4OURSQUARED

4oursquared.unipd@gmail.com

Lumos Minima

Imola Informatica

Motivazione delle Scelte

Informazioni	
Redattori	Ceccato Francesco Soldà Matteo
Versione	1.0.0
Uso	esterno

Descrizione

Questo documento riporta i confronti tra i vari linguaggi presi in considerazione per lo sviluppo del progetto e la motivazione della scelta di uno rispetto agli altri.

Versione	Data	Redattore	Verificatore	Descrizione
1.0.0	20/07/2023	Soldà Matteo	Alberti Nicolas	Approvazione documento per candidatura RTB
0.1.0	20/07/2023	Soldà Matteo	Alberti Nicolas	Verifica del documento per RTB
0.0.1	18/05/2023	Soldà Matteo	Alberti Nicolas	Aggiunta decisione preliminare dei linguaggi e framework
0.0.0	09/05/2023	Soldà Matteo	Ceccato Francesco	Prima stesura.

Contents

1 Introduzione

Il seguente documento ha come scopo principale quello di confrontare i vari linguaggi, database e framework con lo scopo di fornire un prodotto quanto più efficace, efficiente e aderente alle richieste del proponente.

1.1 Sitografia

- https://www.quora.com/Why-are-there-so-many-backend-systems-written-in-dynamically-typed-languages-while-the-benefit-of-static-typing-is-obvious;
- https://mobisoftinfotech.com/resources/blog/typescript-vs-javascript/;
- https://stackoverflow.com/questions/49640121/mixing-javascript-and-typescript-in-node-js;
- https://www.monocubed.com/blog/top-python-frameworks/;
- https://www.theserverside.com/tip/The-differences-between-Java-and-TypeScript-devs-must-know;
- https://medium.com/rewrite-tech/embedding-typescript-in-java-a343576031be;
- https://blog.geekandjob.com/framework-javascript/.

2 Confronti

2.1 Database

Come database, non sapendo al momento come saranno strutturati i dati che necessitano di essere salvati in maniera persistente, abbiamo preso in considerazione MariaDB per i database SQL e MongoDB per i database NoSQL.
Queste due scelte sono state fatte per due motivi principali:

- Ricerca di un database computazionalmente leggero e che fosse aderente agli standard;
- Ricerca di un database la cui community è attiva e la documentazione chiara e puntuale, così da poter risolvere quanto prima i problemi che si potessero presentare data la poca esperienza accumulata con i database nel tempo.

2.2 Backend

Per quanto riguarda il backend, la scelta sarà vincolata dalla preferenza verso un linguaggio staticamente o dinamicamente tipizzato.
Qualora si scegliesse la tipizzazione statica, le scelte ricadono su:

- Java;
- C#;
- TypeScript.

Mentre, scegliendo la tipizzazione dinamica, le scelte ricadrebbero su:

- Python;
- JavaScript.

2.2.1 C#

Pro:

- Facile da eseguire su dispositivi Windows (che rappresenta la stragrande maggioranza degli utenti);
- Essendo un linguaggio compilato e non interpretato, rende difficile in caso di attacchi la lettura del codice sorgente in chiaro.

Contro:

- Deve essere ricompilato per ogni minima modifica;
- Hostabile solo su server Windows.

2.2.2 Java

Pro:

- Linguaggio facile da imparare;
- Linguaggio di alto livello (assenza di puntatori, presenza di garbage collector);
- Linguaggio OOP;

- *Platform Indipendent*;
- Supporto al *multithreading*$_G$.

Contro:

- Rispetto ad altri linguaggi può risultare più lento o povero in fatto di performance;
- Mancanza di feature per il backup dei dati;
- Rispetto ad altri linguaggi, richiede uno spazio di memoria significativo;
- Rispetto ad altri linguaggi, usa molto più codice per svolgere una operazione basilare.

2.2.3 JavaScript

Pro:

- Semplice da imparare;
- Interoperabile;
- Versatile in quanto linguaggio *fullstack*$_G$.

