Buildy

Applicazioni e Servizi Web

Filippo Cavallari - 0001025884 {filippo.cavallari2@studio.unibo.it}

28/05/2022

0.1 Introduzione

Buildy è una semplice piattaforma di continuos deployment, facile ad usare ma allo stesso tempo molto potente, per *"buildare"* e *"deployare"* progetti Java basati su Gradle.

0.2 Requisiti

0.2.1 Requisiti utente

L'utente finale di Buildy è un Java Software Engineer o un Java Software Developer che si aspetta di poter:

- vedere la lista di tutti i progetti registrati con relativo status dell'ultima compilazione
- poter aggiungere un nuovo progetto
- poter consultare tutte le compilazioni di un progetto, sia attuali che passate
- poter accedere ai log di una specifica compilazione
- poter accedere agli artefatti (.jar) di una specifica compilazione
- poter ottenere un link ad un badge che si aggiorna in automatico in base allo status dell'ultima compilazione

0.2.2 Requisiti funzionali

- Autenticazione di un utente
- Registrazione di un nuovo utente
- Aggiunta di un progetto
- Monitoraggio costante dei progetti
- Compilazione dei progetti
- Pubblicazione dei log delle compilazioni
- Pubblicazione degli artefatti delle compilazioni
- Creazione dei badge .SVG per ogni compilazione
- Cronologia delle compilazioni passate di ogni progetto

0.2.3 Requisiti non funzionali

- Rendere il sistema reattivo (e.g. caricamenti minimi o assenti)
- Interfaccia utente adatta anche a persone con problemi di visibilità (e.g. daltonismo e cecità)
- Rendere il processo di autenticazione sicuro
- Rendere il sistema scalabile in base al carico di lavoro (i.e. compatibilità con docker swarm)

0.3 Design

Design dell'architettura del sistema e delle interfacce utente.

0.3.1 Interfacce utente

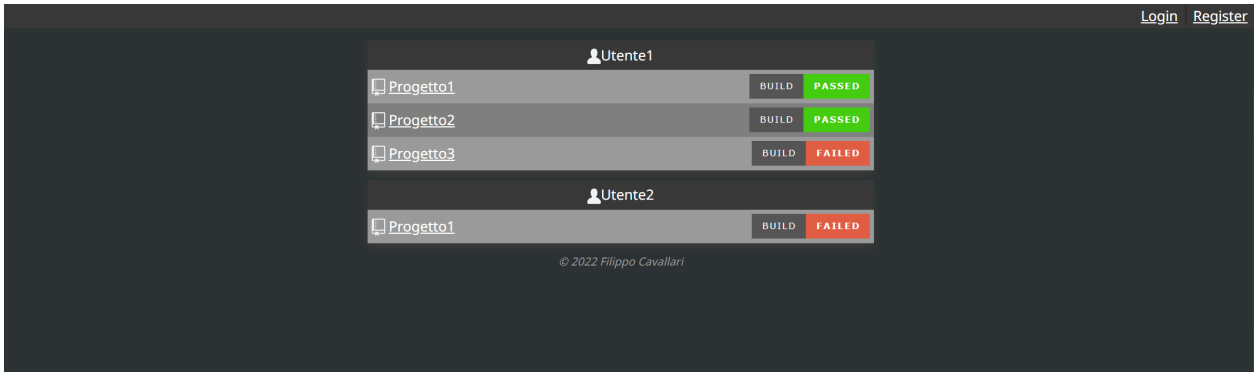

Figure 1: Home page

L'homepage di Buildy (figura 1) contiene la lista di tutti i progetti registrati, raggruppati in base al creatore della repository di ciascun progetto; nella navbar sono inoltre presenti due bottoni per aprire i pannelli per l'accesso e la registrazione.

Nel pannello di accesso (figura 2) vengono richiesti email e password.

Nel pannello di registrazione (figura 3) vengono chiesti l'email e due volte la password (per evitare errori di scrittura).

Dopo aver effettuato l'autenticazione, la home apparirà come in figura 4 ; la navbar mostrerà l'email con la quale si è effettuato l'accesso, un bottone per l'aggiunta di un nuovo progetto ed il bottone per effettuare il logout.

