4OURSQUARED

4oursquared.unipd@gmail.com

Lumos Minima

Imola Informatica

Specifica Tecnica

Informazioni	
Redattori	Soldà Matteo
Versione	0.0.1
Uso	esterno

Descrizione

Questo documento descrive la specifica architetturale del progetto, illustrando gli aspetti progettuali che caratterizzino il design del prodotto.

Versione	Data	Redattore	Verificatore	Descrizione
0.0.1	20/09/2023	Soldà Matteo	Alberti Nicolas	Stesura delle prime sezioni.
0.0.0	05/09/2023	Soldà Matteo	Alberti Nicolas	Prima stesura.

Contents

List of Figures

1 Introduzione

1.1 Scopo del Documento

Questo documento ha lo scopo di descrivere nel dettaglio, sopratutto tramite diagrammi, le caratteristiche architetturali del prodotto software sviluppato.

1.2 Riferimenti Normativi

- Capitolato C2 - Lumos Minima

1.3 Riferimenti Informativi

- Design Pattern Architetturali
- Dependency Injection
- MVC e Derivati
- Pattern Creazionali
- Software Architecture
- Pattern Strutturali
- Pattern Comportamentali
- Programmazione SOLID

2 Tecnologie Utilizzate

2.1 Frontend

Per realizzare il frontend, ossia la parte di applicazione che viene eseguita sul browser dell'utente, sono state utilizzate le seguenti tecnologie:

- React: libreria JavaScript per la creazione di interfacce utente;
- Typescript: linguaggio di programmazione che estende JavaScript aggiungendo i tipi, permettendo una codifica più robusta e sicura;
- Bootstrap: framework CSS per la creazione di interfacce responsive e accattivanti;

2.2 Backend

Per realizzare il backend, ossia la parte di applicazione che viene eseguita sul server, sono state utilizzate le seguenti tecnologie:

- Node.js: runtime JavaScript che permette di eseguire codice JavaScript lato server;
- Typescript: linguaggio di programmazione che estende JavaScript aggiungendo i tipi, permettendo una codifica più robusta e sicura;
- Express: framework che permette di creare applicazioni web e API più facilmente e con una miglior gestione;
- Axios: client HTTP basato su promise per effettuare richieste HTTP basate su Promise;
- Mongoose: libreria che permette di gestire in modo più semplice e intuitivo i database MongoDB;
- Cors: middleware che permette di configurare in modo semplice e veloce le politiche CORS;
- JWT: libreria che permette di gestire in modo semplice e veloce i token JWT;
- Cron: libreria che permette di gestire in modo semplice e veloce i cronjob, ossia per la creazione di routine automatiche;

2.3 Database

Il database utilizzato è di tipo NoSQL, in particolare MongoDB, che permette di gestire documenti in formato BSON (Binary JSON).

2.4 Meccanismo di Comunicazione

Tutte le comunicazioni, sia esterne (da client a server) che interne (da server a server) sono state gestite tramite API REST.

