Università degli Studi di Milano
Dipartimento di Informatica
SSRI

Software Oriented Architecture
Relazione Progetto

Implementazione di API REST per una Web Application di gestione degli ingressi in ufficio

Studente
Francesco Avantaggiato
Matr: 942790

Anno Accademico 2021-2022

Indice

1 Introduzione

La necessità di digitalizzazione all'interno di aziende o nella pubblica amministrazione, si scontra con la burocrazia "cartacea" ancora molto presente in queste realtà.
L'idea di questo progetto web nasce proprio dalla possibilità di rendere digitale anche dei piccoli e semplici processi che vengono ripetuti giornalmente all'interno di alcuni uffici che si occupano di ricevere il pubblico.
Inizialmente viene proposta all'utente un'autenticazione tramite Google, implementata con OAuth2, il protocollo di rete di tiplogia "access delegation" che consente l'accesso all'applicazione tramite API di sicurezza.
Questa applicazione web consente di tenere traccia degli ingressi del pubblico in un ufficio attraverso l'inserimento di informazioni tramite un form apposito, in particolare dati relativi alla persona, orario di ingresso, motivazioni della visita.
I dati inseriti sono memorizzati in un database NoSQL (Not Only SQL), non relazionale, implementato con MongoDB e sono trasmessi in formato JSON (JavaScript Object Notation), utilizzato come alternativa ad XML .
In caso di necessità vi è la possibilità di richiedere assistenza all'amministratore del sistema attraverso un Help Desk composto da un form contente le informazioni del richiedente, il messaggio da trasmettere ed un tasto di invio per l'inoltro automatico di email.
Inoltre l'amministratore del sistema, per essere informato sulla situazione generale del suo ufficio, a fine giornata, riceverà un riepilogo via email del numerico delle persone che hanno effettuato l'ingresso e le loro indicazioni basilari.
Queste informazioni sono fornite tramite query effettuate sul database ed inserite come oggetto delle email di riepilogo.
L'importanza dell'aggiornamento del dipendente che accoglie il pubblico sulle ultime notizie, è soddisfatta tramite l'implementazione di una "sezione notizie" nella Home Page, le quali consentiranno una vista globale delle ultime notizie reperibili dal web tramite una ricerca per argomenti di interesse.

2 Aspetti tecnologici

Gli aspetti tecnologici per questo progetto web si basano sull'utilizzo di API REST per offrire i vari servizi: autenticazione, news, salvataggio dati e query in MongoDB ed invio di email.

2.1 API REST

Le Application Program Interface operanti all'interno dell'architettura sono di tipo REST (REpresentation State Transfer).
Le API REST rispettano l'architettura client/server gestendo le richieste di risorse tramite HTTP e fornendo un'uniformità di interfaccia per i vari componenti, attori del servizio.
La rappresentazione delle risorse può essere facilmente manipolata ed utilizzata dal client in quanto fornita tramite JSON, formato altamente leggibile e molto diffuso.

2.2 Autenticazione

L'implementazione del servizio di autenticazione per l'applicazione, si è reso necessario per garantire l'opportuna sicurezza all'accesso dei dati e delle funzionalità.
Modulo fondamentale per il corretto funzionamento della funzionalità è PassportJS, middleware di autenticazione per Node.js.
Esso fornisce le c.d. "Strategies" ideate appositamente per servizi di autenticazione come Google, utilizzato in questa applicazione.
Essendo un progetto dedicato ed ideato per un utilizzo interno ad un'amministrazione si esige l'aderenza ai requisiti di sicurezza, quali confidenzialità, integrità e disponibilità dei dati (CIA triad).

