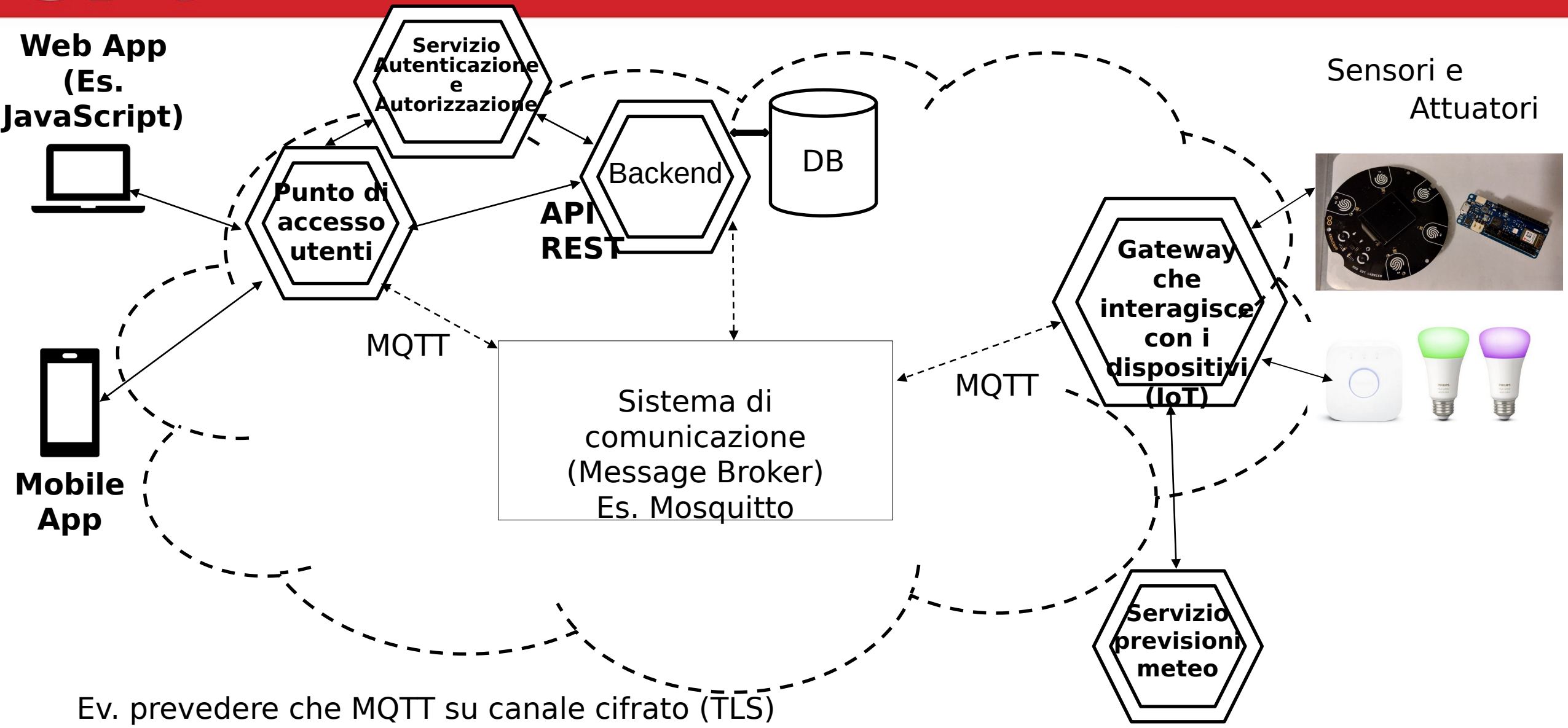
- Architettura basata su servizi il più possibile flessibile rispetto al deployment dei diversi servizi
- Alcuni servizi devono essere disponibili tramite interfaccia REST

Architettura del sistema IoT



Ev. prevedere che MQTT su canale cifrato (TLS)

Architettura del sistema IoT



Componenti principali dell’architettura:

- Client App: può essere molto semplice (linea di comando) oppure eseguito nel browser (es. realizzato con javascript) oppure un’applicazione mobile: interagisce con un microservizio che funge da punto di accesso al sistema e che quale contatta il backend tramite le API REST.
- Backend: permette agli utenti di accedere per consultare le misure presenti nel DB o inserire / modificare i dati relativi alle varie entità gestite. Espone un’interfaccia REST. Comunica con il sottosistema IoT tramite il broker MQTT.
- DB: database contenente tutte le informazioni necessarie al funzionamento del sistema (es. misure rilevate, informazioni di configurazione, …). Accessibile solo tramite il backend
- Sottosistema IoT - si occupa di rilevare misure tramite i sensori che invia tramite broker MQTT e tramite lo stesso canale di comunicazione può accettare comandi per gli attuatori. Può essere quindi sia publisher che subscriber.