3 Architettura Logica

3.1 Frontend

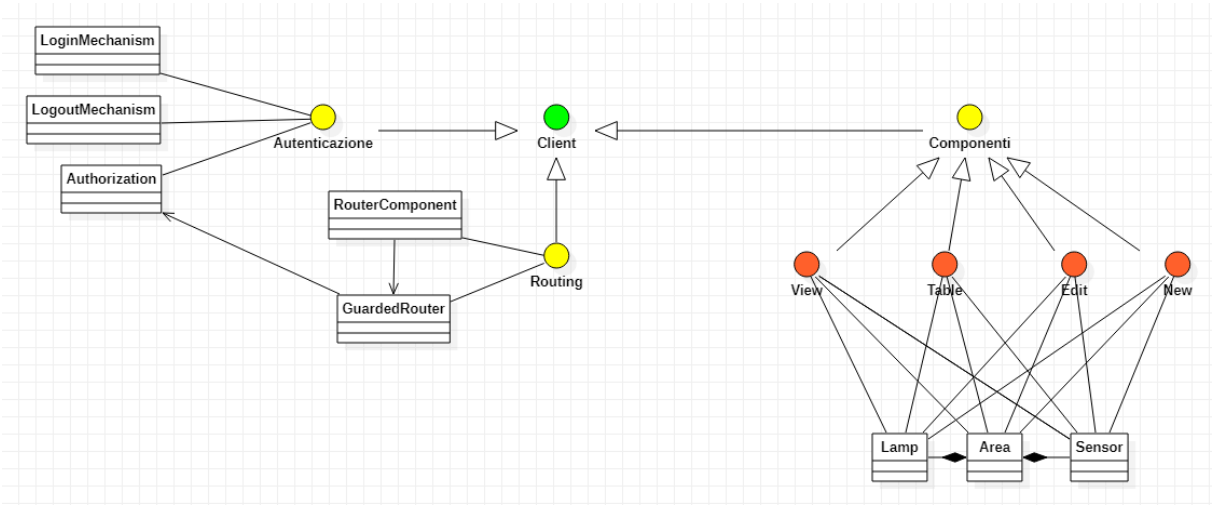

Figure 1: Architettura del client.

3.2 Backend

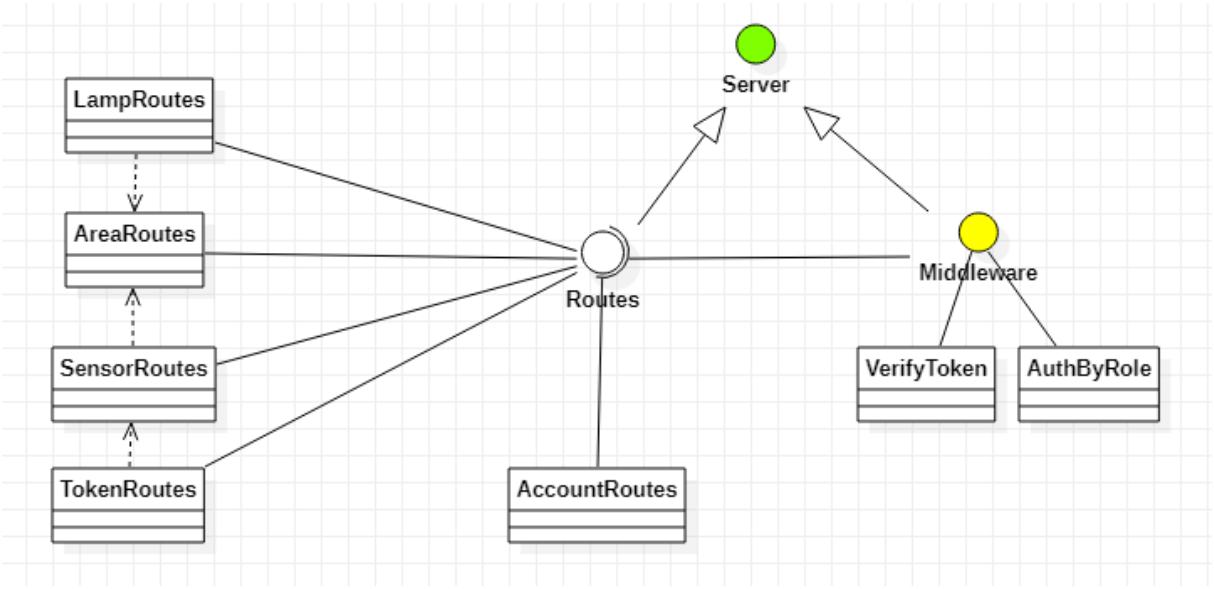

Figure 2: Architettura del server.

4 Design Pattern

Lumos Minima è un'applicazione web di tipo SPA (Single-page application) per la gestione di impianti luminosi e di sensori per la rilevazione del movimento associati ai primi, raggruppati in insiemi definiti aree illuminate.
L'applicazione si avvale del cosiddetto stack MERN (dove le iniziali stanno rispettivamente per MongoDB, Express.JS, React.JS, Node.JS) e presenta un design architetturale "3-tier" dotata di un comparto client (presentazione), un server ("business logic"), e un database No-SQL per ospitare i dati (persistance). Tuttavia, invece di JavaScript "puro" si è preferito TypeScript, che viene compilato in JavaScript e fornisce un supporto opzionale della tipizzazione stretta. Il diagramma mostra le tecnologie di cui si serve ciascun layer.