2.2.1 OAuth2 e Google

L'autenticazione si basa sull'utilizzo di OAuth2.
OAuth2 è un protocollo di rete progettato per essere utilizzato con HTTP (Hyper Text Transfer Protocol) ed è considerato lo standard per l'autorizzazione all'accesso ed allo scambio sicuro di dati per servizi web come la Web Application, descritta in questo progetto.
Di seguito si riporta il codice delle pagine di authentication.

```
1  const express = require('express');
2  const nodemailer = require('nodemailer');
3  const app= express();
4  const session = require('express-session');
5  const passport = require('passport');
6  require('./public/js/auth');
7  function isLoggedIn(req, res, next) {
8    req.user ? next() : res.sendStatus(401);
9  }
10
```

```
app.use(session({ secret: 'cats', resave: false, saveUninitialized: true
    }));
app.use(passport.initialize());
app.use(passport.session());

app.get('/', (req, res) => {
  res.sendFile('autenticazione.html', { root: './public' })
});

app.get('/auth/google',
  passport.authenticate('google', { scope: [ 'email', 'profile' ] }
));

app.get( '/auth/google/callback',
  passport.authenticate( 'google', {
    successRedirect: 'http://localhost:5000/paginainiziale.html',
    failureRedirect: '/auth/google/failure'
  })
);

app.get('/protected', isLoggedIn, (req, res) => {
  res.send(`Buongiorno ${req.user.displayName}`);
});

app.get('/logout', (req, res) => {
  req.logout();
  req.session.destroy();
  res.send('Logout effettuato');
});

app.get('/auth/google/failure', (req, res) => {
  res.send('Autenticazione fallita');
});

app.listen(5000, () => console.log('in ascolto su porta 5000'));
```

Si riporta successivamente il codice di "auth.js"
Si fa presente che il .[dot]env richiamato nella prima riga del codice, è un file di testo di configurazione contenente variabili da passare al sistema.
In questo caso contiene il Client ID e il Client Secret fornito dal servizio di Google Credentials per abilitare la autenticazione.

```
require('dotenv').config()

const passport = require('passport');
const GoogleStrategy = require('passport-google-oauth2').Strategy;

const GOOGLE_CLIENT_ID = process.env.GOOGLE_CLIENT_ID;
const GOOGLE_CLIENT_SECRET = process.env.GOOGLE_CLIENT_SECRET;

passport.use(new GoogleStrategy({
  clientID: process.env.GOOGLE_CLIENT_ID,
```

```
  clientSecret: process.env.GOOGLE_CLIENT_SECRET,
  callbackURL: "http://localhost:5000/auth/google/callback",
  passReqToCallback: true,
},
function(request, accessToken, refreshToken, profile, done) {
  return done(null, profile);
}));

passport.serializeUser(function(user, done) {
  done(null, user);
});

passport.deserializeUser(function(user, done) {
  done(null, user);
});
```

2.3 NewsApi

Per fornire al dipendente utilizzatore del servizio, una semplice gestione delle informazioni dalle maggiori testate giornalistiche del Paese, i dati vengono richiesti attraverso NewsApi. L'utente ha la possibilità di inserire in uno spazio di ricerca, il tema sul quale vuole ricevere un elenco di notizie.
Una volta effettuato l'invio, entra in gioco l'esecuzione dello script news.js, il quale legge l'input, lo utilizza per generare il link per NewsApi ed inoltra la richiesta.
La response viene fornita in formato JSON. A sua volta vengono creati i tag html necessari alla visualizzazione in pagina principale del risultato della ricerca..
In totale viene mostrato a video un elenco di 20 notizie per argomento ricercato, sotto forma di link.

```
const searchFrom= document.querySelector('.search');
const input = document.querySelector('.input');
const newsList = document.querySelector('.news-list');

searchFrom.addEventListener('submit', retrieve)

function retrieve(e){
    if(input.value==''){                              //effettuo un
       controllo sull'input, nel caso di invio senza valore viene mostrato
        in output un alert
        alert ('Attenzione: input mancante');
        return
    }

    newsList.innerHTML=''; //Consente di effettuare un'altra richiesta
       dopo aver eliminato l'input precedentemente inserito
    e.preventDefault()

  const apiKey='991201da19e8452f8118fe37ac2f26e3';
  let topic= input.value;
```

```
18    let url=`https://newsapi.org/v2/everything?q=${topic}&apiKey=${apiKey}&
          language=it`
19  /*Utilizzo l'url dato dal fornitore della API (NewsApi).
20  Nell'url viene inserito l'input relativo al topic richiesto dall'utente e
       l'apiKey memorizzata come costante */
21    fetch(url).then((res)=>{
22        return res.json()
23    }).then((data)=>{
24        console.log(data)
25      data.articles.forEach(article =>{
26          let li= document.createElement('li');            //il tag li serve
               a definire gli indici di un elenco
27           li.className="list-group-item list-group-item-dark";
28          let a = document.createElement('a');             //il tag a
               identifica i collegamenti con le pagine delle notizie
29          a.className="link-dark";
30          a.setAttribute('href', article.url);
31          a.setAttribute('target','_blank');           //nuovo tab al click
               del link della notizia
32          a.textContent = article.title;
33          li.appendChild(a);
34        newsList.appendChild(li)
35      })
36
37    }).catch((error)=>{
38        console.log(error)
39    })
40  }
```

2.4 Database NoSQL MongoDB

La funzione principale della web app, è devoluta all'inserimento di dati tramite form, ed inviati tramite connessione con il database MongoDB e la libreria Mongoose.

