Introduzione

Il progetto denominato “Logview” vuole essere una semplice applicazione web per visualizzare i log con una comoda funzione di ricerca. Permette di scegliere il tipo di log e il contenuto delle entità da visualizzare.
Il progetto è stato reso pubblico sulla nota piattaforma GitHub con il seguente link: https://github.com/XelK/logview/ , inoltre l'applicazione è stata deployata sull'hosting heroku ed è raggiungibile con questo url: https://l0gviewer.herokuapp.com/

Analisi dei requisiti

Destinatari

La tipologia di log utilizzati che è stata scelta è quella di log **Apache**, in quanto ben documentata e molto presente nel mercato aziendale. La documentazione del formato utilizzato è presente in questo url: https://httpd.apache.org/docs/2.4/logs.html , mentre i file di log utilizzati per il test sono stati generati con l'applicazione opensource "flog" reperibile da questo url: https://github.com/mingrammer/flog .
Utente tipo per questo tipo di applicazione è un utente esperto che sà cosa cerca ed utilizza il device con uno schermo grande come computer. Inoltre è stata resa disponibile l'opzione per interrogare il log richiesto senza dover accedere alla pagina web tramite browser ma effettuando delle chiamate API.

Modello

L’obiettivo è quello di fornire un supporto ad uno tecnico o ad un team di nici e vuole consultare con semplicità porzioni di log . È un valore aggiunto poiché permette una rapida estrapolazione dei log di interesse senza dover collegarsi al server web.

Aspetti tecnologici

Aspetti tecnologici
Per la realizzazione di Logviewer sono state sfruttate diverse tecnologie che andrò a riepilogare di seguito:

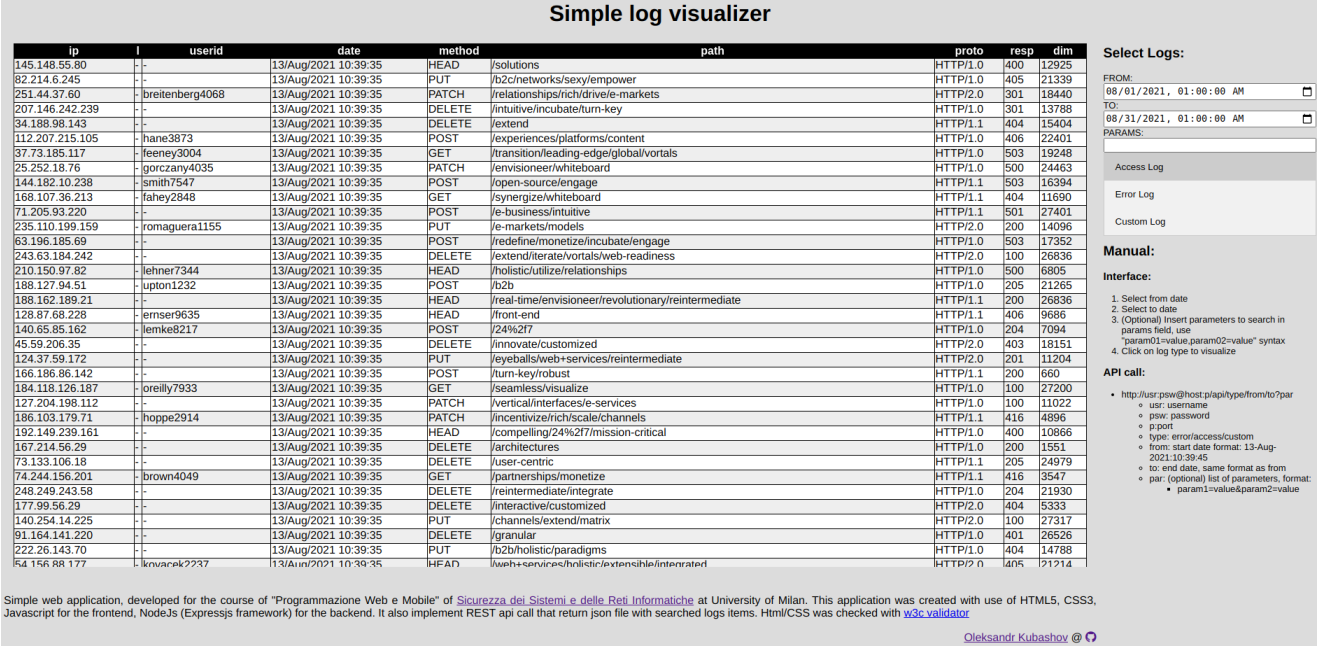
- NodeJS ovvero un runtime JavaScript costruito sul motore di JavaScript V8 di Chrome, sfruttato come backend;