Quando si accede alla pagina di un progetto (figura 5 , si possono: consultare tutte le build passate nella sidebar a sinistra, con tanto di esito, ID e data di compilazione; accedere ai log o agli artefatti cliccando sui relativi link; copiare

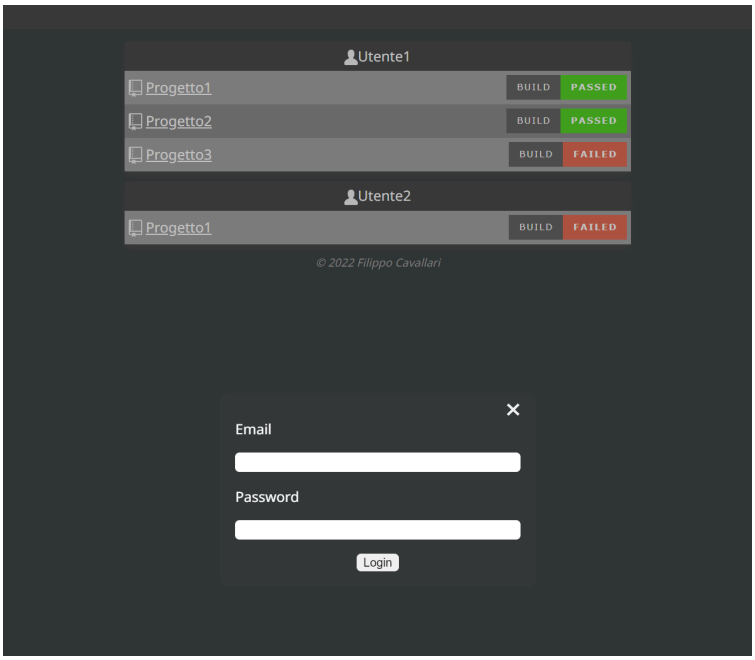

Figure 2: Login modal

il link del badge facendo tasto destro → copia link dell'immagine; consultare altre informazioni fra cui chi ha effettuato l'ultimo commit, in quale data e con quale messaggio.

Quando una build fallisce (figura 6) non sarà possibile accedere agli artefatti (se non compila non potrà aver prodotto artefatti), il colore dello status diventerà rosso e se si tratta dell'ultima compilazione allora anche il badge verrà aggiornato di conseguenza (colore rosso ed etichetta "FAILED").

0.3.2 Architettura del sistema

Frontend

Il frontend è realizzato con React; si occupa di renderizzare l'applicazione e gestire l'input dell'utente. Tutte le richieste HTTP vengono realizzate usando il modulo axios mentre per lo scambio di dati reattivo e con basso overhead viene usato socket.io.
I fogli di stile sono stati realizzati con Sass e poi compilati in automatico dall'IDE in modo da avere dei file CSS3 tradizionali.
È stata data molta importanza all'usabilità da parte di persone con problemi di vista, sia parzialmente che totalmente invalidanti; per esempio per le persone daltoniche che non riescono a distiguere il verde dal rosso, è stata scritto l'esito della build all'interno del badge (SUCCESS o FAILURE); anche la scelta di