2.4.1 Struttura del database

Il Database individuato per questa applicazione web è MongoDB.
È un database NoSQL (Not Only SQL), caratterizzato dall'essere non relazionale, distribuito, in quanto consente di distribuire su più nodi lo storage ed infine scalabile. MongoDB è un document database che sfrutta il formato JSON per le query sui dati memorizzati in esso. La connessione avviene tramite il tool di modellazione degli oggetti "Moongoose" sulla porta 27017, standard per il database e precedentemente definita:

```
1  mongoose.connect('mongodb://localhost:27017/mydb');
```

In questo progetto il “document”, in formato JSON, ha una forma del tipo:

```
{
    "_id": {
        "$oid": "abc123"
    },
    "nome": "Francesco",
    "cognome": "Avantaggiato",
    "phone": "123456789",
    "data": "2022-01-01",
    "ora": "23:20",
    "motivazione": "Cambio CI"
  }
```

Si riporta di seguito la parte di codice estratta da “server.js”.
In questi passaggi viene anzitutto inizializzata la connessione al database NoSQL, ovvero senza modello relazionale, successivamente viene definita la struttura dello schema.
In caso di corretto inserimento dei dati, si reindirizza l’utente ad una nuova schermata, riportante la buona riuscita dell’operazione, nella quale si richiede la successiva attività da intraprendere.

```
//salvataggio dati ingresso in MongoDB
var bodyParser=require("body-parser");
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/mydb');                    //
    richiamo url di connessione con db
var db=mongoose.connection;
db.on('error', console.log.bind(console, "connection error"));
db.once('open', function(callback){
  console.log("connessione ok");                                      //se la
      connessione va a buon fine
})

app.use(bodyParser.json());
app.use(express.static('public'));
app.use(bodyParser.urlencoded({
  extended: true
}));

app.post('/ingresso.html', function(req,res){
  var name = req.body.name;
  var cognome =req.body.cognome;
  var phone =req.body.phone;
    var date =req.body.date;
    var ora =req.body.ora;
    var motivazione = req.body.motivazione

  var data = {                                          //definisco la strutture
      dello schema
    "name": name,
    "cognome":cognome,
    "phone":phone,
    "date": date,
```

```
    "ora":ora,
    "motivazione": motivazione
  }
db.collection('details').insertOne(data,function(err, collection){
                //se inserimento andato a buon fine
    if (err) throw err;
    console.log("Inserimento dati avvenuto con successo");

  });
  return res.redirect('inserito.html');

})
```

2.4.2 Query

Le query sottoposte al database vengono gestite attraverso la composizione di vari valori di richiesta, utilizzando variabili e funzioni di JavaScript.
Di seguito si riporta il codice relativo alle query, con output in console.

```
// query mongodb per mostrare in output gli ingressi odierni
var MongoClient = require('mongodb').MongoClient;
const { WriteConcern } = require('mongoose/node_modules/mongodb');
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  const d = new Date();
  d.setDate(d.getDate());
  var dbo = db.db("mydb");
  var query = { date: '2022'+'-'+(d.getMonth()+1).toString().slice(-2)+'-'
     +'0'+ d.getDate().toString().slice(-2)};
  //getMonth mi restituisce i valori dei mesi da 0 a 11, dunque necessito
     l'aggiunta di 1 per visualizzare la data normalmente
  //slice consente di gestire i giorni con 01...09, in quanto nel db gli
     stessi sono memorizzati con lo 0

  dbo.collection("details").find(query).toArray(function(err, result) {
    if (err) throw err;
    console.log('Gli ingressi di oggi:')
    console.log(result)

    dbo.collection("details").count(function(err, res){           //
       restituisce il totale degli elementi nel db
      if (err) throw err;
      console.log('Il numero di ingressi totali    '+res);
```

2.5 Servizio Email - NodeMailer, NodeCron, Mailtrap

Il servizio di invio email si suddivide in due diverse implementazioni:

1. invio di email tramite il form dell'Help Desk;
2. invio di email automatico con resoconto giornaliero per l'amministratore.