- Express.js un framework web veloce, non categorico e minimalista per NodeJS;
- HTML5, CSS3 e JS sono stati utilizzati per la parte di frontend, si è scelto di non caricare il frontend con dei framework in quanto si è cercato di mantenere l'interfaccia più leggera, semplice e minimale.
  L'accesso inoltre è stato protetto con username e password.

Interfacce

L'applicazione è una classica single-page application con i seguenti parametri:

- scelta dell'intervallo di interesse
- scelta del tipo di log da visualizzare tra i log presenti (access/errror/custom)
- opzionale parametri da ricercare nel log selezionato in precedenza.



Inoltre è possibile ottenere gli stessi risultati effettuando la chiamata get fornendo i campi di interesse. L'url di richiesta deve essere formata in questo modo:
`https://usr:psw@l0gviewer.herokuapp.com/api/type/from/to?params`

dove:

- usr/psw: inidcano username e password necessarie per autenticarsi
- l0gviewer.herokuapp.com: url dove è stata deployata l'applicazione
- api: url della pagina costruita con Express
- type: tipo di log da visualizzare (access/error/custom)
- from/to: intervallo di date in formato: `1-Aug-2021:11:40:52`
- params: lista di parametri opzionale, nel formato:
  `parametro1=valore&parametro2=valore`

Architettura

L'applicazione si compone di 3 grandi blocchi.

1. Pagina principale visibile visitando l'applicazione con browser. Questa ha i suddetti compiti:
    1. interagire con l'utente registrando i dati selezionati;
    2. effettuare una chiamata verso la pagina di api come se si effettuasse una chiamata get descritta in precedenza
    3. e dopo aver ricevuto il file json con il risultato trasformare questo in una tabella da visualizzare all'utente.
    Il codice di maggior interesse in questa parte è realizzato in javascript di seguito forniamo la funzione di interesse, che si occupa di trasformare i parametri selezionati dall'utente in formato accettato dalle api realizzati.

```
/**
 * Function used into html page.
 * Read parameters insereted into html page. Convert data parameters,
check the correctness
 * of parameters and execute the HttpRequest to the api page
 * @param  button [selected button on the page]
 * @param  logType [type of log selected, can be: access/error/custom]
 */
function selectLog(button, logType) {
    var i, tablinks;

      let from=document.getElementById("from").value;
      let to=document.getElementById("to").value;
      let params=document.getElementById("params").value;

      const month=
["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"];


      // convert to date format requested by api
      const reg=/(\d+)-(\d+)-(\d+)T(\d+):(\d+):(\d+)/;
      let t=from.match(reg);
      from=t[3]+"-"+month[t[2]%10-1]+"-
"+t[1]+":"+t[4]+":"+t[5]+":"+t[6];
      t=to.match(reg);
```

```
      to=t[3]+"-"+month[t[2]%10-1]+"-"+t[1]+":"+t[4]+":"+t[5]+":"+t[6];

    tablinks = document.getElementsByClassName("tablinks");
    for (i = 0; i < tablinks.length; i++) {
      tablinks[i].className = tablinks[i].className.replace(" active",
"");
    }
    button.currentTarget.className += " active";

    // Create an XMLHttpRequest object
      const xhttp = new XMLHttpRequest();

      // Define a callback function
      xhttp.onreadystatechange=function(){
        if (xhttp.readyState === 4) {
          if (xhttp.status === 200) {
            createTable(this.responseText);
          } else {
             alert("Error code: "+xhttp.status+"\nPlease insert correct
parameters!");
          }
      }
    }
      // Send a request
      if (params.length>0)

        var
request="api/"+logType+"/"+from+"/"+to+"/?"+params.replace(/\,/g,"&");
      else
        var request="api/"+logType+"/"+from+"/"+to;
      xhttp.open("GET",request, true);
      xhttp.send();

  }
```