Cosa produrre nella fase di progettazione

- Per ciascun servizio: diagramma delle classi con i relativi attributi e metodi, diagrammi di sequenza per mostrare le interazioni tra diversi oggetti ed eventualmente diagrammi di attività per mostrare la suddivisione dei compiti tra diversi thread di un certo servizio.
- Può essere utile strutturare in package le diverse classi per evidenziarne i diversi componenti.

Cosa produrre nella fase di progettazione

Progettazione API REST (e documentazione …)

- Definire Risorse
- Definire la Rappresentazione delle risorse (che verrà scambiato tra client e server; noi useremo un formato JSON)
- Definire gli Endpoint
- Definire le possibili Azioni
- Definire i possibili Errori

Definizione delle Risorse (dal diagramma classi di dominio)

Siamo interessati ad accedere a singole risorse ma anche a collezioni di risorse (per queste ultime usiamo nomi plurali):

- Coltivazioni o Campi
- Sensori/Attuatori (o IoTdevs per accorparli; per distinguerli si può associare a ciascun device un attributo "tipo")
- Utenti (fornitore idrico / agricoltore)

Rappresentazione delle risorse

Coltivazione:

{ "id": 14,
"id_azienda_agri": 3,
"descrizione": "Serra fiori",
"fabbisogno": "30",
….,
"IoTdevs": [{"id": 10, "tipo": "sensore_temp"},{"id": 24, "tipo": "sensore_umid"}, {"id": 28, "tipo": "impianto_irrig1"}] }

Rappresentazione delle risorse

Misurazione:

{"id": 1,
"idColtivazione": 2,
"data": "2021-05-11",
"orario": {"ore": 9, "minuti":10},
"valore":"30"
}

Endpoint

Definire le URI (Uniform Resource Identifier) corrispondenti ai possibili endpoint che permettono di accedere alle risorse; hanno in comune una URI base: **http://api.greenfarms.org** – può anche essere utile includere la versione

Esempio: **/v1/aziende** restituisce un array di oggetti "aziende agricole"

Esempio: **/v1/aziende/14** restituisce l'azienda con id 14

Dato che le aziende possiedono coltivazioni e queste contengono dei device IoT possiamo utilizzare una URI gerarchica

Esempio: **/v1/aziende/14/coltivazioni/10**

È anche possibile aggiungere dei parametri definendo così delle query:

Esempio: **/v1/aziende/14/misure?data=2023-05-12**

Nota: per esperimenti in locale la URI base sarà semplicemente localhost:porta

Azioni CRUD sulle risorse

Possibili azioni sulle risorse:

- Visualizzare le coltivazioni di una data azienda agricola
- Creare una nuova coltivazione
- Modificare una coltivazione esistente
- Cancellare una coltivazione

Associamo ciascuna azione ad un «verbo» http – GET, POST, PUT, DELETE; inoltre consideriamo anche il tipo di errore da restituire in caso di fallimento:

Azioni CRUD sulle risorse

Codici Restituiti

- 200 (successo: GET nel body c’è la risorsa, PUT/POST nel body informazioni sull’esito)
- 201 (POST conferma creazione risorsa, restituisce id)
- 400: richiesta non valida (errore sintassi)
- 401: accesso non autorizzato
- 404: risorsa non trovata
- 500: Errore interno