Contro:

- Linguaggio interpretato;
- Tipizzazione dinamica;
- Inconsistenza tra browser.

2.2.4 Python

Pro:

- Facile da imparare;
- Flessibile ed estensibile;
- Presenza di moltissime librerie;
- Estremamente scalabile;
- Portabile;
- Vasta integrazione con l'*IoT*$_G$.

Contro:

- Forte utilizzo di memoria;
- Tipizzazione dinamica;
- Multithreading complesso;
- Garbace collection che porta ad un potenziale spreco di memoria.

2.2.5 Typescript

Pro:

- Linguaggio compilato;
- Tipizzazione statica.

Contro:

- Inconsistenza tra browser;
- Tipizzazione alcune volte complessa;
- Necessità di ricompilare per ogni minima modifica.

2.3 Backend Framework

2.3.1 C#

- ASP.NET.

2.3.2 Java

- Java Spring.

2.3.3 JavaScript

- Next;
- Node.

2.3.4 Python

- Django;
- Flask.

2.3.5 Typescript

- Nest;
- Feathers;
- Loopback.

2.4 Frontend

Escludendo *HTML5*, *XHTML* e *CSS* che saranno di sicuro usati per la parte di frontend della webapp, i linguaggi tra cui scegliere risultano essere solamente

- TypeScript;
- JavaScript.

2.4.1 JavaScript

2.4.2 TypeScript

2.5 Frontend Framework

2.5.1 JavaScript

- Angular;
- React;

- Vue;
- Backbone;
- Preact;
- Express.

2.5.2 TypeScript

- Angular;
- Ember.

Per i pro e i contro dei framework, sia frontend che backend, si rimanda il lettore a questo link.

3 Scelte e Motivazioni

A seguito di una votazione unanime del gruppo, si è pensato di utilizzare il framework React per il frontend e il framework Node per il backend, entrambi utilizzeranno TypeScript insieme a JSX. Queste scelte sono dovute a diversi fattori, nello specifico:

- Si è scelto TypeScript al posto di JavaScript in quanto utilizza una tipizzazione statica e non dinamica, portando di conseguenza i seguenti vantaggi:
  - Rilevazione degli errori a tempo di compilazione: nei linguaggi a tipizzazione statica, gli errori di tipo vengono rilevati durante la fase di compilazione; ciò significa che gli errori di tipo, come l'uso di un tipo errato di dato o la mancanza di coerenza tra i tipi, vengono evidenziati prima che il programma venga eseguito. Questo aiuta a prevenire errori costosi o bug difficili da individuare in fase di esecuzione;
  - Maggiore robustezza e sicurezza: la dichiarazione esplicita dei tipi di dati può aiutare gli sviluppatori a comprendere meglio l'intento del codice e a individuare eventuali incongruenze o problemi potenziali. Inoltre, poiché i tipi sono controllati a tempo di compilazione, i programmi tipizzati staticamente tendono ad avere meno errori di tipo durante l'esecuzione;
  - Ottimizzazione delle performance: i linguaggi tipizzati staticamente possono spesso beneficiare di ottimizzazioni a livello di compilazione. Il compilatore può effettuare analisi statiche più avanzate, come l'inferenza di tipo o la rilevazione di pattern specifici, per generare codice più efficiente. Di conseguenza, i programmi tipizzati staticamente possono ottenere migliori prestazioni rispetto ai programmi tipizzati dinamicamente;
  - Documentazione implicita: nei linguaggi con tipizzazione statica, i tipi di dati possono fungere da documentazione implicita del codice. La dichiarazione esplicita dei tipi di dati aiuta gli sviluppatori a capire come utilizzare correttamente le funzioni e le variabili, migliorando la leggibilità e la manutenibilità del codice;
  - Semplicità nella collaborazione e per la scalabilità: nei progetti di grandi dimensioni o quando si lavora in team, la tipizzazione statica può favorire una migliore collaborazione. La dichiarazione dei tipi rende più chiaro come le diverse parti del sistema interagiscono tra loro, facilitando l'integrazione del codice scritto da più persone. Inoltre, la tipizzazione statica può aiutare a identificare e prevenire errori di integrazione durante il processo di sviluppo e di manutenzione del software.
- Si è scelto di utilizzare React e NodeJS in quanto consigliati dall'azienda proponente, rendendo quindi possibile la comunicazione con la stessa per la risoluzione di problemi di qualsiasi entità dovuta alla poca conoscenza dei framework, inoltre:
  - React è stato scelto per i seguenti motivi:
    - Component-based architecture: utilizza un'architettura basata su componenti, che consente di suddividere l'interfaccia utente in componenti modulari e riutilizzabili. Questo approccio favorisce la modularità del codice, semplifica la gestione dello stato e offre una maggiore scalabilità del progetto;
    - Virtual DOM: utilizza un concetto chiamato "Virtual DOM" (DOM virtuale). Questo significa che React mantiene una rappresentazione leggera