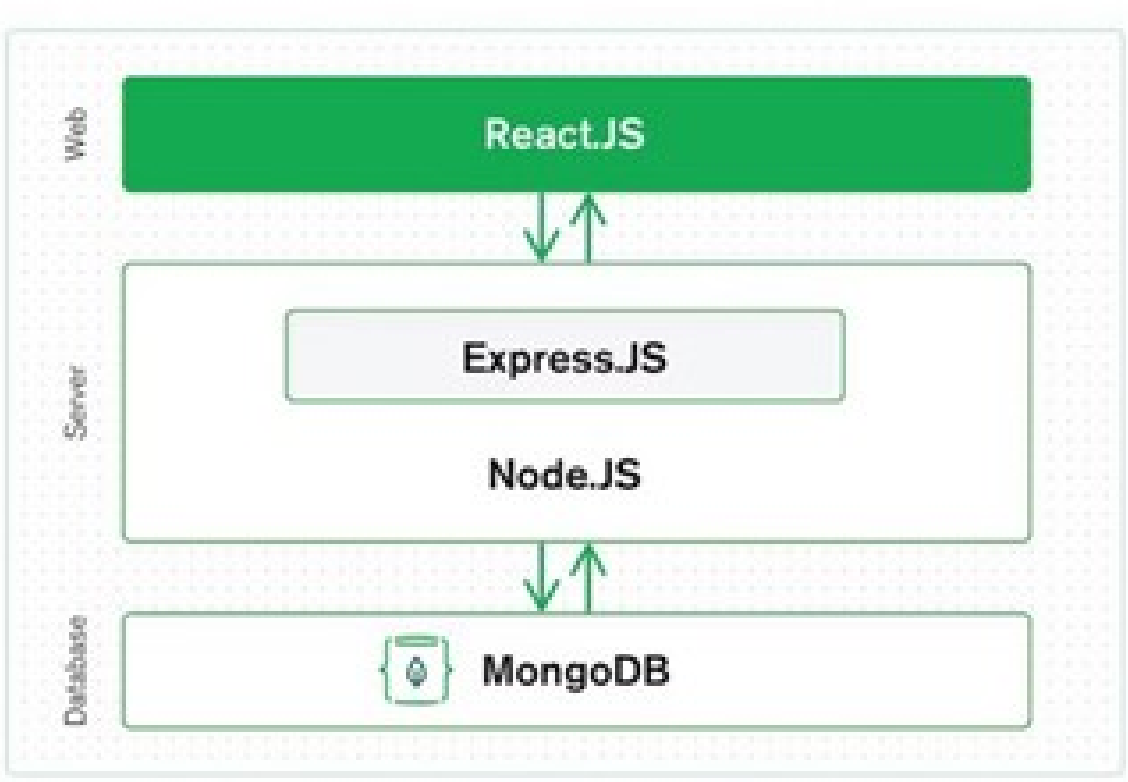

Figure 3: Rappresentazione grafica dell'architettura MERN.

Questo sistema "a livelli" è stato scelto poiché, oltre a trattarsi di un sistema consolidato (e quindi che si prestasse a uno sviluppo anche gravato da vincoli temporali stringenti), consente in modo agevole il mock del layer sottostante in occasione dei test di unità, vista la necessità di raggiungere la percentuale di code coverage inizialmente indicata nel capitolato.

5 Altri Aspetti Progettuali Rilevanti

5.1 Persistenza dei Dati

Per realizzare lo strato di persistenza dei dati, come precedentemente indicato, è stato utilizzato MongoDB, un database NoSQL che permette di gestire documenti in formato BSON (Binary JSON).

