Università degli Studi di Modena e Reggio Emilia

Dipartimento di Ingegneria "Enzo Ferrari"

Corso di Laurea in Ingegneria Informatica

Progettazione e sviluppo di una piattaforma riutilizzabile in contesto aziendale

Relatore:
Prof. Francesco Guerra

Presentata da:
Matteo Sirri

Anno Accademico 2020-2021

Abstract

Indice

Capitolo 1

Introduzione

1.1 Obiettivo

testo

1.2 Campo di applicazione

testo

1.3 Panoramica

testo

Capitolo 2

Descrizione generale

2.1 Inquadramento

testo

2.2 Macrofunzionalità del sistema

testo

2.3 Caratteristiche degli utenti

testo

2.4 Vincoli generali

testo

2.5 Analisi future

testo

Capitolo 3

Tecnologie

3.1 Implementazione

In questa sezione verranno descritti gli strumenti utilizzati per implementare i componenti che permettono alla piattaforma di erogare i propri servizi. La motivazione principale che ha portato alla scelta delle tecnologie di seguito elencate è il *know-how* aziendale.

3.1.1 Linguaggio

Typescript

Typescript[1] è un linguaggio open-source sviluppato da Microsoft.

È un *super-set* del linguaggio JavaScript, permettendone l'estensione con l'introduzione di un meccanismo di tipizzazione statico e il supporto alla programmazione orientata agli oggetti. Per via della sua natura può essere utilizzato in tutti i contesti in cui viene usato JavaScript grazie ad un processo di transpilazione che traduce codice Typescript in codice JavaScript, permettendone così una successiva compilazione ed esecuzione.

3.1.2 Ambiente di sviluppo

Node.js

Node.js[2] è un ambiente runtime JavaScript open-source e multipiattaforma.

Le caratteristiche fondamentali sono: l'esecuzione dell'engine V8, sviluppato da Google, che permette di compilare ed eseguire codice JavaScript al di fuori di browser web, l'uso di un insieme di primitive I/O asincrone di tipo non bloccante e l'esecuzione di applicazioni su un solo processo, senza generazione di nuovi thread per ogni richiesta. Questo sta a significare che quando si deve eseguire una operazione I/O, come una richiesta ad un web server, Node.js non blocca il thread, mettendo in attesa la CPU, ma, al contrario, la lascia libera di portare avanti altri compiti e si occuperà di ripristinare l'operazione non appena arriverà una risposta utilizzando una *callback*

Grazie a queste peculiarità è possibile realizzare applicazioni performanti in grado di gestire connessioni concorrenti con un singolo server.

In questo ambiente è poi possibile utilizzare lo standard ECMAScript in modo flessibile in quanto è possibile modificare il set di funzionalità abilitate, potendo così adattarsi al meglio nei vari contesti di utilizzo.

Infine, Node.js permette anche di aumentare la produttività di un team di sviluppo perchè fornisce agli sviluppatori *front-end*, che conoscono il linguaggio JavaScript, la possibilità di sviluppare codice *server-side*; senza dover imparare un linguaggio del tutto nuovo. Grazie alle sue caratteristiche Node.js risulta essere un'ottimo strumento per lo sviluppo di servizi web.

3.1.3 Node Package Manager

Node Package Manager[3] (NPM) è un *software registry* per applicazione Node.Js. Questo si compone di due parti principali: il registro e una *Command Line Interface*(CLI).

Il primo è una raccolta di librerie open-source che permettono di integrare in una applicazione numerose funzionalità e che può favorire la condivisione di codice, anche con l'uso di registri privati.

Il secondo permette di interagire con il registro e gestire le dipendenze del progetto. In particolare, il meccanismo di gestione delle dipendenze di NPM permette di gestire con semplicità i pacchetti sul quale dipendende una applicazione grazie all'utilizzo di un file particolare chiamato *package.json*. Al suo interno sono infatti raccolte tutte le informazioni relative alla applicazione, gli script eseguibili e l'elenco delle dipendenze. Questo risulta essere di fondamentale importanza perchè permette ad un team di sviluppo di avere un meccanismo che garantisce consistenza tra i vari ambienti usati dagli sviluppatori.

3.1.4 Framework

NestJS

NestJS [4] (Nest) è un framework basato su Node.js per realizzare delle web *Application programming interface* (API) e microservizi. Offre supporto sia il JavaScript che il TypeScript e combina elementi di programmazione ad oggetti e programmazione funzionale.

Nel dettaglio questo framework si pone come un layer di astrazione tra lo sviluppatore e un server HTTP basato su Express.js o Fastify (due framework per realizzare server web veloci e flessibili). Grazie a questo è inoltre possibile usufruire tutti i componenti aggiuntivi compatibili con la piattaforma sottostante, con ovvi vantaggi in termini di riusabilità e flessiblità.