Tabella delle possibili azioni, con eventuale input e risposta

Verbo http	Endpoint	Input	Output in caso di successo	Messaggio Errore	Descrizione
GET	/aziende/ {idAzienda}/ coltivazioni/	Body: vuoto	Stato: 200 Body: lista coltivazioni	Stato: 500	Fornisce un array di coltivazioni
GET	/aziende/ {idAzienda}/ coltivazione/ {idColtivazione}	Body: vuoto	Stato: 200 Body: dati coltivazione	Stato: 404 o 500	Fornisce la coltivazione {idColtivazione}
GET	/aziende/ {idAzienda}/misure ev. default=data oggi	Body: vuoto Parametro: data (opzionali oraInizio, oraFine)	Stato: 200 Body: lista misure nel giorno (ora)	Stato : 500	Fornisce un array di misure
POST	/richieste_acqua	Body: nuova richiesta	Stato: 201 Body: id della nuova richiesta	Stato: 401, 500	Inserisce nuova richiesta
PUT	/aziende/{id}	Body: nuovi attributi da sostituire	Stato: 200 Body: vuoto	Stato: 401, 404, 500	Modifica dati azienda

Tabella delle possibili azioni, con eventuale input e risposta

Verbo http	Endpoint	Input	Output in caso di successo	Messggio Errore	Descrizione
GET	/aziende/{idAzienda}/ coltivazioni/ {idColtivazione}/ IoTDevs	Body: vuoto	Stato: 200 Body: dati dispositivi presenti nella serra	Stato: 500	Fornisce l'array dei dispositivi della coltivazione {idColtivazione}
DELETE	/aziende/{idAzienda}/ richieste/{idRichiesta}	Body: vuoto	Stato: 200	Stato : 404 o 500	Cancella una richiesta (se esiste)
DELETE	/aziende/{idAzienda}/ richieste	Non Definito	Non Definito	Stato: 400	Azione vietata

Cosa produrre nella fase di progettazione

Progettazione TOPIC e messaggi MQTT

aziendaYYY/coltivazioneXXX/sensori/sensoreTemp oppure

aziendaYYY/coltivazioneXXX /sensoreTempZZZ (o sensoreUmidWWW)

{"tempCelsius": 18.5, "time": data-e-ora}

{"percUmid": 70, "time": data-e-ora}

(simile per tutti gli altri sensori)

