Documentazione progetto Test 22HBG Backend API

Andrea Munari

November 27, 2023

1 Introduzione

Questa documentazione fornisce dettagli approfonditi sull'API del backend del Progetto Test 22HBG. L'API è sviluppata in JavaScript utilizzando Node.js, Express, MySQL, Nodemon, Redis e Swagger.

2 Configurazione dell'Ambiente di Sviluppo

Per iniziare, segui i passaggi seguenti:

1. Installa le dipendenze con il comando: `npm install`.
2. Configura le variabili d'ambiente nel file `.env` o utilizza i valori di default.
3. Avvia l'applicazione con il comando: `npm start`.

3 File database.js

Il file `database.js` contiene la configurazione necessaria per connettersi al database MySQL offerto da freesqldatabase.com.
I parametri di connessione includono:

- `host`: Indirizzo del server MySQL.
- `database`: Nome del database.
- `user`: Nome utente per la connessione al database.
- `password`: Password per la connessione al database.
- `port`: Porta di connessione al server MySQL.

Il file gestisce la connessione al database e fornisce l'oggetto di connessione da utilizzare nelle altre parti dell'applicazione.

4 Documentazione API

La documentazione API è generata automaticamente utilizzando Swagger e può essere esplorata interattivamente visitando progettotest22hbg.onrender.com/api-docs.

5 Endpoints

5.1 Ottenere la lista di tutti i post

Endpoint: `GET /posts`
Questo endpoint restituisce la lista di tutti i post dal feed.

```
app.get('/posts', (req, res) => {
    fetch(URL)
    .then(response => {
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status
                }`);
        }
        return response.json();
    })
    .then(data => {
        const postList = [];
        k = data;
        postList.push(k);
        console.log('my data ', postList);
        res.json(postList);
    })
    .catch(error => {
        console.error('Fetch error:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    });
});
```

Listing 1: Implementazione dell'endpoint per ottenere la lista di tutti i post

Quando una richiesta `GET` viene effettuata su `/posts`, l'applicazione esegue una richiesta al feed tramite la funzione `fetch`. Se la chiamata ha successo (`200 OK`), restituisce un file JSON contenente tutti i post dal feed. In caso di errore durante la richiesta al feed, viene restituito uno stato di errore interno del server.

5.2 Ottenere i post filtrati

Endpoint: `GET /posts-filtered`
Questo endpoint restituisce una lista di post filtrati in base ai parametri:

- `title`: filtra solo i post contenenti la stringa specificata
- `items`: numero dei post da restituire

Questo endpoint è l'unico ad avere una gestione di cache per migliorare le prestazioni. Per farlo utilizza la funzione `cache` come middleware per Redis.

```
app.get('/posts-filtered', cache, (req, res, next) => {
    fetch(URL)
    .then(response => {
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status
                }`);
        }
        return response.json();
    })
    .then(data => {
        console.log("Fetching data...")
        const postList = [];
        const filteredData = data.filter(element =>
            element.title.rendered.includes(req.query.title)
        );

        for (let i = 0; i < req.query.items && i < filteredData.
            length; i++) {
            postList.push(filteredData[i]);
        }

        console.log('my data ', postList); // This gives me the
            filtered posts

        //create a key to cache this result
        redis_key = req.query.title+req.query.items;
        console.log("THE REDIS KEY IS:"+redis_key);
        //memorize in cache for one hour
        client.setEx(redis_key, 3600, JSON.stringify(postList));

        res.json(postList);
    })
    .catch(error => {
        console.error('Fetch error:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    });
});
```

Listing 2: Implementazione dell'endpoint per ottenere i post filtrati

La funzione `cache` controlla se i risultati desiderati sono già presenti nella cache Redis. In caso affermativo, restituisce immediatamente i risultati cached senza effettuare una nuova richiesta al feed. In caso contrario, effettua il `fetch` e salva l'output in Redis usando come chiave la concatenazione dei parametri della richiesta.
Ad esempio, se la richiesta fosse `/posts-filtered?title=home&items=5`, allora la chiave per accedere all'output salvato su Redis sarà `home5`. Il codice JavaScript implementa l'endpoint, gestendo la richiesta al feed, la manipolazione dei dati, la cache e la risposta al client. In caso di errore durante la richiesta al feed, viene restituito uno stato di errore interno del server.