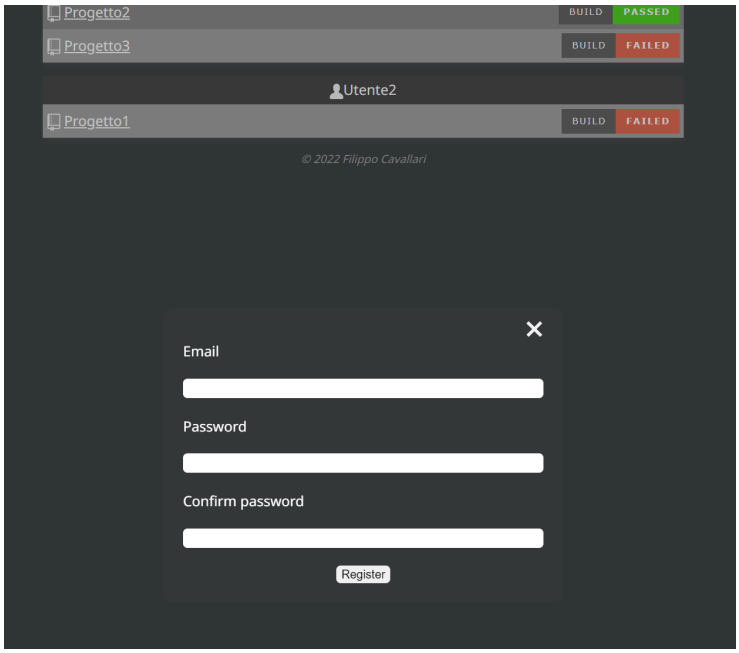

Figure 3: Register modal

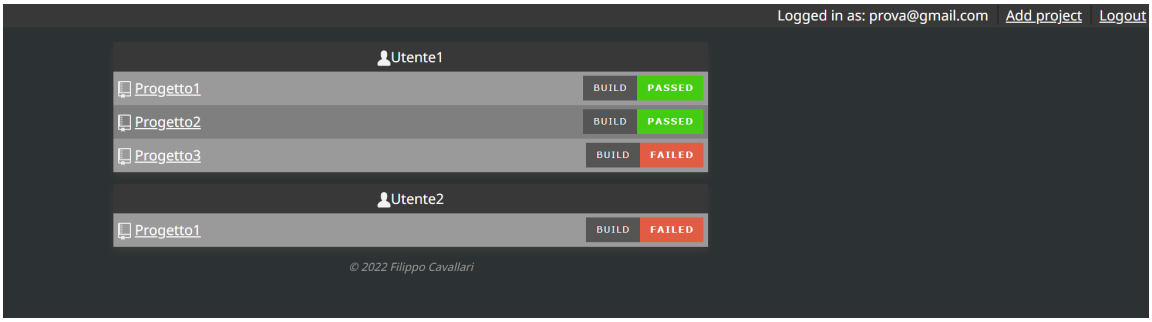

Figure 4: Post authentication home page

usare un font chiaro su sfondo scuro permette una maggiore leggibilità; inoltre tutte le immagini hanno un tag *alt* in modo che siano compatibili con lettori di testo usati da persone non vedenti.

Backend

Il backend è realizzato usando NodeJS+Express; si occupa essenzialmente di gestire il processo di compilazione dei processi e di scambiare dati con il frontend (React) attraverso gli endpoint di Express e il websocket di socket.io.