aziendaYYY/serraXXX/attuatori/attuatoreIrrig

{"attivo": true, "time_start": data-e-ora, "time_stop": data-e-ora}

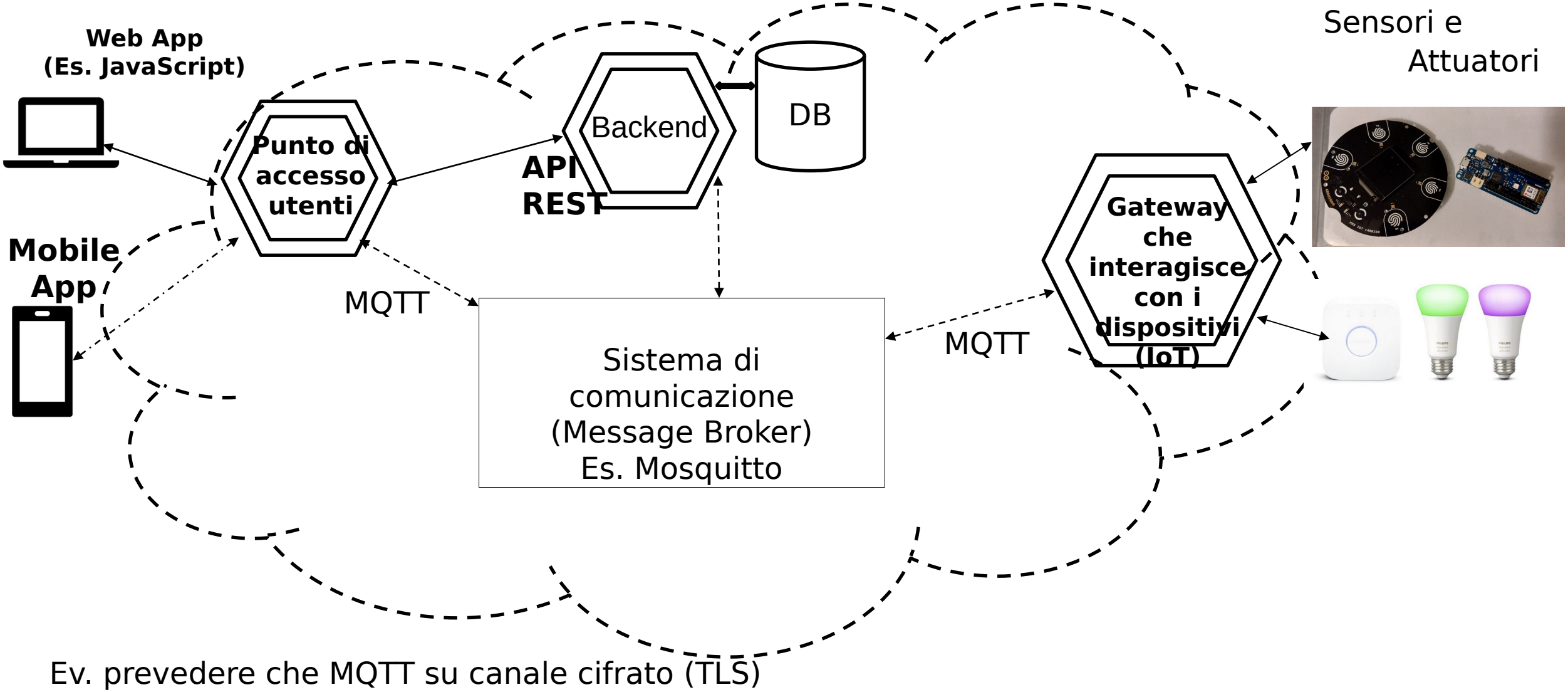
Uso wildcard

Sottoscrizione alle misure più sensori:

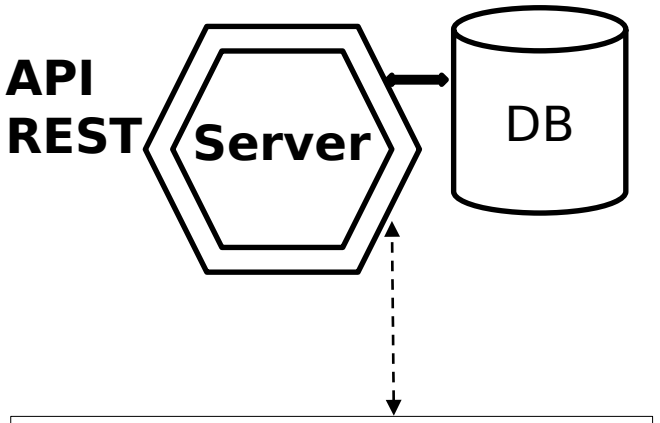
aziendaYYY/serraXXX /sensori/#
Tutti i sensori della serraXXX