2.5.1 Gestione dell'Help Desk

Il form dell'Help Desk consente all'utente di inviare una mail con una richiesta di assistenza o altre tipologie di richieste, tramite un campo "messaggio" libero.
Il funzionamento si basa sull'utilizzo di NodeMailer, un modulo NodeJS che consente il settaggio e l'invio di email, il framework Express, progettato per la creazione di Web Application e di API.
Infine un servizio di email sandbox, che consente la creazione di server SMTP fake al solo scopo di development, chiamato MailTrap.
Lato server (server.js), viene definita la connessione al servizio tramite porta 2525, l'autenticazione a MailTrap, tramite credenziali fornite dallo stesso e le opzioni email, ovvero la struttura del corpo del messaggio da inviare.
Si implementano anche vari console.log e alert per determinare il successo o meno delle connessioni e/o invii di email.
Lo script "app.js" consente di prelevare i dati forniti in input ed inviarli tramite AJAX con l'oggetto XMLHttpRequest.
XMLHTTP è un set di API che possono essere usate da JavaScript per trasferire XML o altri dati da e verso un web server tramite HTTP.
In caso di invio andato a buon fine, vengono ripuliti i campi relativi del form, consentendo eventualmente di effettuare un nuovo inserimento.
Si riporta di seguito il codice relativo a quanto descritto.
Di seguito il codice.

```
1  const PORT =process.env.PORT || 2525;
2
3  app.use(express.static('public'));
4  app.use(express.json())
5
6  app.get('/', (req,res)=>{
7      res.sendFile(__dirname + '/public/paginainiziale.html')
8  })
9
10 app.post('/', (req, res)=>{
11     console.log(req.body);
12     const transporter = nodemailer.createTransport({        //
           autenticazione al servizio mail
13         host: "smtp.mailtrap.io",
14         port: 2525,
15         auth: {
16             user: "949b0d30a196ba",
```

```
            pass: "66fb874925b2a6"
        }
      });
      //opzioni email da inviare
      const mailOptions = {
        to: "francescoavant2000@gmail.com",
        from: req.body.email,       //nelle opzioni della mail richiamo
            ci   che l'utente ha inserito in input

        subject: `Messaggio da ${req.body.email}: ${req.body.subject}`, //
           nell'oggetto della mail inserisco i dati in input forniti dall'
           utente
        text: req.body.message
    }

    //invio della email
    transporter.sendMail(mailOptions, (error, info)=>{
        if (error){
            console.log(error);
            res.send('error');
        }else{
            console.log('Email inviata: '+ info.response);          //
               stampo nel log l'invio andato a buon fine con il response
            res.send('successo');
        }
    });
});
```