5.3 Sincronizzare il database con i post dal feed

Endpoint: `GET /sync-db`
Questo endpoint legge i contenuti dal feed e li scrive nella tabella 'posts' nel database.

```
app.get('/sync-db', (req, res) => {
    fetch(URL)
    .then(response => {
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }
        return response.json();
    })
    .then(data => {
        const postList = [];
        k = data;
        postList.push(k);

        // extracting the info from the json file to put in the table
        console.log(postList);

        var query = "INSERT INTO posts ("+
            "ID, post_author, post_date, post_date_gmt, post_content, "+
            "post_title, post_excerpt, post_status, comment_status, "+
            "ping_status, post_modified, post_modified_gmt"+
            ") values ";
        let i;
        let values = '';
        for (i = 0; i < postList[0].length; i++) {
            let post = postList[0][i];

            let id = post.id;
            let post_author = post.author;
            let post_date = post.date;
            let post_date_gmt = post.date_gmt;
            let post_content = post.content.rendered; // remove the 'rendered' if it doesn't work
            let post_title = post.title.rendered; // same
            let post_excerpt = post.excerpt.rendered; // same
            let post_status = post.status
            let comment_status = post.comment_status;
            let ping_status = post.ping_status;
            let post_modified = post.modified;
            let post_modified_gmt = post.modified_gmt;

            if (i == postList[0].length-1) {
                values += '('+id+', '+
                            post_author+', '+ // int, it doesn't need quotes
                            `'${post_date}', ` + // enclose the date in single quotes
                            `'${post_date_gmt}', ` + // same
```

```
                                `${database.escape(post_content)}, ` +
                                    // Escape the post content
                                `${database.escape(post_title)}, ` + //
                                     same
                                `${database.escape(post_excerpt)}, ` +
                                    // same
                                `'${post_status}', ` +
                                `'${comment_status}', ` +
                                `'${ping_status}', ` +
                                `'${post_modified}', ` +
                                `'${post_modified_gmt}')`;
                    console.log("\n"+values+"\n");
                } else {
                    values += '('+id+', '+
                                post_author+', '+ // int, it doesn't
                                    need quotes
                                `'${post_date}', ` + // enclose the
                                    date in single quotes
                                `'${post_date_gmt}', ` + // same
                                `${database.escape(post_content)}, ` +
                                    // Escape the post content
                                `${database.escape(post_title)}, ` + //
                                     same
                                `${database.escape(post_excerpt)}, ` +
                                    // same
                                `'${post_status}', ` +
                                `'${comment_status}', ` +
                                `'${ping_status}', ` +
                                `'${post_modified}', ` +
                                `'${post_modified_gmt}'),`;
                    console.log("\n"+values+"\n");
                }
            }
            query+=values;
            console.log(query);

            // res.json(postList);
            let resText = "Requested adding i="+i+" rows to the
                database table 'posts'.<br><br>";
            database.query(query, (error, data)=>{
                if (error) {
                    res.send(resText+"Request had some errors:<br>"+
                        error)
                } else {
                    res.send(resText+"Request completed");
                }
            })
        })
        .catch(error => {
            console.error('Fetch error:', error);
            res.status(500).json({ error: 'Internal Server Error' });
        });
});
```

Listing 3: Implementazione dell'endpoint per sincronizzare il database con i post dal feed

Questo endpoint legge i contenuti dal feed attraverso una richiesta HTTP,

e successivamente inserisce i dati nella tabella 'posts' nel database MySQL. La funzione esegue un'unica query SQL di tipo INSERT per tutti i post ottenuti dal feed e restituisce una risposta che informa l'utente del successo o del fallimento della query. La scelta di effettuare un'unica INSERT per tutti i post individuati nasce dalla volontà di minimizzare il numero di interazioni con il database dato che il servizio di hosting utilizzato è piuttosto lento nel piano gratuito.

5.4 Ottenere i post dal database

Endpoint: `GET /posts-db`
Questo endpoint legge e restituisce i contenuti della tabella 'posts' nel database in formato JSON.

```
app.get('/posts-db', (req, res)=> {
    var query = "SELECT * FROM posts";

    database.query(query, (error, data)=>{
        if(error) {
            throw error;
        }
        const ris = JSON.stringify(data, null, 2);
        console.log(ris);
        res.header('Content-Type', 'application/json');
        res.send(ris);
    });
});
```

Listing 4: Implementazione dell'endpoint per ottenere i contenuti della tabella 'posts' dal database

Quando una richiesta `GET` viene effettuata su `/posts-db`, l'applicazione esegue una query SQL di tipo `SELECT` per ottenere tutti i dati dalla tabella 'posts' e restituisce il risultato in formato JSON. Nel caso in cui la tabella fosse vuota, viene restituito un file JSON vuoto, senza causare errori. In caso di errori imprevisti durante l'esecuzione della query, viene generata un'eccezione.

5.5 Eliminare tutti i post dal database

Endpoint: `GET /posts-db-del`
Questo endpoint elimina tutti i record dalla tabella 'posts' nel database. Questo endpoint non fa parte dei requisiti, è stato scelto di aggiungerlo solo per favorire maggiore praticità nella verifica del comportamento dell'API.

```
app.get('/posts-db-del', (req, res)=> {
    var query = "DELETE FROM posts";

    let resText = "Request to delete all rows in 'posts' table.<br>
        ";
    database.query(query, (error, data)=>{
        if(error) {
            res.send(resText+"Request had some errors:<br>"+error);
        } else {
            res.send(resText+"Request completed successfully");
```

```
10          }
11      });
12  });
```

Listing 5: Implementazione dell'endpoint per eliminare tutte le righe dalla tabella 'posts' del database

Quando una richiesta `GET` viene effettuata su `/posts-db-del`, l'applicazione esegue una query SQL di tipo `DELETE` per eliminare tutte le righe dalla tabella 'posts' e restituisce una risposta che informa l'utente sul successo o il fallimento della query SQL.

6 Cache

L'API utilizza la cache Redis per migliorare le prestazioni. Se un risultato è già presente nella cache, verrà restituito immediatamente senza effettuare una nuova richiesta al feed.

7 Ambiente di Sviluppo

Per lo sviluppo, puoi utilizzare Nodemon per il riavvio automatico dell'applicazione in caso di modifiche al codice. Quando si avvia l'application con `npm start`, nodemon viene eseguito automaticamente.