del DOM in memoria e si occupa di aggiornare solo le parti dell'interfaccia utente che sono state effettivamente modificate. Questo approccio porta a un'efficienza notevole nelle operazioni di rendering e aggiornamento dell'interfaccia utente, migliorando le prestazioni complessive dell'applicazione;
    - Reattività e gestione dello stato: fornisce un'eccellente gestione dello stato dell'applicazione. Attraverso il concetto di "state" (stato) e "props" (proprietà), è possibile gestire in modo semplice e prevedibile i dati dell'applicazione e reagire ai cambiamenti in modo efficiente. Inoltre, React offre un'ampia gamma di librerie e soluzioni per la gestione dello stato, come Redux e MobX, che semplificano ulteriormente la gestione delle informazioni in un'applicazione complessa;
    - Ampia comunità e supporto: ha una delle comunità di sviluppatori più grandi e attive nel panorama dello sviluppo frontend. Ciò significa che è possibile trovare una vasta gamma di risorse, documentazione, tutorial e librerie di terze parti per supportare lo sviluppo di progetti. Inoltre, la comunità offre costantemente aggiornamenti, nuove funzionalità e correzioni di bug per React, garantendo che il framework sia all'avanguardia;
    - Ecosistema ricco: è supportato da un vasto ecosistema di strumenti e librerie. Ci sono numerose librerie, come React Router per il routing, Axios per le richieste HTTP e molti altri che semplificano lo sviluppo di funzionalità specifiche;
    - Compatibilità con TypeScript: ha un'ottima integrazione con TypeScript, un linguaggio di programmazione con tipizzazione statica. Utilizzando React con TypeScript, puoi ottenere i benefici della tipizzazione statica, come rilevazione degli errori a tempo di compilazione e autocompletamento, per una migliore esperienza di sviluppo e una maggiore sicurezza del codice.
- NodeJS è stato scelto per i seguenti motivi:
    - Efficienza e scalabilità: è noto per la sua efficienza e scalabilità. Grazie al suo modello di I/O non bloccante e orientato agli eventi, Node.js può gestire un grande numero di richieste simultanee con una risposta rapida. Ciò lo rende particolarmente adatto per applicazioni in tempo reale e applicazioni con un alto traffico;
    - Comunità e supporto: ha una comunità di sviluppatori attiva e una vasta quantità di risorse disponibili. Ciò significa che si può trovare un supporto significativo, documentazione, tutorial e librerie di terze parti a supporto delle difficoltà che si potrebbero incontrare;
    - Adattabilità: è un runtime flessibile che può essere utilizzato per diversi tipi di progetti. Può essere utilizzato per sviluppare API RESTful, servizi web, applicazioni in tempo reale, microservizi e molto altro ancora. La sua natura leggera e modulare consente di adattarlo alle specifiche esigenze del tuo progetto.

Inoltre, per lo sviluppo del *Proof of Concept* si è deciso di utilizzare *MongoDB* in quanto la struttura degli oggetti che verranno utilizzati (quali sensori, lampioni e aree illuminate) sono facilmente adattabili ad un formato *JSON*, fortemente supportato dalle collezioni del database prescelto.