2. Pagina api, questa si posiziona in mezzo tra l'interfaccia web e il file che trasforma il file log in formato json. La suddetta pagina è accessibile anche via chiamata **GET**. Svolge inoltre il compito di verificare la correttezza dei dati passati ed eventualmente restituire la pagina d'errore:

```
router.get('/:type/:from/:to', function (req, res,next) {

    /**
     * Object that contain parameters passed via REST call.
     * @param  type [type of log to read, can be access/error/custom]
     * @param  from [start date]
     * @param  to [end date]
     * @param  params [optional list of parameters]
     */
    const request={
        // mandatory parameters:
        type: req.params.type,
        from:req.params.from,
        to:req.params.to,
        // optional params:
        params:req.query
    }

        let resp=getData(request);
        let js = createJson(resp,request.type);


        // Errors handling
        // from/to inserted not correctly
        if(dataRangeError(request.from,request.to))
            return next(createError(400)); // 400: bad request
        // wrong parameters format inserted
        if(Object.values(request.params).includes(''))
            return next(createError(400)); // 400: bad request


        resp=[];
        js.forEach(element => {
            if(dateBetween(request.from,request.to,element.date)){
                if(find(element,request.params))
                    resp.push(element);
            }
        });
        res.json(resp);
});
```

Funzioni di controllo che sono stati realizzati nel codice:

```
/**
 * Function that search if param passed via REST is present in the js
object
 * @param  item [item from file log]
 * @param  params [array of params passed via REST call]
 */
function find(item,params){
    var ret=true;
    Object.keys(params).forEach(pKey => {
        if( (typeof params[pKey] != "undefined") &&
(!item[pKey].includes(params[pKey])) )
            ret=false;
    });
    return ret;
}

/**
 * Function that check the correctness of data range
 * @param  start [start date]
 * @param  end [end date]
 */
function dataRangeError(start,end){
    let s=start.replace(":"," ");
    let e=end.replace(":"," ");

    s=Date.parse(s);
    e=Date.parse(e);
    if(e<s)  // end date is before start date
        return true;
    return false;
}
/**
 * Function that check if date from log is between start/end date
passed by user
 * @param  start [start date]
 * @param  end [end date]
 * @param  data [date from file]
 */
```

```
function dateBetween(start,end,data){
    let s=start.replace(":"," ");
    let e=end.replace(":"," ");
    let d=data.day+"/"+data.month+"/"+data.year+"
"+data.hour+":"+data.minutes+":"+data.seconds;

    s=Date.parse(s);
    e=Date.parse(e);
    d=Date.parse(d);

    return( (s<=d) && (d<=e));

}
```

3. Infine è presente un terzo file javascript `logfiles.js` che si preoccupa di leggere il file log corretto e trasformare il contenuto in formato json. Questo è stato realizzato come una libreria esterna. Di seguito il codice per il log di tipo `access`:

```
/**
 * Function that trasform apache ACCESS log file into json
 * @param  data [file content]
 * @returns json data with access log
 */
exports.jsonAccess = function (data){

    /* apache common log format: LogFormat "%h %l %u %t \"%r\" %>s %b"
common
    * see: https://httpd.apache.org/docs/2.4/logs.html
    * example:
    * 127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET
/apache_pb.gif HTTP/1.0" 200 2326
    */

    var temp=data.split("\n");
    var resp=[];
    var i=0;
    const regAccess=/(\d+\.\d+\.\d+\.\d+) (\S+) (\S+) \
[(\d+)\/(\w+)\/(\d+)\:([\d]+)\:(\d+)\:(\d+)(?:.*)\] "(\w+) (\S+)
(\S+)\" (\d+) (\d+)/;
```

```
    temp.forEach((line)=>{
        const w=line.match(regAccess);
        if(w !== null){
            resp.push({});
            resp[i].ip=w[1];
            resp[i].l=w[2];
            resp[i].userid=w[3];
            resp[i].date={};
                resp[i].date.year=w[6];
                resp[i].date.month=w[5];
                resp[i].date.day=w[4];
                resp[i].date.hour=w[7];
                resp[i].date.minutes=w[8];
                resp[i].date.seconds=w[9];
            resp[i].method=w[10];
            resp[i].path=w[11];
            resp[i].proto=w[12];
            resp[i].resp=w[13];
            resp[i].dim=w[14];
            i++;
        }
    });
    return resp;
}
```