aziendaYYY/+/sensori/#
Tutti i sensori di tutte le serre dell’aziendaYYY

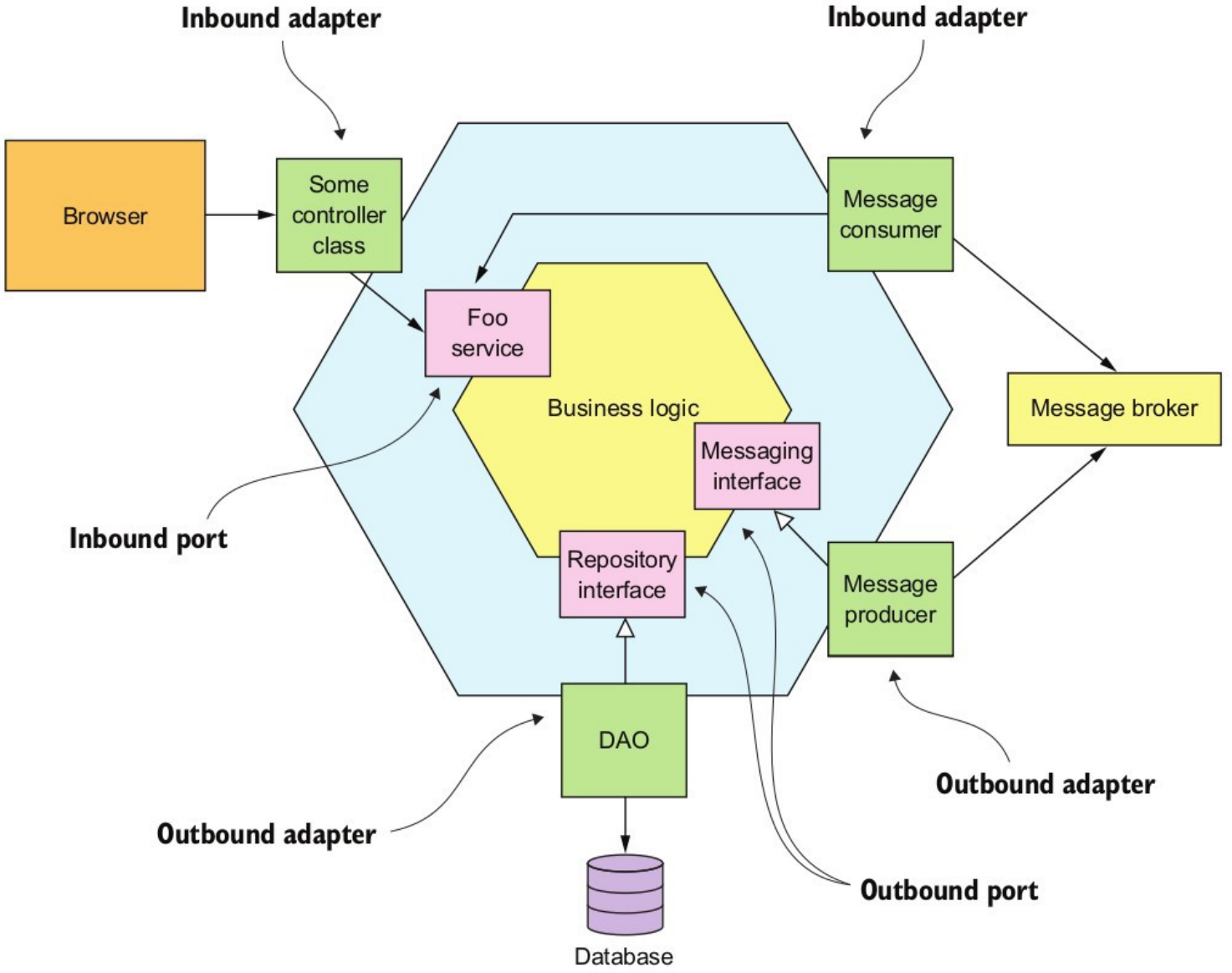
Architettura del sistema IoT



Ev. prevedere che MQTT su canale cifrato (TLS)

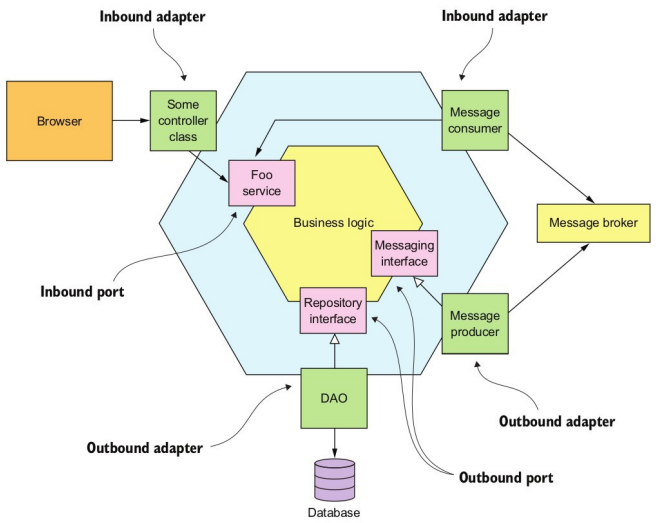


Sistema di comunicazione (Message Broker) Es. Mosquitto



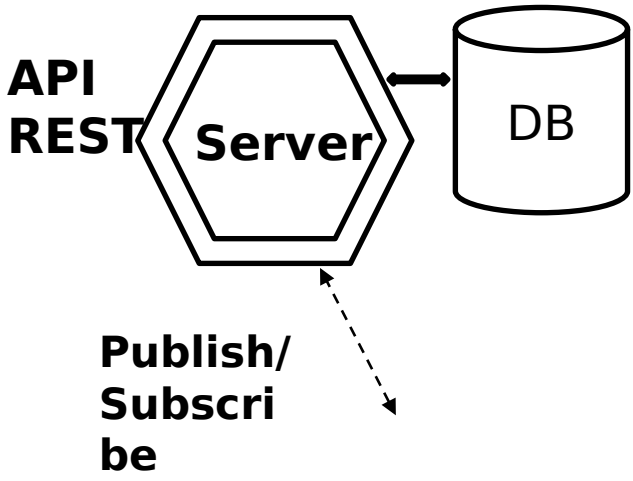
Classi: business logic e adapters

- Interfaccia della business logic: metodi che corrispondono a comandi che cambiano lo stato interno o a query che richiedono risposte
- Adapter
  - Inbound: costituiscono un'interfaccia per il mondo esterno che invia comandi e query al microservizio. Es. API REST
  - Outbound: costituiscono un'interfaccia verso sistemi esterni
    - Database
    - Broker