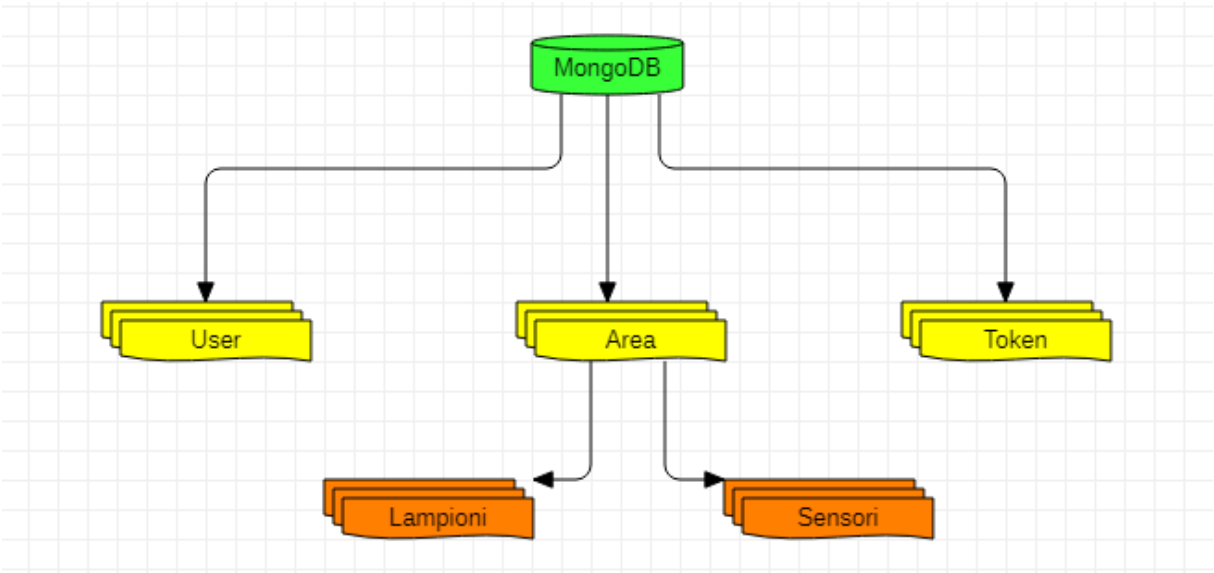

Figure 4: Struttura del database.

A differenza di un database di tipo SQL, in MongoDB, i dati sono raccolti in *collezioni*. Queste *collezioni* contengono uno, nessuno o una moltitudine di *documenti*.
Dato che nei database di tipo NoSQL non esiste il concetto di "relazione", ogni documento di tipo area conterrà un array di documenti di tipo sensore e un array di documenti di tipo lampione.

Per quanto riguarda invece la comunicazione tra il server e il database, per poter comunicare i dati tra una parte e l'altra, sono stati utilizzati gli schemi di *Mongoose* che permettevano di definire la struttura dei dati, i tipi di dati, i vincoli e le validazioni, oltre a fornire una interfaccia che permettesse di comunicare con il giusto tipo di dati.
Di seguito la struttura dei documenti (e conseguentemente degli schemi) utilizzati:

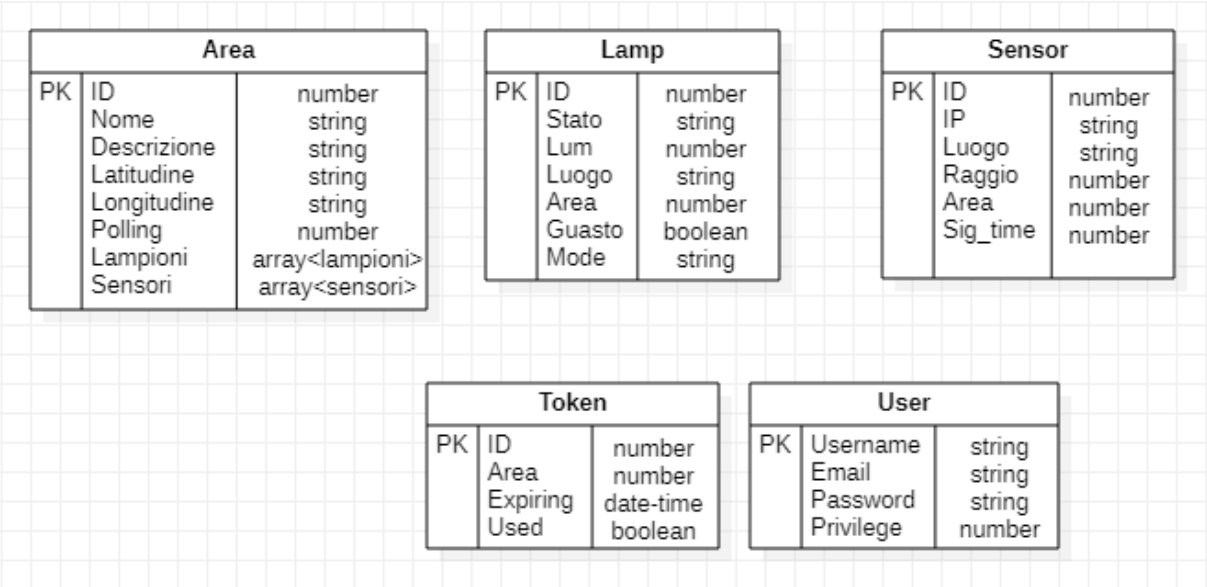

Figure 5: Struttura dei dati nel DB.

5.2 Interfacciamento tra Sensori e Lampioni

Per accedere ai dati riguardanti un determinato lampione e/o sensore, abbiamo utilizzato la lettura di parametri dalla URL.
Ovviamente gli endpoint relativi ai sensori e ai lampioni sono distinti, infatti:

- Endpoint sensori: */api/aree/<IdArea>/sensori/<IdSensore>*;
- Endpoint lampioni: */api/aree/<IdArea>/lampioni/<IdLampione>*;

5.3 Comunicazione Automatica

Per quanto riguarda il meccanismo che permette di aumentare e diminuire automaticamente la luminosità dei lampioni che si trovano dentro una determinata area illuminata, il soggetto principale sono i sensori: essi, qualora rilevassero il movimento di una persona, di un veicolo o di un animale, dovrebbero inviare una richiesta di tipo HTTP all'endpoint prestabilito. Alla ricezione della richiesta, il server si occuperà della gestione della stessa come descritto nel diagramma sottostante.

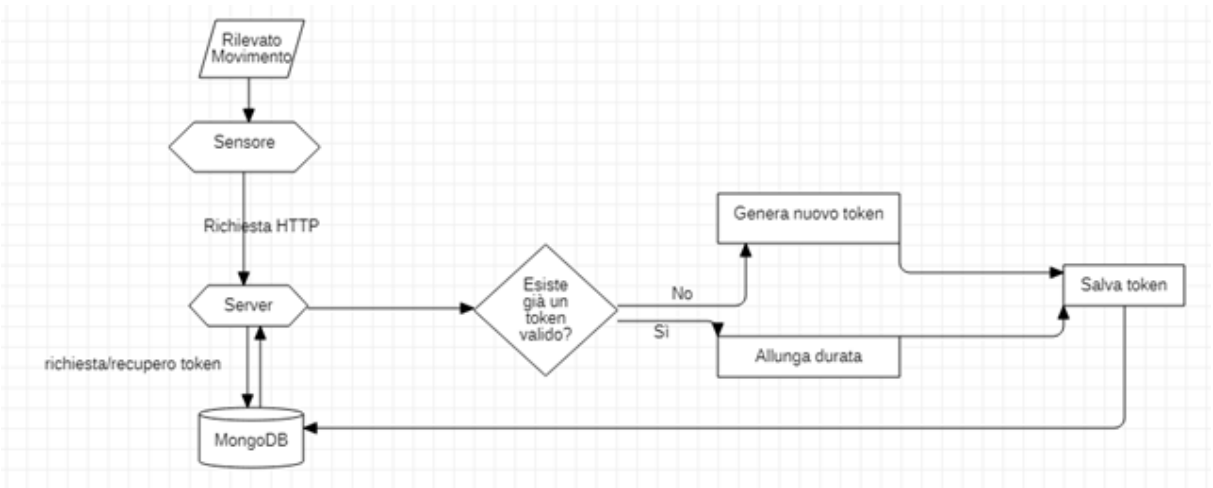

Figure 6: Diagramma di flusso del segnale relativo al movimento rilevato dai sensori.