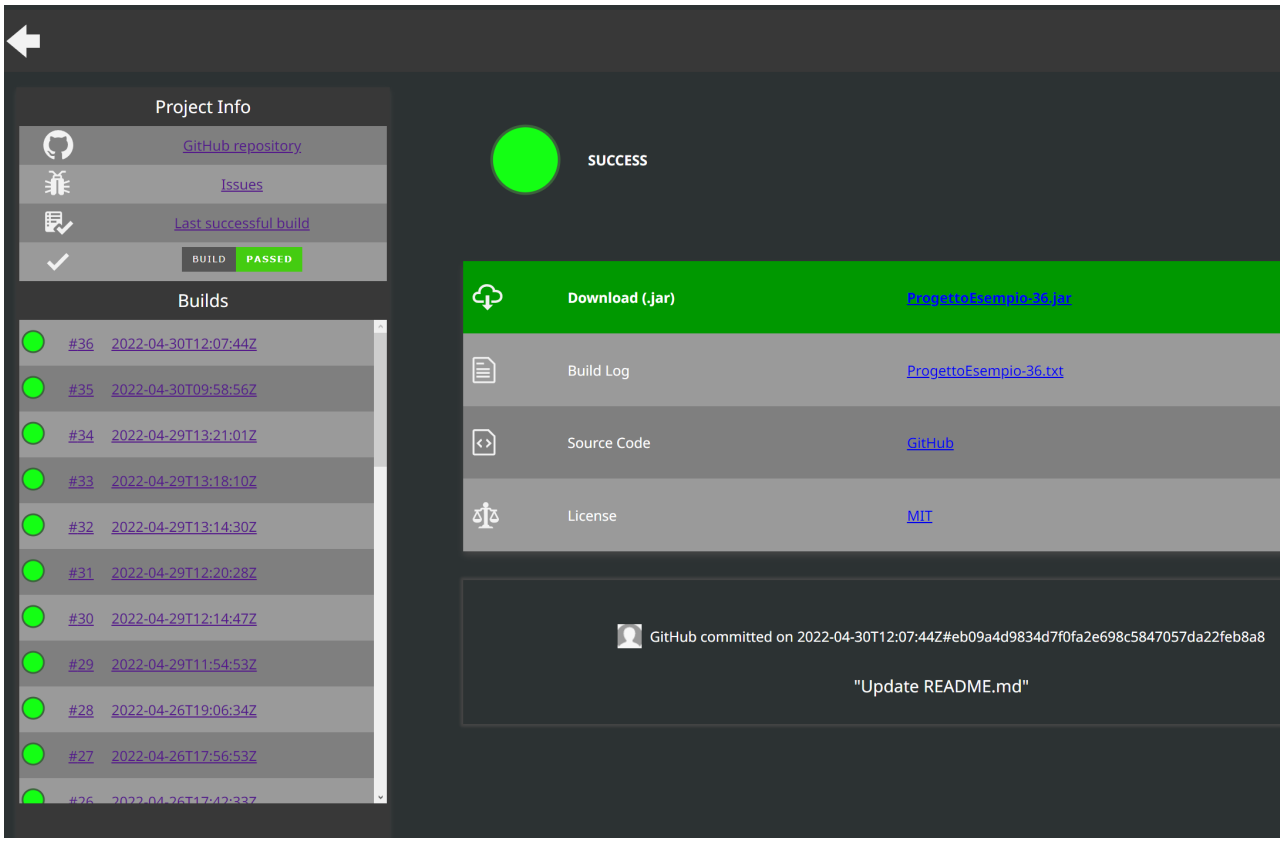

Figure 5: Project page

Andando più nel dettaglio si possono distinguere tre funzionalità chiave.

Compilazione progetti:

1. Ogni 10 minuti controlla ogni progetto per vedere se sono stati effettuati nuovi commit;
2. Se ci sono nuovi commit clona il progetto e lo compila;
3. Gli artefatti, i file di log e il badge .SVG vengono pushati su GitHub
4. Vengono aggiornate le informazioni del progetto (e.g. last build, last commit) sul DB

Autenticazione utente:

1. Le credenziali dell'utente vengono ricevute da una richiesta HTTP POST (usando il modulo *axios*) in modo che i data vengano cifrati usando il TLS;

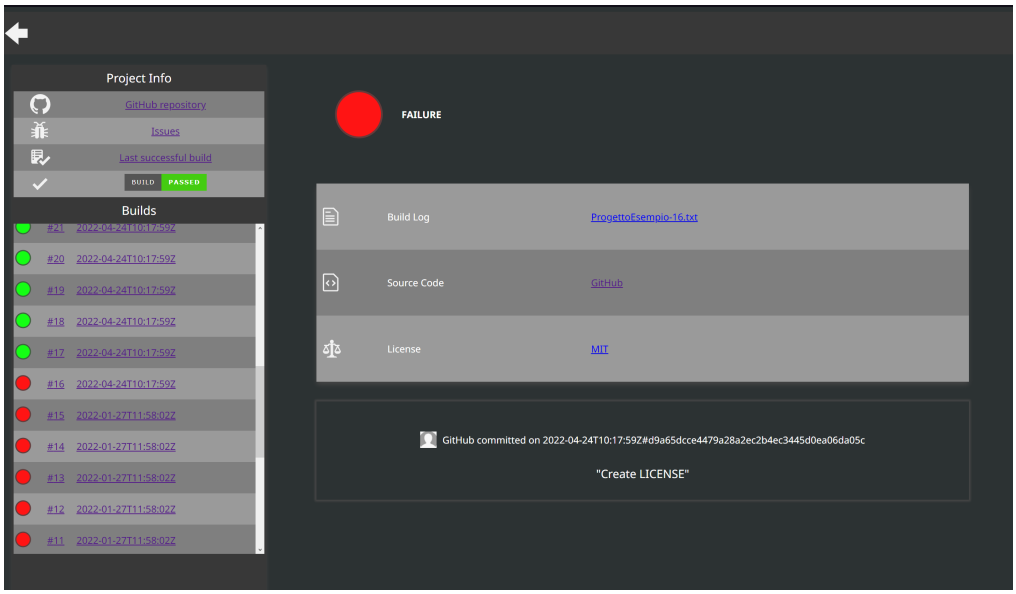

Figure 6: Failed build project page

2. Le password non sono salvate in chiaro sul DB, ma vengono cifrate usando una funzione di hash (sha512) con l'aggiunta di un sale;
3. se le credenziali fornite sono corrette, l'utente viene autenticato e viene emesso un nuovo token di autenticazionel;
4. tale token verrà inserito in un cookie *HTTP only* (per motivi di sicurezza) e scadrà dopo sette giorni;
5. per ogni richiesta HTTP verrà controllato il token di autenticazione presente nel cookie e, in caso di responso positivo, la richiesta dell'utente verrà soddisfatta.