Altro aspetto significativo di questo framework è che guida lo sviluppatore a realizzare una applicazione con una architettura *three-tier* ("a tre strati"), ovvero suddividendo gli elementi principali in 3 strati dedicati alla gestione delle richieste dell'utente, alla gestione della logica funzionale e alla gestione dei dati.

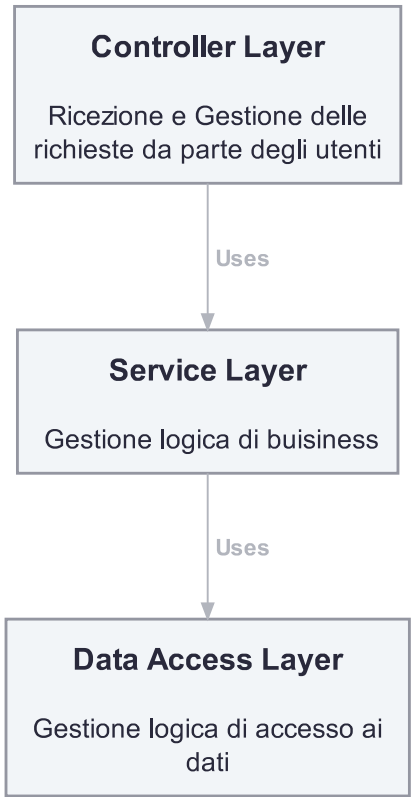

Figura 3.1: Schema riassuntivo della architettura *three-tier*

Questa architettura viene supportata grazie a dei componenti di base, offerti dal framework stesso, che possono essere estesi dallo sviluppatore in base alle proprie esigenze. Importante caratteristica di tutti questi componenti è che fanno largo uso della *Depenceny Injection*[1].

Pertanto l'utilizzo di Nest offre agli sviluppatori utili strumenti per velocizzare lo sviluppo di una web API prestando attenzione alle performance e alla architettura software del prodotto da realizzare risultando un'ottima scelta per la realizzazione si servizi web.

3.1.5 Testing

Jest

testo

3.2 Gestione dati

3.2.1 Database

testo

[1]La *Dependency Injection* è un meccanismo che permette di applicare l'inversione del controllo ad un compoente software. In generale permette ad una classe di non dover configurare le proprie dipendenze in modo statico perchè vengono configurate dall'esterno. Ciò offre grossi vantaggi in termini di riusabilità e rende la fase di test molto più semplice.

MongoDB

testo

3.3 Servizi esterni

3.3.1 AWS SES

testo

3.4 Protocolli comunicazione

3.4.1 HTTP

testo

3.4.2 RabbitMQ

testo

3.5 Sicurezza

3.5.1 Autenticazione

testo

3.5.2 Autorizzazione

testo

JWT

testo

OAuth2.0

testo

3.6 Gestione codice condiviso

3.6.1 Git

testo

3.6.2 Monorepo

testo

Nx

testo

3.7 Distribuzione

3.7.1 Docker

testo

3.7.2 Jenkins

testo

3.7.3 Gitlab CI

testo

3.8 Deployment

3.8.1 AWS

testo

Capitolo 4

Architettura

4.1 Descrizione generale

4.2 API Server

4.2.1 Descrizione generale

testo

4.2.2 Principi di design

testo

4.2.3 Auth module

testo

4.2.4 Demo module

testo

4.2.5 User module

testo

4.2.6 Mail module

testo

4.3 Mailer microservice

4.3.1 Descrizione generale

testo

4.3.2 Principi di design

testo

4.3.3 Template Service

testo

4.3.4 Transport Service

testo

4.4 Database Server

4.4.1 Descrizione generale

testo

4.4.2 Modellazione dati

testo

4.5 MQTT Server

4.5.1 Descrizione generale

testo

4.5.2 Principi di design

testo

Capitolo 5

Distribuzione

5.1 Descrizione generale

testo

5.2 CI/CD pipeline

5.2.1 Descrizione generale

testo

5.2.2 Motivazioni

testo

5.2.3 Integrazione continua

testo

5.2.4 Distribuzione continua

testo

5.2.5 Deployment continuo

Capitolo 6

Conclusioni

6.1 Valutazioni complessive

6.2 Sviluppi futuri

Fonti bibliografiche e sitografia

[1] Typescript. [Online]. Available: https://www.typescriptlang.org/

[2] Node.js. [Online]. Available: https://nodejs.org/en/about/

[3] N. P. Manager. [Online]. Available: https://docs.npmjs.com

[4] NestJS. [Online]. Available: https://docs.nestjs.com/

Elenco delle figure