Codice

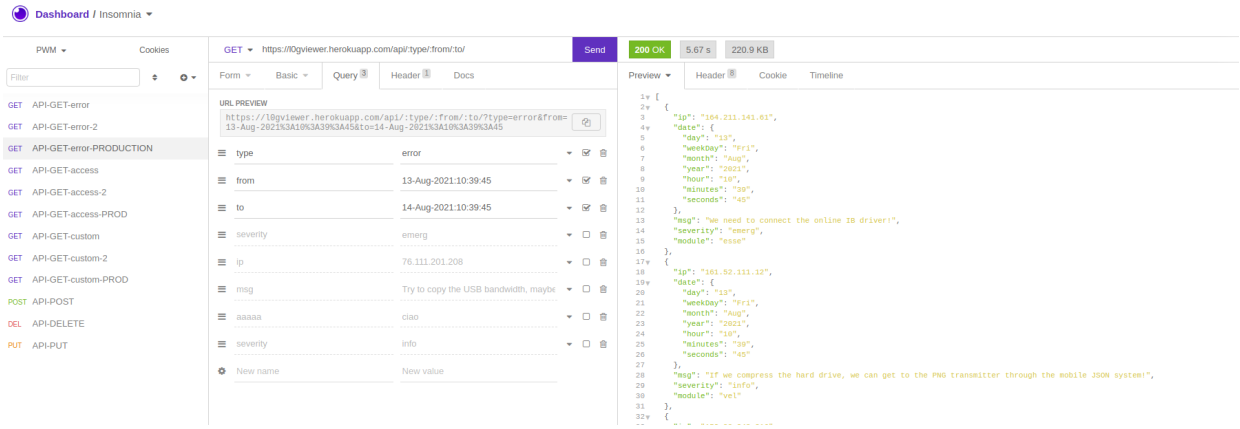
Utilizzo di **nodejs** e specialmente framework **express** ha facilitato la realizzazione dell'applicazione in questione. In particolare è stato possibile utilizzare **javascript** sia per il frontend che per il backend. Inoltre utilizzo del framework ha semplificato la realizzazione dell'autenticazione e in particolare la realizzazione del routing con la costruzione di una url dinamica per la chiamate API, che, altrimenti, sarebbe stata una delle parti più ardue da realizzare. Il codice html/css inoltre è stato testato con **w3cvaidator** https://validator.w3.org/ superando correttamente il test.
Considero un grande vantaggio aver adottato il **modello MVC** separando la parte di frontend da quella di backend. E quindi realizzato e reso disponibile delle API in modo sia di garantire una fruizione di questa via chiamate GET, senza dover aprire necessariamente il browser o anche utilizzando ulteriori tool automatici o a

riga di commando per una futura elaborazione di dati.
Inoltre il fatto di separare la parte frontend da quella backend permette un eventuale futuro sviluppo da parte di 2 team di sviluppatori o anche una eventuale futura sostituzione di una delle due parti.