Richiesta dati dal frontend:

1. Tutte le richieste di dati che non necessitano di autenticazione vengono effettuate usando socket.io;
2. Quando l'homepage viene caricata, React richiederà al backend la lista di tutti i progetti con tanto di badge .SVG;
3. Quando un progetto specifico viene aperto, React richiederà la cronologia di tutte le compilazioni più altre informazioni.

Docker

Tutti i componenti di Buildy sono stati dockerizzati in modo che possono essere eseguiti facilmente usando i comandi di docker-compose.

0.4 Tecnologie

Si è utilizzato come solution stack MERN. Le tecnologie utilizzate per sviluppare il sistema sono:

- Node ed Express: sviluppo componente server;
- React : sviluppo componente client;
- MongoDB: gestione persistenza dati;
- Socket.io: scambio dati real time;
- Docker: per il deploy dell'applicazione. I linguaggi utilizzati all'interno del progetto sono:
- Javascript (ES6): sia lato client che lato server;
- JSX: per la struttura delle pagine web;
- Scss: per i fogli di stile;

0.5 Codice

```
async function main(){
    projects = await getProjectsFromDb()
    await setGitIdentity()
    for (const project of projects) {
        await analyzeProject(project)
    }
    console.log("Saving projects on db")
    await saveProjectArray(projects)
}

async function analyzeProject(project){
    console.log(`Retrieving last commit for project
        ${project.projectName}`)
    let latestCommit = await getLatestCommit(project)
    let sha = latestCommit.sha
    console.log(`Retrieving repository data for project
        ${project.projectName}`)
    await project.repository.getInformation()
    if(sha !== project.latestCommitSha){
        console.log(`Cloning project ${project.projectName}`)
        await project.clone()
        console.log(`Building project ${project.projectName}`)
        let build = await project.build(latestCommit)
        console.log(`Saving build ${build.id} for project
            ${project.projectName}`)
```

```
        await project.saveBuild(build)
        console.log(`Pushing build ${build.id} for project
            ${project.projectName}`)
        await project.commitBuild(build)
    }
}
```

Le funzioni main() e analyzeProject(), entrambe asincrone, sono il fulcro del backend; la funzione main viene invocata ogni 10 minuti e si occupa di invocare la funzione *analyzeProject* su ciascun progetto registrato per poi aggiornarli nel DB; la funzione *analizeProject* è invece quella che effettua l'analisi vera e propria del progetto; l'analisi si suddivide nei seguenti step:

1. *getLatestCommit()*: recuperare lo SHA dell'ultimo commit
2. *repository.getInformation()*: recupera le informazioni sulla repository del progetto (e.g. commit messagge, commit timestamp, clone url, ecc.)
3. verifica che lo SHA dell'ultimo commit sia diverso da quello dell'ultima compilazione effettuata
4. *project.clone()*: clona il progetto
5. *project.build()*: compila il progetto
6. *project.saveBuild()*: serve a serializzare la classe Build; in altre parole crea il badge .SVG, crea i file di log e sposta gli artefatti nella destinazione finale
7. *project.commitBuild()*: carica gli artefatti e i log su GitHub

Tutte le funzioni sono asincrone.

```
function registerListeners(socketIO){
    socketIO.on('connection', (socket) => {
        socket.on('project-page-request', (message) => {
            let proj = getProjects().filter(it => it.projectName ===
                message.projectName)[0]
            socketIO.emit('project-page-response', {project: proj,
                builds: [...proj.builds].reverse(), latestBuildId:
                proj.builds.slice(-1)[0].id})
        })
        socket.on('index-page-request', ()=>{
            let authors = []
            let projects = getProjects()
            for(const proj of projects){
                let authorName = proj.repository.owner
                let filterResult = authors.filter((author) => author.name
                    === authorName)[0]
                if(filterResult === undefined){
                    let author = {name: authorName, projects: [proj]}
                    authors.push(author)
```

```
            }else{
                filterResult.projects.push(proj)
            }
        }
        socketIO.emit('index-page-response', {authors: authors})
    })
});
}
```

La funzione *registerListeners()* si occupa di registrare i listener di *socket.io*; nello specifico vengono creati due listener, uno per le richieste dei dati della homepage (*index-page-request*), ed uno per le richieste dei dati di un progetto specifico (*project-page-request*).
Quando il websocket riceve una richiesta la elabora ed emette un nuovo evento (i.e. *index-page-response* o *project-page-response*) con i dati richiesti.

```
function cloneProject(project){
    return spawn("git", ["clone", "-b", project.mainBranch,
        project.repository.cloneUrl,
        `projects/${project.repository.name}`], { stdio: 'inherit' })
}
```