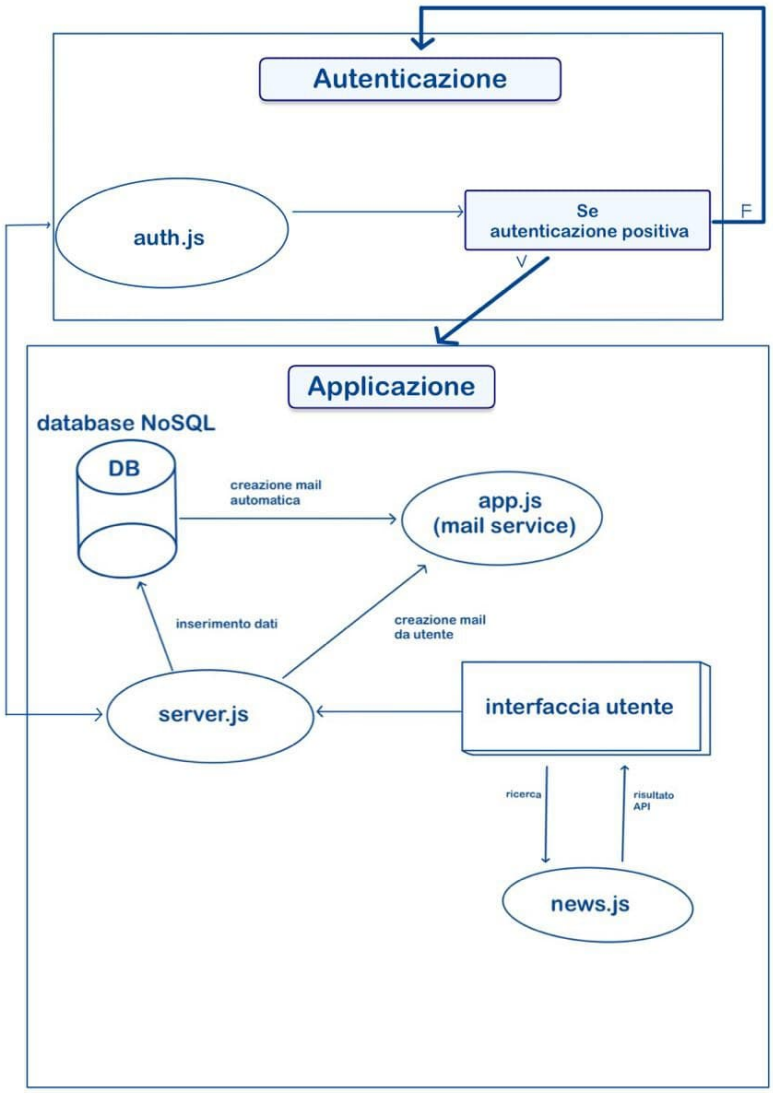
2.5.2 Automatismo dell'invio

L'invio automatico di mail all'amministratore di sistema basa il suo funzionamento sulle query al database e sull'utilizzo, fondamentale, del modulo Nodecron, il quale consente di settare orari predefiniti e scadenze per l'attuazione di determinate azioni. In questo caso l'azione è l'invio della mail di resoconto giornaliero, che viene inoltrata nell'orario corrispondente alla chiusura dell'ufficio.
Questa impostazione risulta essere a disposizione delle necessità implementative in "real life".

```
const contactForm = document.querySelector('.contact-form');


let name= document.getElementById('name');
let email= document.getElementById('email');
let subject= document.getElementById('subject');
let message= document.getElementById('message');

contactForm.addEventListener('submit', (e)=>{
    e.preventDefault();

    let formData= {
```

```
13          name: name.value,
14          email: email.value,
15          subject: subject.value,
16          message: message.value
17      }
18
19      let xhr= new XMLHttpRequest();
20      xhr.open('POST', '/');
21      xhr.setRequestHeader('content-type', 'application/json');
22      xhr.onload = function(){
23          console.log(xhr.responseText);
24          if(xhr.responseText == 'successo'){                    //dopo
                invio della mail i campi del form vengono ripuliti
25              alert ('Email inviata');                           //in caso
                    di invio andato a buon fine, viene mostrato un alert all'
                    utente
26              name.value='';
27              email.value='';
28              subject.value='';
29              message.value='';
30          }else{
31              alert('qualcosa non ha funzionato');
32          }
33      }
34      xhr.send(JSON.stringify(formData));
35  });
```

3 Schema di funzionamento dell'applicazione



4 Conclusioni

L'architettura della web application messa a disposizione per una variegata tipologia di clientela, si offre all'immediatezza, semplicità e scalabilità.
La scelta implementativa dei servizi descritti, si è basata sulla attuale modernità dei frameworks e sull'aderenza del progetto al programma di studio relativo all'insegnamento di Software Oriented Architecture, soprattutto riguardo le API REST.
Nella progettazione di questo sito web, si è posta l'attenzione sulle principali problematicità, facendo particolare riferimento all'interazione con l'utente finale.
L'importanza dell'autenticazione rappresenta il core principale del servizio, senza la quale esso si presenta come privo di fondamenti di sicurezza per un'architettura da mettere in funzione in tempi moderni.
L'utilizzo di query specifiche sul DB, per avere come risultato dati singoli e composti su ciò che è stato fornito in input, è risultato essere determinante per l'implementazione di alcune funzioni successive come quella dell'invio automatico di email.
Si è posta attenzione alla figura dell'amministratore in quanto utente principale,identificato come referente nel caso di ricorso all'Help Desk ed individuo in grado di ricevere aggiornamenti costanti sulla situazione generale del servizio, con resoconti inviati tramite testo di una email.

5 Note Sitografiche

1. W3school https://www.w3schools.com/
2. OAuth - documentation https://oauth.net/
3. PassPortJS - documentation https://www.passportjs.org/docs/
4. Google APIs - credentials https://console.cloud.google.com/apis/credentials
5. MongoDB - documentation https://docs.mongodb.com/
6. MailTrap Guide https://mailtrap.io/
7. Nodemailer - documentation https://nodemailer.com