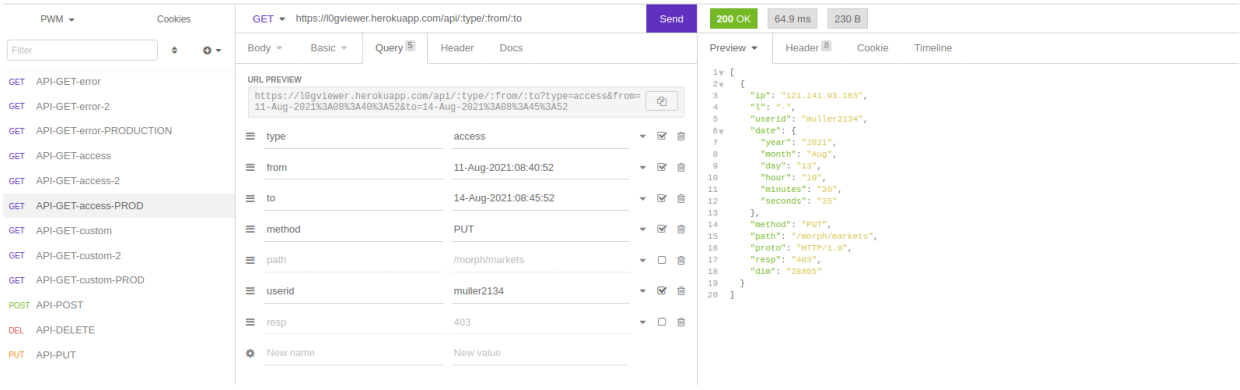
Strumenti

L'applicazione è stata realizzata utilizzato **vscode** per la scrittura del codice in quanto si integra perfettamente con javascript/html/css. Inoltre è stato utilizzato **git** con **github** per il versioning del codice. Durante lo sviluppo inoltre sono stati utilizzati **nodemon** tool che aiuta lo sviluppo con node.js in quanto esegue in automatico il riavvio dell'applicazione quando si effettua la modifica del codice. Per il test delle API invece è stata utilizzata l'pplicazione **insomnia** per la semplicità di configurazione, di seguito qualche screenshot di questa:

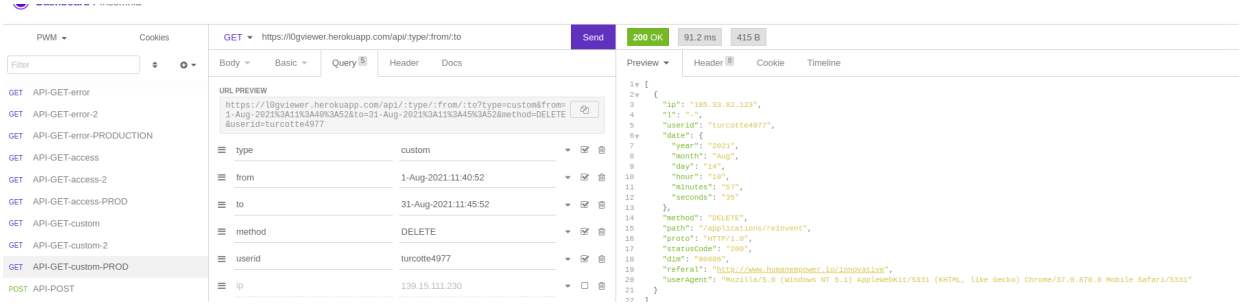
- Error log:



- Access log:



- Custom log:



Nel codice inoltre sono presenti numerosi **espressioni regolari**, per semplicità di realizzazione è stato utilizzato lo strumento disponibile a questo url: https://regex101.com/

Per quanto riguardo il deploy è stato scelto **heroku** in quanto presenta un piano gratuito e si integra perfettamente con github e nodejs.

Conclusione

In conclusione posso dire che il progetto è stato interessante, mi ha permesso di interfacciarmi con il nuovo linguaggio NodeJs e framework Express. Ho trovato di notevole interesse il funzionamento di NodeJs e la semplicità del linguaggio permettendomi di concentrarmi sulla parte backend, dove ho potuto divertirmi con l'elaborazione dei dati di log, e realizzazione di un semplce **API REST** anche se implementando solo il comando **GET**. In aggiunta ho potuto esercitarmi con delle espressioni regolari, strumento indispensabile per qualsiasi sistemista.