Il token trova un utilizzo concreto solo per i lampioni dell'area che sono configurati in modalità PULL (manuale).
Per questo motivo, di seguito c'è una piccola spiegazione di come viene gestito il rilevamento di movimento da entrambi i tipi di lampione:

- PUSH: i lampioni si illuminano quando il sensore segnala un movimento, senza necessità di utilizzare il token
- PULL: la parte di server vhe gestisce l'area illuminata, con cadenza regolare e definita dall'atributo *polling time*, controlla nel database se c'è un token valido per l'area di riferimento:
  - Se il token è valido e inutilizzato: illumina tutti i lampioni impostati in modalità PULL;
  - Se il token non è valido ma è stato utilizzato: riduce la luminosità di tutti i lampioni a quella iniziale;
  - Negli altri casi, verrà restituito un codice stato consono, ma senza effettuare nessun'altra azione.

5.4 Autenticazione e Autorizzazione tramite JWT

Dopo aver inserito le proprie credenziali nella login mask, se la password (criptata con hashing SHA512) coincide con quella dell'utente viene restituito un token JWT firmato dal server, una stringa che contiene informazioni sul ruolo dell'utente. L'autorizzazione (necessaria per l'accesso alle pagine dal lato "client", e per utilizzare le API del server) sfrutta questo meccanismo:

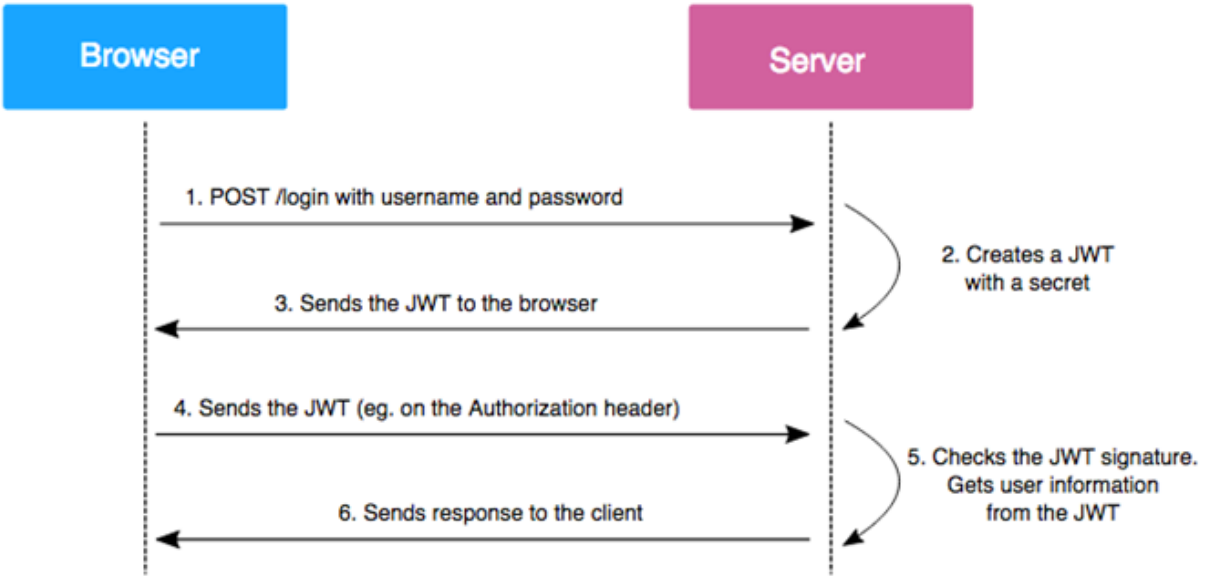

Figure 7: Diagramma di flusso del meccanismo di autorizzazione tramite JWT.

Tuttavia, il JWT è memorizzato (e inviato al server) all'interno di un cookie HTTP-Only: questo impedisce che il contenuto del cookie possa essere letto da codice JavaScript malevolo, vanificando così attacchi alla confidenzialità di tipo XSS (Cross-Site Scripting).