API REST ADAPTER (Inbound Adapter)

Classi che si occupano di gestire le richieste in arrivo e richiamano i metodi offerti dal cuore del microservizio



Business Logic (gestore serre, piani di irrigazione o illuminazione e misure)
Classi che si occupano di rispondere a comandi/query:

- crea prenotazione, cancella prenotazione, modifica prenotazione, elenco prenotazioni
- identifica irrigatore attuale, notifica inizio/fine irrigazione
- registra misura, elenco misure

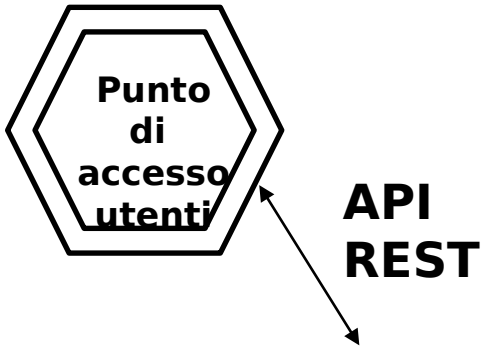
PERSISTENZA (outbound adapter)
Mappa operazioni CRUD sugli oggetti trattati dalla business logic su operazioni sul DB

Broker messaging ADAPTER (In/Outbound Adapter)

- In: riceve misure (via subscribe a broker) – richiama registra misura
- Out: invia notifiche di inizio/fine irrigazione– consuma notifiche

View (UI)

Classi che si occupano di gestire le viste dell'interfaccia utente

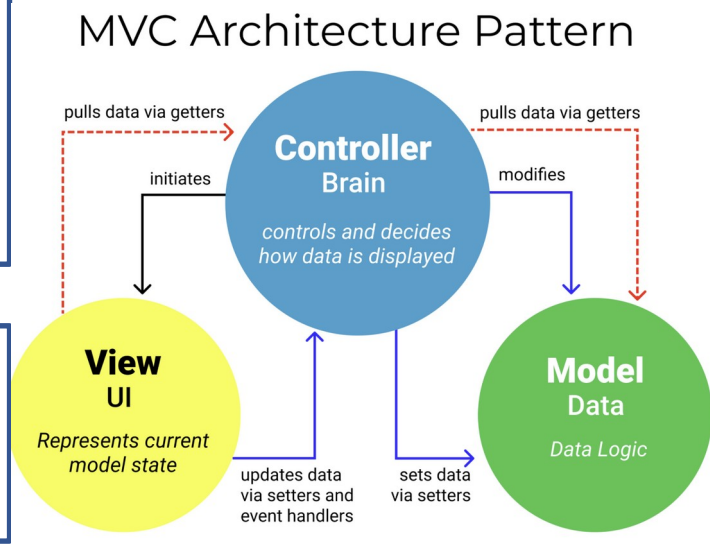


Controller

gestisce la sequenza di interazioni con l'utente (innescate dalla UI) e di conseguenza aggiorna il modello

Model (gestisce gli oggetti del dominio)

- Interagisce con l'outbound adapter
- Aggiorna le viste



Outbound Adapter verso il Servizio che gestisce coltivazioni e misure

IoT ADAPTER (In/Outbound Adapter)
- In: rilevare misure (attraverso il meccanismo di interazione con sensori)
- Out: comanda gli attuatori (es. richiama le API REST delle lampadine)

Gestisce una tabella serre e sensori e i piani di irrigazione in corso
Gestisce le informazioni su misure monitorabili e sullo stato di attuatori
Monitorare i parametri di stato delle serre
Agisce per mantenerli modificando lo stato degli attuatori

Broker messaging ADAPTER (In/Outbound Adapter)
- Out: invia misure (via publish a broker) – richiama lettura sensori
- In: riceve notifiche di inizio/fine irrigazione – comanda gli attuatori