La funzione cloneProject() crea un processo figlio al quale fa clonare il progetto che viene passato negli argomenti del comando git; lo stdin e lo stdout vengono ereditati dal processo padre, di conseguenza i log verranno scritti negli stream (di default il terminale) di NodeJs.

```
async build(latestCommit) {
        await
            fs.promises.chmod(`projects/${this.repository.name}/gradlew`,
            0o777)
        let gradleBuildScript = path.resolve(`src/gradle_build.sh`)
        return new Promise((resolve, reject) => {
            let log = ""
            let child = child_process('bash', [gradleBuildScript,
                this.repository.name], {shell: true})
            child.stdout.on('data', (data) => log += data)
            child.stderr.on('data', (data) => log += data);
            child.on('close', (code) => {
                if (code === 0) {
                    resolve(this.createBuild(latestCommit, true, log))
                } else {
                    resolve(this.createBuild(latestCommit, false, log))
                }
            });
        })
    }
```

La funzione *build()* si occupa di compilare un progetto; per poterlo fare aggiunge

i permessi di esecuzione ad un apposito script bash e poi lo fa eseguire ad un processo figlio; quest'ultimo avrà gli stdin e stdout indipendenti, i cui dati verranno incanalati all'interno di una variabile *log* che verrà poi passata come argomento del costruttore della classe *Build* per un successivo utilizzo.

```
async saveBuild(build) {
        this.latestCommitSha = build.commitSha
        this.wasModified = true
        build.logFileName = `${this.projectName}-${build.id}.txt`
        this.builds.push(build)
        await fs.promises.mkdir(`builds/${this.projectName}/`,
            {recursive: true})
        if (build.isSuccess) {
            console.log(this.projectName)
            let buildFolder =
                `projects/${this.repository.name}/build/libs/`
            let buildFile = (await
                fs.promises.readdir(buildFolder)).filter((allFilesPaths)
                =>
               allFilesPaths.match(/\.jar$/) !== null)[0]
            let buildPath = buildFolder + buildFile
            let splintedBuildFileName = buildFile.split(".")
            build.fileName =
                `${splintedBuildFileName[0]}-${build.id}.${splintedBuildFileName[1]}`
            await fs.promises.rename(buildPath,
                `builds/${this.projectName}/${splintedBuildFileName}`)
        }
        await
            fs.promises.writeFile(`builds/${this.projectName}/${build.logFileName}`,
            build.log, "utf-8")
        await this.createBadge(build)
        return fs.promises.rm(`projects/`, {recursive: true, force:
            true})
    }
```

La funzione *saveBuild()*, come accennato in precedenza, si occupa di "serializzare" la classe *Build*; occorre quindi che crei i file di log, che sposti gli artefatti dalla cartella di output della compilazione in una nuova destinazione, ed infine creare il badge .SVG (invocando la funzione *createBadge()*)

```
async commitBuild(build) {
        let scriptPath = path.resolve(`src/commit_build.sh`)
        return new Promise((resolve) => {
            let child = child_process("bash", [scriptPath,
                this.repository.name, build.fileName, build.logFileName,
                process.env.MYTOKEN], {stdio: 'inherit', env:
                {...process.env}})
            child.on('close', (code) => {
               if (code === 0) {
```

```
                resolve(true)
            } else {
                resolve(false)
            }
        })
    })
}
```

La funzione commitBuild() crea un processo figlio e gli fa eseguire uno script bash; quest'ultimo contiene tutti gli step per caricare su GitHub (committare e pushare) i vari file della compilazione; per poter effettuare questa operazioni occorre utilizzare un personal access token che verrà utilizzato da GitHub per autenticare la richiesta.

```
function encryptPassword(password){
    const salt = genRandomString(8)
    return sha512(password, salt)
}

function verifyPassword(passwordHash, password, salt){
    return sha512(password, salt).passwordHash === passwordHash
}

const genRandomString = function (length) {
    return crypto.randomBytes(Math.ceil(length / 2))
        .toString('hex') /** convert to hexadecimal format */
        .slice(0, length); /** return required number of characters */
};

const sha512 = function(password, salt){
    const hash = crypto.createHmac('sha512', salt); /** Hashing
        algorithm sha512 */
    hash.update(password);
    const value = hash.digest('hex');
    return {
        salt:salt,
        passwordHash:value
    };
};
```