Le API sono protette da due livelli di middleware che prendono in carico la richiesta HTTP.

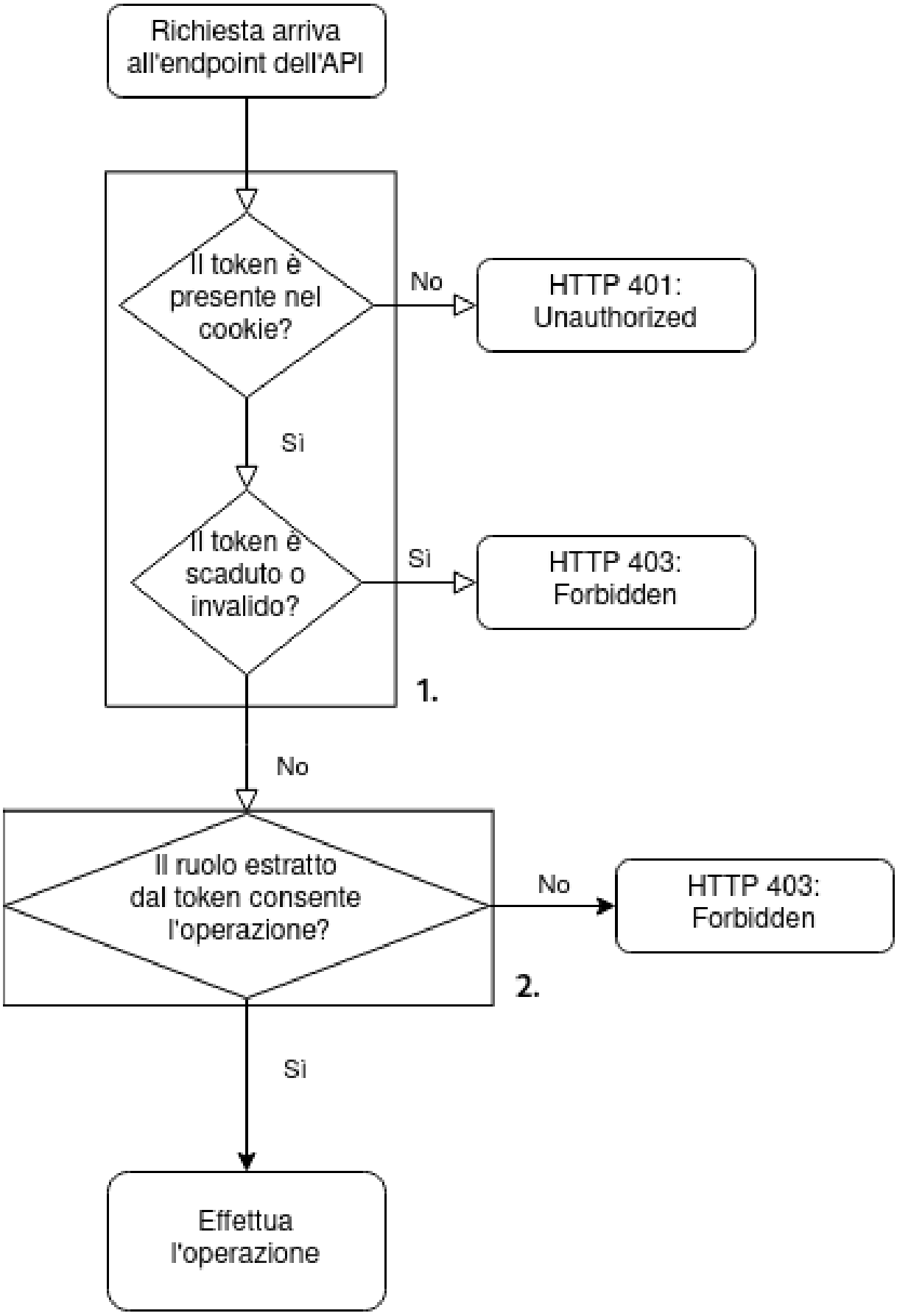

Figure 8: Diagramma di flusso del meccanismo di autorizzazione tramite middleware.