La funzione *encryptPassword()* riceve in input una password in chiaro e restituisce l'output della funzione di hash *sha512* con l'aggiunta di un sale; in questo modo anche in caso di furto dell'hash della password, senza avere accesso al sale sarà impossibile risalire alla password originale. La funzione *verifyPassword()* controlla che l'hash della password in chiaro con l'aggiunta del sale sia uguale all'hash passato come parametro (normalmente l'hash salvato sul DB). La funzione *getRandomString()* fornisce una stringa di lunghezza N (dove N è un parametro) da usare per la generazione del sale. La funzione *sha512()* è la

funzione di hash che accetta come parametri la password in chiaro ed il sale.

0.6 Test

0.6.1 Postman

Per testare il corretto funzionamento degli endpoint di Express è stato utilizzato Postman Desktop come mostrato in figura 7.

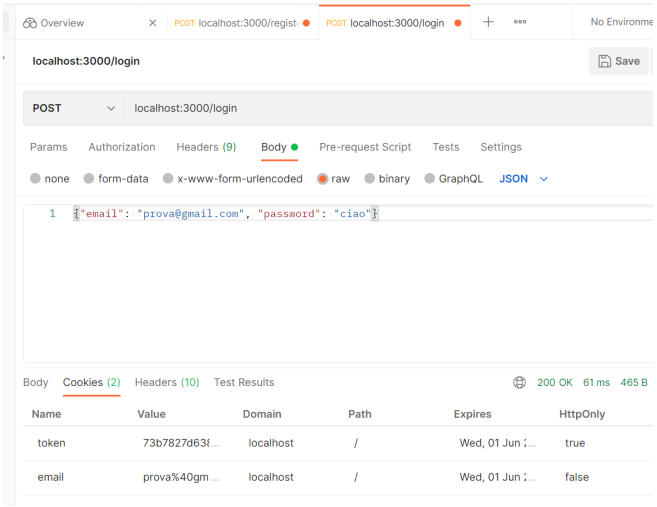

Figure 7: Postman test

0.6.2 Utente

Buildy è stato testato da altri studenti che hanno aggiunto i propri progetti e sono rimasti molto soddisfatti; in particolare hanno apprezzato:

- la semplicità d'utilizzo
- l'assenza di bug
- il design dell'interfaccia utente

I nomi degli studenti non verranno scritti per loro richiesta ma hanno acconsentito ad essere nominati verbalmente.

0.7 Deployment

Requisiti: [*Docker*, *Docker-compose*] Per installare Buildy è sufficiente clonare la repository ed eseguire il comando *docker-compose up –build -d*.

Prima di eseguire il comando è necessario creare un file .env nella root del progetto con le seguenti variabili: Installare una copia

```
MYTOKEN=<your_github_personal_access_token>
MONGO_IP=mongodb #You can leave that unchanged, unless you modify the
    MongoDB container's name
NODE_IP=nodejs #You can leave that unchanged, unless you modify the
    NodeJS container's name
REACT_IP=localhost #You can leave that unchanged, unless your react
    container is on a different IP
REPO_URL=github.com/Filocava99/Buildy.git #Set your repository URL (used
    for storing artifacts and log files)
```

Vanno inoltre modificate le variabile in frontend/src/settings.js in modo che combacino con le variabili d'ambiente e le vostre eventuali preferenze.
È altamente raccomandato di modificare le credenziali dell'utente root di MongoDB all'interno del file docker-compose.yml per ovvi motivi di sicurezza.
Una volta avviato, Buildy sarà raggiungibile sulla porta 80 del vostro dispostivo (http://localhost se l'accesso viene effettuato in locale); il backend utilizza la porta 3000 per Express e la porta 3001 per socket.io.

0.8 Conclusioni

Sono personalmente molto soddisfatto da come ho realizzato Buildy; ho in mente diverse modifiche che effettuerò in futuro, fra cui aggiungere il supporto a progetti Maven. Ho anche realizzato una versione basata interamente su NodeJs hostata tramite GitHub pages che può essere facilmente installa forkando la repository. Spero che possa nascere una piccola community di utilizzatori per entrambi le versioni; proprio per questo ho realizzato un README abbastanza dettagliato e sto lavorando ad una guida su come contribuire ai progetti.